

DISEÑO DE UN SISTEMA BIOMÉTRICO PARA EL RECONOCIMIENTO DE
HUELLAS DACTILARES

JULIAN DAVID MANTILLA BLANCO

UNIVERSIDAD AUTÓNOMA DE BUCARAMANGA

FACULTAD DE INGENIERÍA FISICOMECAÑICAS

PROGRAMA DE INGENIERÍA MECATRÓNICA

BUCARAMANGA

2016

DISEÑO DE UN SISTEMA BIOMÉTRICO PARA EL RECONOCIMIENTO DE
HUELLAS DACTILARES

JULIAN DAVID MANTILLA BLANCO

LINEA DE INVESTIGACION:
DISEÑO MECATRÓNICO Y ROBÓTICA

DIRECTOR:
M.Sc. HERNANDO GONZÁLEZ ACEVEDO
INGENIERO ELECTRÓNICO

UNIVERSIDAD AUTÓNOMA DE BUCARAMANGA
FACULTAD INGENIERÍA MECATRÓNICA
BUCARAMANGA

2016

Nota de aceptación

Aprobado por el comité de grado en cumplimiento de los requisitos exigidos por la Universidad Autónoma de Bucaramanga para optar el título de Ingeniero Mecatrónico.



HERNAN GONZALEZ ACUÑA

Jurado

HERNANDO GONZALEZ ACEVEDO

Bucaramanga, 05 de Diciembre de 2016.

AGRADECIMIENTOS

Llegando al final de un camino largo y con algunos tropiezos, es fundamental agradecer a quienes hicieron parte de este. Mis padres, quienes, con su esfuerzo, enseñanzas y comprensión para el desarrollo de mis estudios, me dejaron la herencia más valiosa que pudiera recibir. Mis hermanos. Cada uno a su manera, supo darme el apoyo familiar que por momentos necesite.

A todas las personas que conocí en este trayecto, quienes de una forma u otra me enseñaron algo y me ayudaron a convertirme en la persona que soy hoy. A Jessica y a Olmer, por tantas alegrías y momentos compartidos. A Katherin, Cristian y Aldair, por esa disposición de ayudarme incondicionalmente siempre que los necesite. A todos mis demás amigos que siempre estuvieron dispuestos a escucharme.

Mis profesores, Hernando González y Hernán González. Me hicieron recordar, por qué elegí esta profesión. A Nayibe, por todas las cosas que me enseñó y por todo el apoyo que me brindó durante la carrera. En pocas palabras, a ellos tres, por haberme moldeado como profesional y como persona para mi futuro.

Por último, pero no menos importante, a Dios por poner a estas maravillosas personas en mi camino, y por mostrarme que sus tiempos son perfectos, y que, aunque a veces las cosas se vean mal, debemos confiar en que el guarda lo mejor para nosotros.

Gracias, totales.

DEDICATORIA

“A mi madre, con el tributo pleno de mi amor ardentísimo; A ella, faro en mis tinieblas, puerto en mis naufragios, caridad y bálsamo en el dolor cruel de mis heridas”

Jorge Eliecer Gaitán Ayala

CONTENIDO

1. OBJETIVOS	12
1.1. OBJETIVO GENERAL	12
1.2. OBJETIVOS ESPECÍFICOS	12
2. SISTEMA BIOMÉTRICO	13
2.1. HUELLA DACTILAR	14
2.1.1. CARACTERÍSTICAS FUNDAMENTALES	14
2.2. MINUCIAS	15
2.3. SISTEMA DE ADQUISICIÓN.....	16
2.4. SISTEMA EMBEBIDO	17
2.4.1. PHYTON	20
2.4.2. OPEN CV	21
2.4.3. QT CREATOR.....	21
2.4.4. SQL	22
3. PROCESAMIENTO DE LA HUELLA DIGITAL	24
3.1. PREPROCESAMIENTO.....	25
3.1.1. NORMALIZACIÓN.....	30
3.1.2. SEGMENTACIÓN	31

3.1.3.	ESTIMACIÓN DE LA ORIENTACIÓN Y LA FRECUENCIA	33
3.1.4.	FILTRO GABOR.....	38
3.1.5.	BINARIZACIÓN.....	41
3.1.6.	PRUEBAS	42
3.2.	ADELGAZAMIENTO Y DETECTOR DE MINUCIAS.....	45
3.2.1.	ADELGAZAMIENTO	46
3.2.2.	DETECTOR DE MINUCIAS	48
3.3.	COMPARACIÓN.....	49
4.	IMPLEMENTACIÓN DEL SISTEMA BIOMÉTRICO	53
4.1.	INTERFAZ GRAFICA.....	54
4.2.	VALIDACIÓN DEL SISTEMA.....	58
5.	CONCLUSIONES.....	68

LISTA DE IMÁGENES

Figura 1. Crestas y valles de una huella dactilar.....	15
Figura 2. Minucias de la huella dactilar.	16
Figura 3. Comparación de los modelos de Raspberry.	19
Figura 4. Diagrama general del sistema embebido.....	20
Figura 5. Filtro Notch en una imagen.	27
Figura 6. Normalizacion de una imagen.....	31
Figura 7. Diagrama para la segmentacion de una imagen.....	32
Figura 8. Segmentacion aplicada en la imagen.	33
Figura 9. Estimacion de la orientacion aplicada en la imagen.....	35
Figura 10. Estimacion de la frecuencia de una imagen.....	37
Figura 11. Estimacion de la frecuencia aplicada en la imagen.....	38
Figura 12. Filto Gabor de una dimension	39
Figura 13. Filtro Gabor en dos dimensiones.	40
Figura 14. Filtro Gabor aplicado a la imagen.	41
Figura 15. Binarizacion aplicada a la imagen.....	42
Figura 16. Ventana de 3x3 pixeles para el adelgazamiento.....	46
Figura 17. Resultado del adelgazamiento	48
Figura 18. Ventana de 3x3 pixeles para la deteccion de minucias.....	48
Figura 19. CrossNumber de una bifurcacion y una terminacion.....	49
Figura 20. Metodo de comparacion basado en minucias.....	51

Figura 21. Vista inicial de la GUI.	54
Figura 22. Vista administrador de la GUI.	55
Figura 23. Vista nuevo registro de la GUI.	55
Figura 24. Eliminar registro de la GUI.	56
Figura 25. Vista nuevo administrador de la GUI.....	56
Figura 26. Vista ingreso de la GUI.	57

LISTA DE TABLAS

Tabla 1. Características técnicas del UareU4500.	17
Tabla 2. MSE y PSNR para el pre-procesamiento de la imagen.....	28
Tabla 3. Resultado de la implementación de los filtros.	28
Tabla 4. Resultados de la normalización aplicada a la imagen.....	43
Tabla 5. Resultados de la segmentación aplicada a la imagen.....	44
Tabla 6. Información de las minucias.....	50
Tabla 7. Algoritmos realizados para el desarrollo del sistema biométrico.	54
Tabla 8. Pruebas y validación del funcionamiento del sistema biométrico.....	59
Tabla 9. Base de huellas utilizadas para la validacion del sistema.....	61
Tabla 10. Resultados de las pruebas de validacion	62
Tabla 11. Comparacion del procesamiento de la imagen para un falso negativo.	64
Tabla 12. Resultados falsos de la huella.....	67
Tabla 13. Resultados positivos de la huella.....	68

RESUMEN

La biometría está adquiriendo una gran importancia en el tema de la seguridad, ya que son procesos tecnológicos de difícil filtración, y utilizan la particularidad de las características biológicas, que son únicas para cada individuo. De igual manera, se han venido implementando diferentes técnicas de visión artificial que permiten mejorar la precisión de estos sistemas. Este trabajo de investigación, desarrolla un algoritmo de procesamiento de una huella dactilar, a partir de técnicas encontradas en la literatura como el filtro Gabor, para posteriormente ser usado en un sistema embebido.

PALABRAS CLAVE: biometría, visión artificial, Gabor, huella dactilar, seguridad.

Biometrics is acquiring importance in security systems because they are difficult to hack, for it uses the particularity of the biological characteristics of everyone, which are unique to each one. Similarly, they have been implementing various techniques of computer vision to improve the accuracy of these systems. This research develops a processing algorithm of a fingerprint, from the latest techniques applied in the literature, such as the Gabor filter, later to be used in an embedded system.

KEYWORDS: Biometrics, computer vision, Gabor, fingerprint, security.

1. OBJETIVOS

1.1. OBJETIVO GENERAL

Diseñar un algoritmo de procesamiento de huellas digitales para un sistema biométrico que permita determinar la identidad de un individuo.

1.2. OBJETIVOS ESPECÍFICOS

- Desarrollar un algoritmo de pre procesamiento basado en el dominio espacial, para eliminar el ruido presente en la imagen de una huella dactilar.
- Desarrollar un algoritmo de procesamiento de imágenes que permita identificar minucias en una imagen de huella dactilar.
- Desarrollar un algoritmo que permita comparar las minucias detectadas en una huella dactilar con información almacenada.
- Programa un sistema embebido para el reconocimiento de personas a partir de la huella dactilar.

2. SISTEMA BIOMÉTRICO

Se entiende biometría como la ciencia que analiza y mide, ciertas características biológicas y particulares, las cuales se utilizan para generar un identificador único para cada individuo.

Por ende, entenderemos por sistema biométrico a un sistema automatizado que realiza labores de biometría. Es decir, un sistema que fundamenta sus decisiones de reconocimiento por medio de una característica personal que puede ser reconocida o verificada de manera automatizada [1].

Un sistema biométrico por definición, es un sistema capaz de:

- Obtener la muestra biométrica del usuario.
- Extraer los datos de la muestra.
- Comparar los datos obtenidos con los existentes en la base de datos.
- Indicar el resultado de la verificación.

2.1. HUELLA DACTILAR

Una huella dactilar es una representación de la forma de la piel de las yemas de los dedos, que se produce cuando se presionan los dedos sobre una superficie lisa, se trata de un patrón, único y diferente del dedo humano.

Las huellas digitales se encuentran completamente formadas alrededor de los siete meses de gestación y este patrón permanecerá invariable durante toda la vida del individuo, salvo el caso de accidentes como heridas o cortes graves, cuyas cicatrices pasarían a ser parte de la identidad de la persona [1].

2.1.1. CARACTERÍSTICAS FUNDAMENTALES

- Una de las características principales de las huellas digitales, es su individualidad, ya que se considera, con fuertes evidencias, que las huellas digitales son diferentes de persona a persona, e incluso un mismo individuo posee huellas diferentes en cada uno de los dedos de sus manos. Esta característica permite el uso de las huellas digitales como uno de los métodos de reconocimiento más usados en muy diversas aplicaciones.
- La característica más evidente de una huella es un patrón de crestas y valles intercalados entre sí, que aparecen en las imágenes como partes

oscuras y claras respectivamente [1]. En la figura 1 podemos ver las crestas representadas por líneas oscuras y los valles como las partes claras de la huella.

Figura 1. Crestas y valles de una huella dactilar.



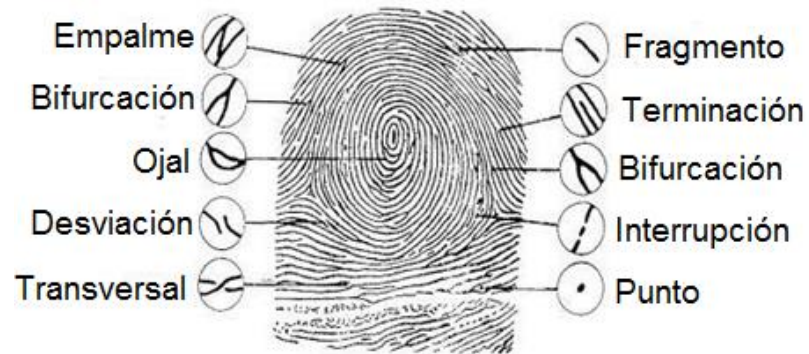
Fuente: Verificación de correspondencia en huellas dactilares aplicando técnicas de procesamiento y análisis digital de imágenes para la disminución del tiempo de cotejo [2].

2.2. MINUCIAS

En un nivel más detallado, se denotan otras características importantes dentro de los patrones digitales, conocidas como minucias. Las minucias se refieren a las diferentes formas en que las crestas pueden ser discontinuas. Por ejemplo, una cresta puede súbitamente finalizar (terminación), o puede dividirse en dos crestas independientes (bifurcación) [2]. Estas minucias son las más usadas en

los sistemas biométricos y a partir de estas minucias principales, podemos encontrar las minucias que se presentan en la figura 2.

Figura 2. Minucias de la huella dactilar.



Fuente: Verificación de correspondencia en huellas dactilares aplicando técnicas de procesamiento y análisis digital de imágenes para la disminución del tiempo de cotejo [2].

2.3. SISTEMA DE ADQUISICIÓN

Para el sistema de adquisición, se determinó utilizar el sensor Digital Personal UareU4500 [3], siendo este el sensor de huella más utilizado y más comercializado en el mercado. Arroja tomas con una resolución de 512 dpi y tiene conexión USB, lo que facilita la conexión con la placa. Los dpi (Dots Per Inch) son una medida utilizada para determinar la calidad de digitalización de un parámetro físico, en este caso la huella dactilar. En la siguiente tabla encontramos las características técnicas de este sensor.

Voltaje de alimentación	5.0V \pm 5% de alimentación por USB
Corriente de alimentación-escaneo	< 100mA (Típicamente)
Corriente de alimentación-stand by	<0.5mA (Típicamente)
Temperatura de operación	0 – 40 C
Scan Data	8-bit grayscale
Resolucion de pixel	512 dpi

Tabla 1. Características técnicas del UareU4500.

2.4. SISTEMA EMBEBIDO

Un sistema embebido se puede definir como un sistema electrónico diseñado específicamente para realizar unas determinadas funciones [4] . La característica principal es que emplea para ello uno o varios procesadores digitales (CPU's) en formato microprocesador o microcontrolador, lo que le permite aportar “inteligencia” al sistema anfitrión al que ayuda a gobernar y del que forma parte. Para este caso se decidió utilizar una de las plataformas de desarrollo que han tenido gran acogida por parte de los desarrolladores.

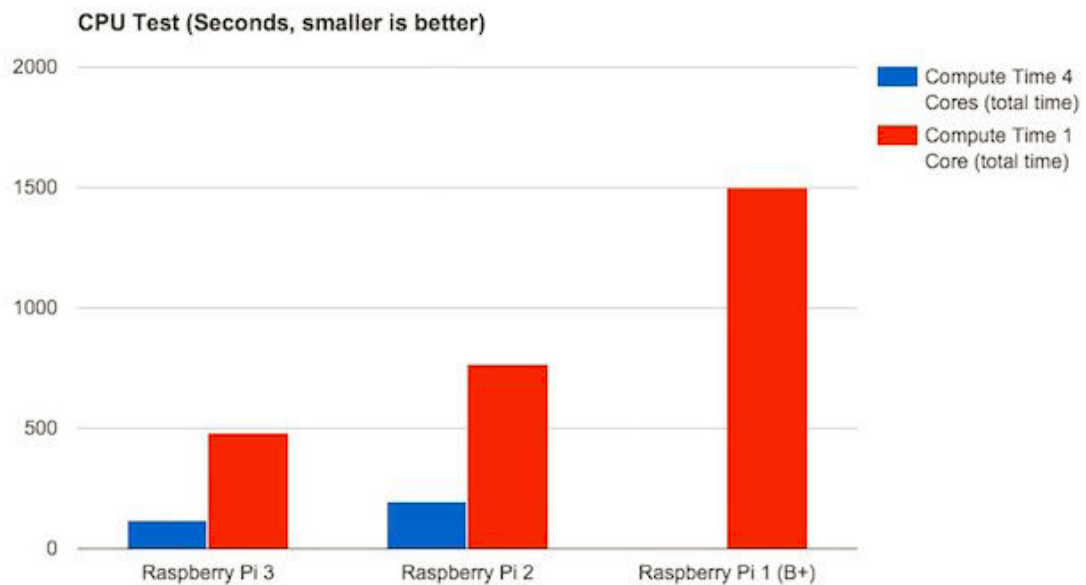
En la selección de la placa se tuvo en cuenta el trabajo realizado por Casco [5], quien realizó un estudio de comparación entre un Arduino, un Beaglebone Black y una Raspberry Pi B. Para empezar, se descarta la tarjeta Arduino ya que no cuenta con los requerimientos de velocidad ni de RAM necesarios para un

proyecto de estas características. Se tiene en cuenta las recomendaciones realizadas por el autor en las que indica que la tarjeta Raspberry es ideal para proyectos de multimedia complejos o basados en Linux o para el desarrollo de proyectos que requieran una interfaz gráfica y varias entradas seriales, y la Beaglebone preferible en los casos en que se necesiten más pines de propósito general y networking.

Otra ventaja importante para la raspberry sobre la beaglebone en este proyecto en particular, es que la raspberry tiene una mejor retroalimentación de sus usuarios en la red, facilitando el desarrollo de proyectos. El precio inferior de la Raspberry es otro factor a considerar.

En el mercado podemos encontrar diferentes tipos de tarjetas Raspberry, por lo que también se analizaron las tres últimas tarjetas desarrolladas, la Raspberry Pi B+, la Raspberry Pi 2 y la Raspberry Pi 3. En este caso tomamos un reportaje de la web de tecnología y diseño, gizmodo, en el cual hacen un test de CPU utilizando sysbench [6], la cual es una herramienta multiplataforma para evaluar el desempeño de diferentes sistemas. Como podemos observar de la siguiente gráfica, la Raspberry Pi 3 es mucho más rápida que sus predecesoras, incluso utilizando uno solo de sus cuatro núcleos.

Figura 3. Comparación de los modelos de Raspberry.

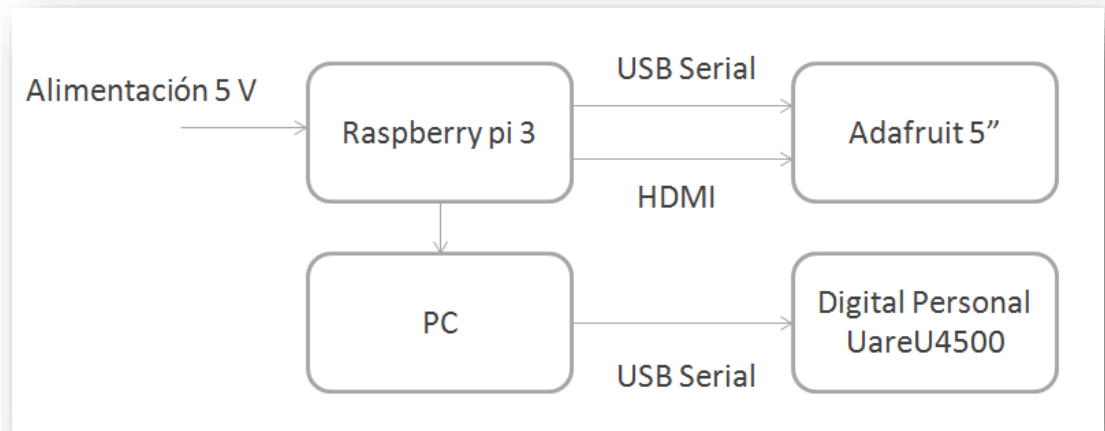


Fuente: El rendimiento de la Raspberry Pi 3, comparado contra los modelos anteriores [5].

Por último, como se decidió utilizar la Raspberry Pi 3, se realizó una búsqueda entre los productos disponibles de Adafruit para la tarjeta. Aquí encontramos una pantalla táctil de 5" que se conecta por HDMI y tiene alimentación independiente. Al ser una pantalla táctil con una buena calidad de imagen, mejora la interacción del sistema con el usuario, haciéndola más dinámica.

En la figura 4 podemos encontrar la conexión de los componentes que conformar el sistema. El sensor va conectado al computador para la adquisición de la imagen, la cual se pasa a la raspberry, que está conectada a su alimentación y a la pantalla HDMI táctil.

Figura 4. Diagrama general del sistema embebido.



2.4.1. PHYTON

Python es un lenguaje de programación de alto nivel, orientado a objetos y con una semántica dinámica. Su facilidad de programación lo hace atractivo para el desarrollo rápido de aplicaciones, o para utilizarlo como lenguaje para conectar componentes existentes. La sintaxis sencilla y fácil de aprender de Python hace hincapié en la legibilidad y por lo tanto reduce el costo del mantenimiento del programa. El intérprete de Python y la amplia biblioteca estándar están disponibles en forma de código fuente o binario sin ningún cargo y se pueden distribuir libremente [7] .

Python es un lenguaje diseñado para facilitar el aprendizaje al momento de empezar con la programación, y está diseñado para ser de fácil comprensión. Es

uno de los lenguajes predeterminados en el sistema operativo de la Raspberry, y gracias a su auge, es uno de los lenguajes más utilizados en la actualidad, lo que facilita el encontrar información de librerías y códigos en la red.

2.4.2. OPEN CV

Open Source Computer Vision Library (OpenCV) es una biblioteca de software de vision artificial y machine learning. Fue construido para proporcionar una infraestructura común para aplicaciones de vision artificial. Esta biblioteca cuenta con más de 2500 algoritmos optimizados. Se utiliza para procesar imágenes o videos con diferentes propósitos y por parte de grandes empresas y gobiernos de todo el mundo. Tiene interfaces C ++, C, Python, Java y MATLAB y soporta Windows, Linux, Android y Mac OS. La librería se usa principalmente en aplicaciones de visión en tiempo real [8].

2.4.3. QT CREATOR

Es un framework multiplataforma orientado a objetos ampliamente utilizado para desarrollar programas que utilicen una interfaz gráfica de usuario. Es desarrollado como un software libre y de código abierto de la cual participan la comunidad en general, pero también desarrolladores de Nokia y otras grandes empresas. Está basado en C++ de forma nativa y puede ser utilizado en varios

otros lenguajes de programación a través de enlazadores, como PyQt en este caso, que nos permite tomar el archivo generado por el diseño de la interfaz gráfica y guardarlo como un código de Python que puede ser llamado luego.

2.4.4. SQL

SQL (Structured Query Language) es un lenguaje declarativo de acceso a bases de datos relaciones que permite especificar diversos tipos de operaciones en ellas. Basado originalmente en el álgebra relacional, SQL está diseñado para la definición, manipulación y control de los datos presentes en una base de datos.

SQLite es un sistema de manejo de base de datos relacional contenida en una biblioteca de programación. A diferencia de otros sistemas de manejo de bases de datos, sqlite no necesita un servidor donde esté ubicada la base de datos. Por el contrario, esta embebida en el programa final. SQLite lee y escribe directamente de un archivo de disco, y está en un formato multiplataforma.

SQLite es una elección popular como base de datos embebida y es probablemente el motor de base de datos más utilizado en el mundo, ya que es ampliamente usado por browsers, sistemas operativos y sistemas embebidos, entre otros. Adicional a lo anterior, otro beneficio de SQLite es que cuenta con enlaces para diferentes lenguajes de programación.

En este caso en particular se trabajó con la librería para Python 'sqlite3', que permite hacer queries o consultas a base de datos sin problema desde Python [9].

3. PROCESAMIENTO DE LA HUELLA DIGITAL

Entendiendo procesamiento como cualquier ordenación o tratamiento de datos, o los elementos básicos de información, mediante el empleo de un sistema, logrando así una transformación sobre los datos, convirtiéndola en información. En este caso en particular se realiza una transformación de los datos obtenidos mediante el sensor de huella y los estamos convirtiendo en información útil, como las características particulares e individuales de las personas que utilicen este sistema.

En este proceso es importante resaltar 3 grandes etapas del manejo de los datos. Inicialmente se realiza el pre procesamiento que me permite una mejor respuesta de las siguientes etapas. Luego tenemos el adelgazamiento y la detección de minucias, donde se identifican las partes de la huella con minucias, que en nuestro caso es la información que buscamos. Por último, tenemos la etapa de comparación, que, utilizando la información generada de la fase anterior, me permite determinar si una huella pertenece o no al mismo individuo. A continuación, abordaremos a detalle estas tres etapas del procesamiento.

3.1. PREPOCESAMIENTO

En esta etapa se le realiza un filtrado a la imagen para eliminar basura en la toma y resaltar las crestas y minucias formadas por las mismas. Inicialmente se tuvo en cuenta el trabajo de Kazi [10], quien tomo una misma huella como base y la analizo con los métodos de ecualización de histograma, transformada de Fourier y el filtro Gabor, cada uno por separado, para determinar con cuál de los métodos se obtenían mejores resultados por medio de la implementación de las pruebas Peak Signal-to-noise ratio (PSNR) y el Mean Square Error (MSE).

El Peak Signal-to-noise ratio representa la medida del pico de error entre dos imágenes y es usualmente utilizada para determinar la calidad entre la imagen reconstruida y la imagen original, entre más alto el valor del PSNR, mejor calidad tiene la imagen procesada y el Mean Square Error (MSE) representa el error acumulado al cuadrado entre la imagen procesada y la original, a diferencia del PSNR, entre más bajo el valor de MSE, representa una mejor calidad en la salida del filtro. En la ecuación 1 encontramos el MSE, donde g es la imagen original, \hat{g} es la imagen procesada y (i, j) representan los pixeles de cada imagen.

$$MSE = \sum_{i=1}^I \sum_{j=1}^I [\hat{g}(i, j) - g(i, j)]^2 \quad (1)$$

Por otro lado, encontramos el cálculo del PSNR en la ecuación 2. En esta el valor de S representa el valor de pixel máximo en la imagen.

$$PSNR = -10 \log_{10} \frac{MSE}{S^2} \quad (2)$$

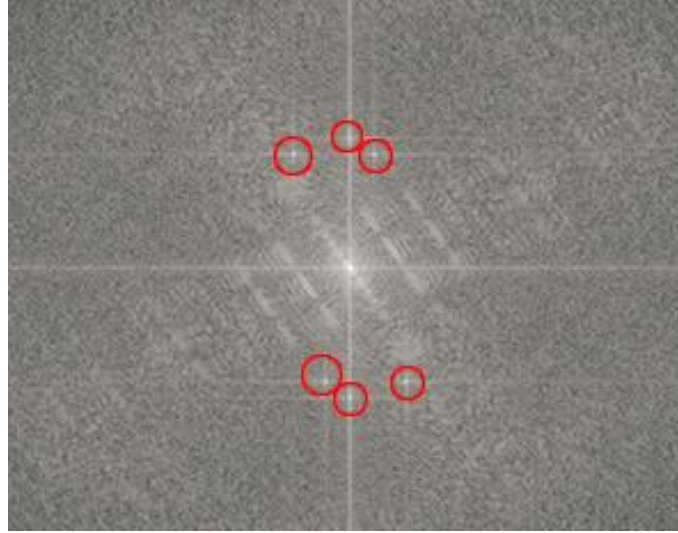
Con esto Kazi determino que el mejor algoritmo de pre-procesamiento de los que él evaluó, es el filtro Gabor.

Tomando esto como referencia, se decidió probar el filtro Gabor con otros métodos de filtrado adicional, buscando determinar con cual se obtienen mejores resultados.

Se utilizó el filtro Gabor agregándole un filtro Notch, un filtro pasa-altos y por último utilizando únicamente el filtro Gabor. El filtro Notch y el filtro pasa-altos se trabajan en el dominio de la frecuencia, por lo que primero se debe aplicar la transformada de Fourier a la imagen.

El filtro Notch, está diseñado para eliminar ruidos repetitivos en las imágenes, por lo que se debe tomar la imagen en el dominio de la frecuencia y eliminar el ruido que se representa como picos altos (puntos blancos en el dominio de la frecuencia), como se observa en la figura 5:

Figura 5. Filtro Notch en una imagen.



En el caso del filtro pasa altos Butterworth, es un filtro que deja pasar las frecuencias altas, atenúa las frecuencias que están por debajo de la frecuencia de corte y está dado por la siguiente ecuación (Ecuación 3):

$$H(i, j) = \frac{1}{1 + [Do / \sqrt{i^2 + j^2}]^{2n}} \quad (3)$$



Donde Do es el radio de corte y n , el orden del filtro. Estas pruebas se realizaron utilizando la herramienta Matlab con funciones presentes en el programa y en la página de desarrolladores de Matlab, se implementaron estos filtros previos, al filtrado con Gabor. En el caso del filtro pasa altos se hizo un filtro de orden 2 con un radio de corte del 10%.

Estas pruebas se evaluaron usando los mismos criterios de utilizados por Kazi [10], es decir el MSE y el PSNR. La prueba se realizó utilizando una sola huella

encontrada en la red y con el código anexo Q, arrojando los siguientes resultados de calidad en la tabla 2 y los resultados gráficos en la tabla 3:

	MSE	PSNR
Gabor	2.1068e-06	5.7509
Notch-Gabor	0.0014	5.9086
Pasa Altos-Gabor	8.2007e-05	10.0667

Tabla 2. MSE y PSNR para el pre-procesamiento de la imagen.

	Filtro Notch y filtro Gabor
	Filtro Pasa Altos y filtro Gabor

	<p>Filtro Gabor</p>
---	----------------------------

Tabla 3. Resultado de la implementación de los filtros.

Como podemos ver, al agregarle el filtro Notch o el filtro pasa-altos al proceso, se genera un ruido alrededor de la zona de trabajo lo que representa un problema al momento de realizar la identificación, puesto que este, arrojaría minucias falsas. Por otro lado, la agregación de filtros al proceso, también hace que el proceso del filtrado tome más tiempo, lo que es un problema a la hora de cumplir los requerimientos.

En vista de lo anterior se determina utilizar el filtro Gabor planteado por Thai [11], el cual aplica el procedimiento sin agregarle ningún proceso de filtrado previo o posterior.

3.1.1. NORMALIZACIÓN

La normalización es utilizada para estandarizar los valores de intensidad en una imagen, ajustando el rango de escala de grises a unos valores dentro de un rango deseado [11]. Esto generalmente es conocido como un método para mejorar el contraste de las imágenes y no cambia la estructura de las crestas, si no que estandariza los niveles dinámicos de variación en los valores de los pixeles, lo que facilita el proceso en las siguientes etapas.

Si tomamos entonces $I(i,j)$ como el valor en escala de grises en el pixel (i,j) y $N(i,j)$ representa el pixel normalizado en (i,j) . La imagen normalizada se define de la siguiente manera (Ecuación 1):

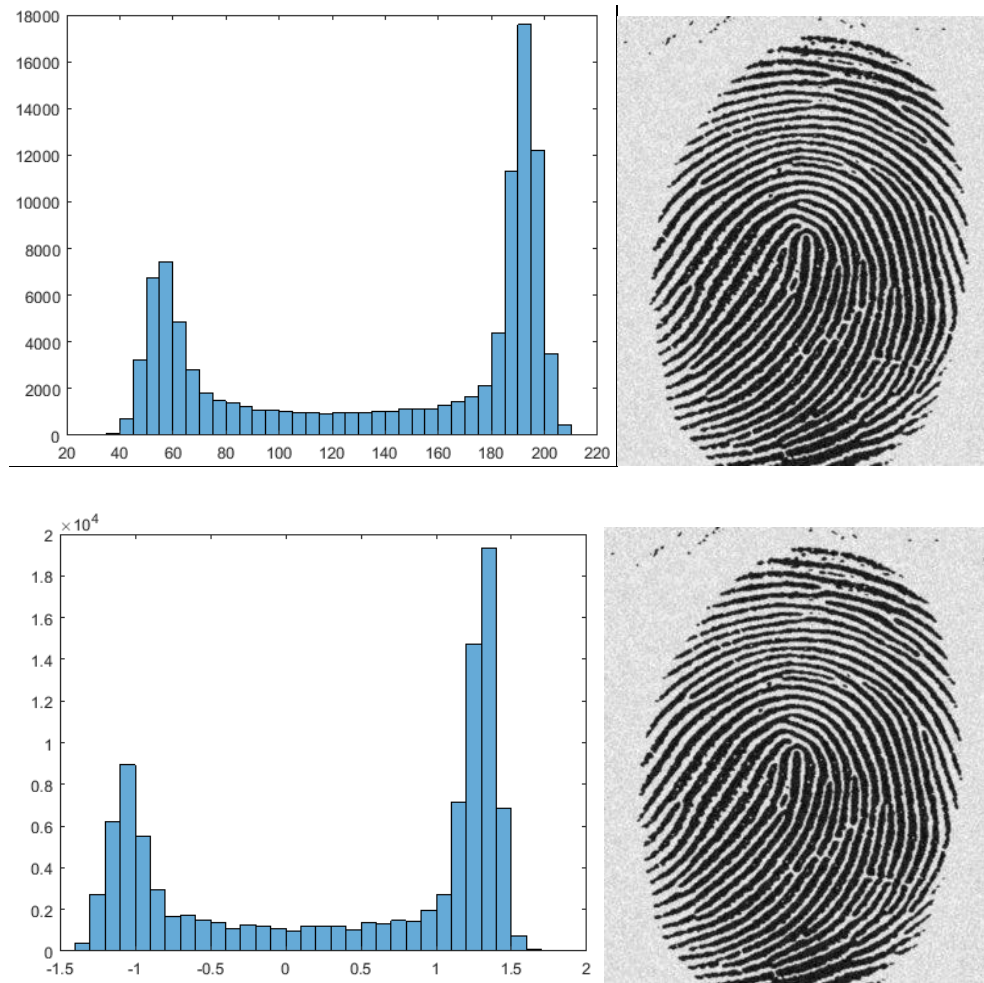
$$N(i,j) = \begin{cases} M_0 + \frac{\sigma_0(I(i,j)-M)}{\sigma} & \text{Si } I(i,j) > M \\ M_0 - \frac{\sigma_0(I(i,j)-M)}{\sigma} & \text{Si } I(i,j) < M \end{cases} \quad (4)$$

Donde M y σ son la media y desviación estándar estimada de $I(i,j)$, respectivamente, y M_0 y σ_0 representan los valores deseados de media y desviación estándar. Este método consiste en la estandarización de los valores de la imagen para su posterior normalización.

Como vemos en la figura 6, los valores de los pixeles cambian y hacen que la imagen tenga un mejor contraste. En caso de que la calidad del sensor no sea

buena o que el usuario no ponga su huella con suficiente presión, los cambios serán más notorios.

Figura 6. Normalización de una imagen.



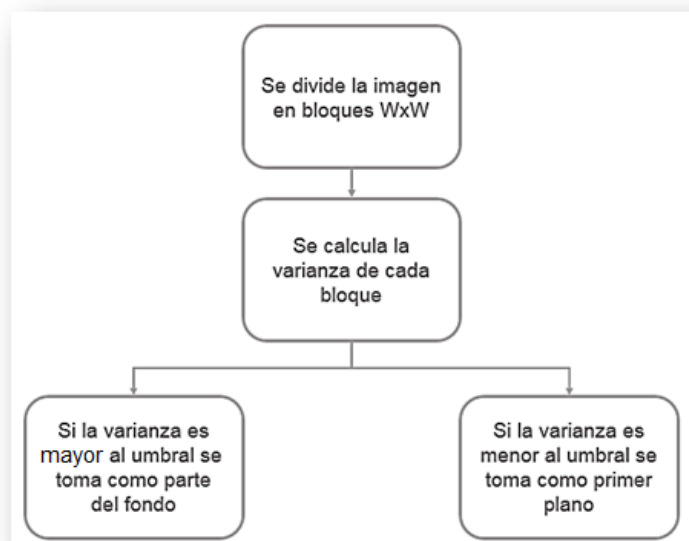
3.1.2. SEGMENTACIÓN

La segmentación es el proceso de separar el fondo de la imagen, de la región de importancia, es decir, la parte de la imagen que contiene información de la huella.

Este primer plano corresponde al área que contiene las crestas y valles de la muestra, es decir la zona de interés [11]. El fondo, por otro lado, corresponde a la región fuera de los bordes de la huella, la cual no tiene información relevante para la validación. Cuando se aplican procesos de extracción de minucias a una toma sin segmentar, se obtiene como resultado la detección de minucias falsas por el ruido presente en esta región, por lo que es importante hacer esta separación para los pasos subsecuentes.

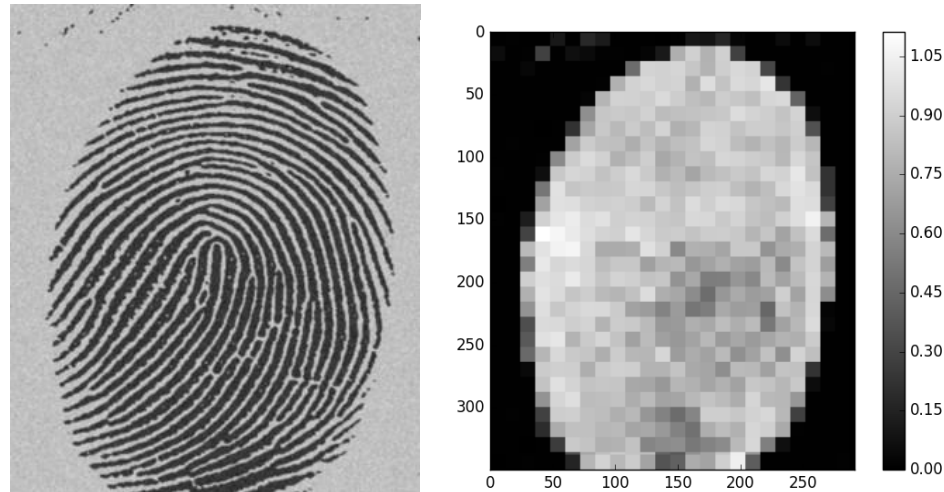
En la imagen de una huella, la región del fondo generalmente presenta una varianza en escala de grises muy baja, mientras que el primer plano, tiene una varianza más alta. Por ende, se utiliza un método basado en un umbral de varianza para realizar la segmentación (código anexo C). A continuación, se presenta un diagrama que explica el proceso de segmentación aplicado.

Figura 7. Diagrama para la segmentación de una imagen.



Como podemos ver en la siguiente imagen, las zonas en las que la varianza es mínima o igual a cero, tomara que es parte del fondo de la imagen y que no contiene información de la huella.

Figura 8. Segmentación aplicada en la imagen.



3.1.3. ESTIMACIÓN DE LA ORIENTACIÓN Y LA FRECUENCIA

3.1.3.1. Estimación de la orientación

La estimación de orientación es un paso fundamental en el proceso de mejoramiento de la imagen, ya que en los pasos posteriores del filtro Gabor, es fundamental tener la orientación local del pixel para obtener mejores resultados en el filtrado. Para este proceso, se utiliza el método desarrollado por Kass [12], para el análisis de patrones orientados. En este se utiliza los gradientes de cada

pixel para hallar la estimación media cuadrática y determinar de esta manera los ángulos [11].

Los pasos para calcular la orientación utilizando este método son los siguientes:

1. Primero se halla el gradiente en x y el gradiente en y de cada pixel utilizando el operador Sobel. Este gradiente representa la magnitud de cambio en el gradiente en ambas direcciones. El operador Sobel horizontal es utilizado para computar el $\partial x(i, j)$:

$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \quad (5)$$

Y el operador Sobel vertical para determinar $\partial y(i, j)$:

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \quad (6)$$

2. Luego de obtener estos gradientes, se aplica la fórmula de arco tangente para encontrar la orientación de los pixeles en la imagen.

$$\theta(i, j) = \tan^{-1} \frac{\partial y(i, j)}{\partial x(i, j)} \quad (7)$$

Donde $\theta(i, j)$ es la estimacion que necesitamos.

3. Posteriormente le hacemos un suavizado a la estimación de orientación utilizando un filtro Gaussiano (Ecuación 8 y 9):

$$\Phi_x(i,j) = \cos(2\theta(i,j)) \quad (8)$$

$$\Phi_y(i,j) = \sin(2\theta(i,j)) \quad (9)$$

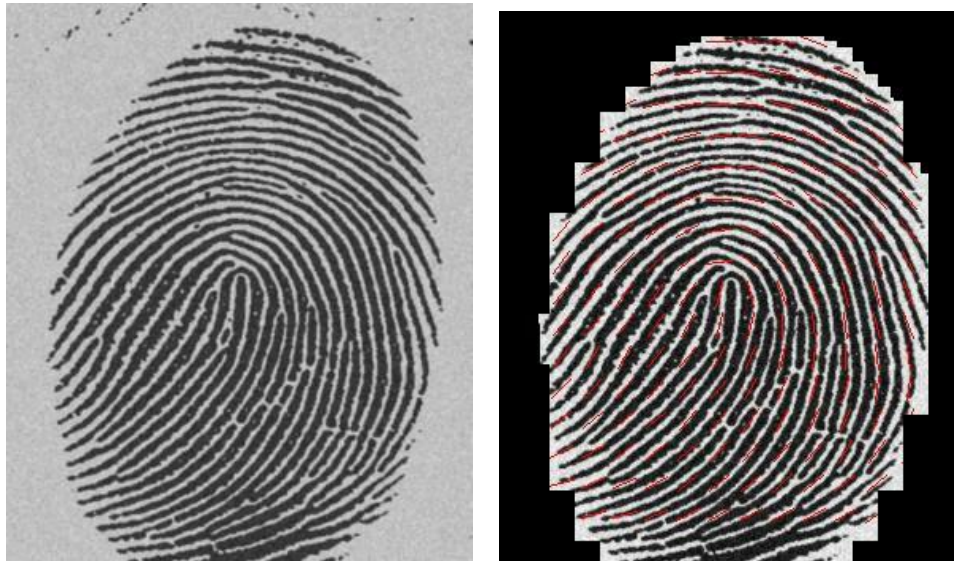
4. Teniendo los valores de Φ_x y Φ_y , se les aplica un filtro gaussiano de orden cero y una desviación estándar de 5, obteniendo la siguiente mascara.

$$\begin{bmatrix} 0.109634 & 0.111842 & 0.109634 \\ 0.111842 & 0.114094 & 0.111842 \\ 0.109634 & 0.111842 & 0.109634 \end{bmatrix} \quad (10)$$

5. Por último, con la salida de los procesos implementados anteriormente, se encuentra la orientación final ya suavizada.

$$O(i,j) = \frac{1}{2} \tan^{-1} \frac{\Phi'_x(i,j)}{\Phi'_y(i,j)} \quad (11)$$

Figura 9. Estimación de la orientación aplicada en la imagen.



En la figura 9 se ve la prueba realizada para verificar el correcto funcionamiento de la estimación de orientación. En esta se hizo una línea recta con la orientación hallada para ver que tanto concordaba con la imagen.

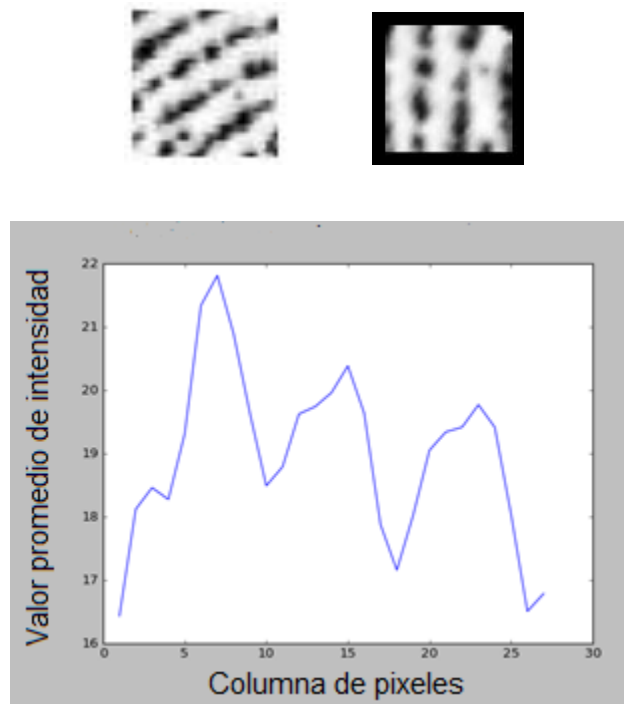
El resultado entonces de esta etapa, es una matriz que contiene la orientación de todos los pixeles de la imagen con su respectiva orientación.

3.1.3.2. Estimación de la frecuencia

Además de la orientación de los pixeles, otro parámetro importante en la implementación del filtro Gabor, es la frecuencia local de las crestas. En este caso se utilizó el método planteado por Raymond Thai [11], el cual incluye el uso de un filtro pasa bajos que mejora los resultados.

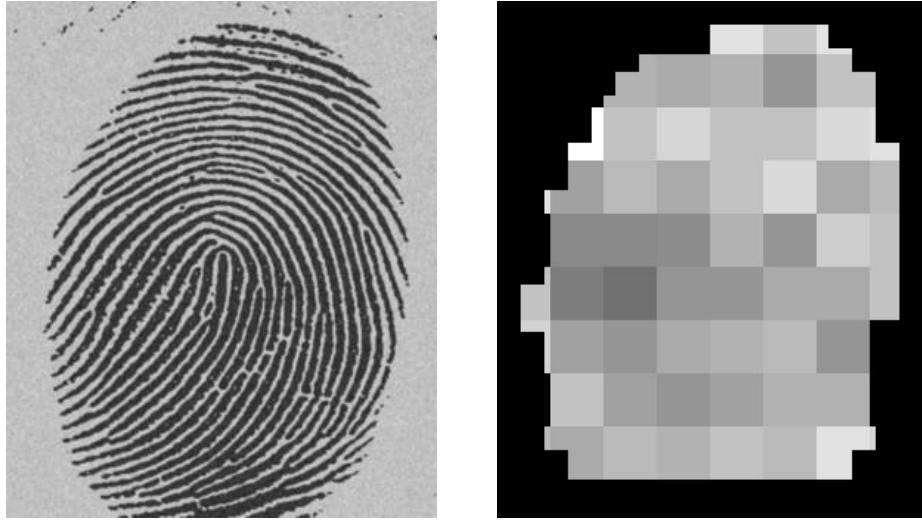
El proceso de la estimación de frecuencia, al igual que en la segmentación, se recorre la imagen utilizando bloques cuadrados de 9x9 pixeles. Sin embargo, en este caso, no se halla la varianza de los bloques. Primero se encuentra la media de orientaciones (con las orientaciones halladas en la estimación de orientación) en los pixeles de ese bloque, para con este girar la huella y que las crestas queden en posición vertical. Posteriormente se hace una proyección de las crestas usando el valor de intensidad promedio de cada columna de pixeles, como se ve en la figura 10.

Figura 10. Estimación de la frecuencia de una imagen.



Luego de esto, se hace un conteo de los máximos en ese bloque y se encuentra el periodo de la onda utilizando estos máximos. Se toma el primer y último máximo de la imagen y se cuenta la diferencia en píxeles que hay entre estos, siendo este el periodo, y como sabemos que el periodo es la inversa de la frecuencia, ya tenemos el valor estimado de la frecuencia para el bloque que se esté tratando. Este valor de frecuencia espacial, se les asigna entonces a todos los píxeles dentro de ese bloque. En la figura 11 se observa la imagen de los bloques de la huella y la diferencia de valores de frecuencia en la misma.

Figura 11. Estimación de la frecuencia aplicada en la imagen.



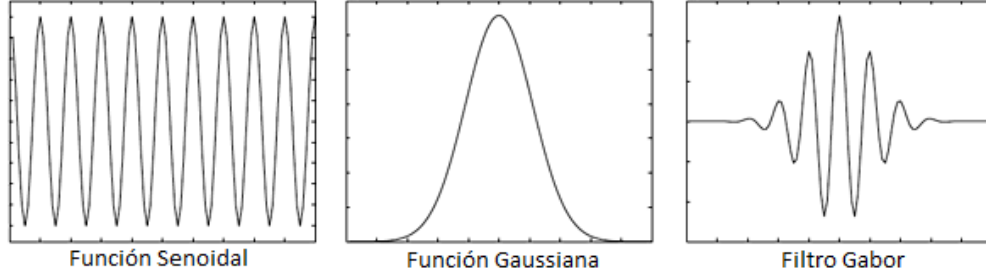
3.1.4. FILTRO GABOR

El filtro Gabor consiste en el producto entre una función Gaussiana y una senoidal [13]. Una función de Gabor elemental en una dimensión, se puede definir como el producto de un pulso con la forma de una función de probabilidad y un armónico oscilatorio de cualquier frecuencia (Ecuación 12).

$$g(t) = e^{-\alpha^2(t-t_0)^2} e^{i2\pi f_0 t + \varphi} \quad (12)$$

Donde α es la duración del ancho de banda, t_0 denota el centroide, f_0 es la frecuencia de la senoidal y φ el cambio de fase. En la figura 12 podemos ver la representación gráfica del filtro Gabor en una dimensión. Aquí podemos ver el resultado de la función senoidal con la función Gaussiana.

Figura 12. Filtro Gabor en una dimensión.



Fuente: Gabor Filter Visualization [13].

De igual manera, la ecuación se puede generalizar para dos dimensiones. En 2D, la variable del tiempo se reemplaza por coordenadas espaciales (x, y) en el dominio espacial (Ecuación 13).

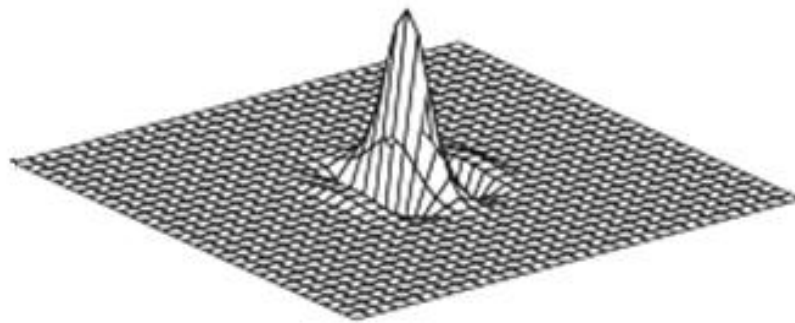
$$G(x, y, \theta, f) = \exp\left\{-\frac{1}{2}\left[\frac{x_{\theta}^2}{\sigma_x^2} + \frac{y_{\theta}^2}{\sigma_y^2}\right]\right\} \cos(2\pi f x_{\theta}) \quad (13)$$

$$x_{\theta} = x \cos\theta + y \sin\theta \quad (14)$$

$$y_{\theta} = -x \sin\theta + y \cos\theta \quad (15)$$

Donde θ es el ángulo de orientación del filtro gabor, f la frecuencia y donde σ_x y σ_y son las desviaciones del gaussiano. En la figura 13 encontramos lo que sería un filtro Gabor en 2D, donde la señal seno tiene una orientación que será la que tendrá el filtro, permitiéndole actuar específicamente en esos ángulos.

Figura 13. Filtro Gabor en dos dimensiones.



Fuente: Gabor filter visualization [13].

Para implementar el filtro Gabor, en primer lugar, se encuentran los valores donde la imagen tenga valores de frecuencia válidos, es decir donde la frecuencia sea diferente de cero. Con estas frecuencias y las orientaciones de cada pixel, se generan los filtros para cada zona valida de la huella, es decir cada zona que contenga información relevante, se le aplicara un filtro diferente según su orientación y frecuencia estimadas. Esta orientación será la orientación que tenga la señal seno, al igual que la frecuencia.

La implementación del filtro consiste en la convolucion espacial de la imagen con la mascara de coeficientes que son parte del filtro, esto permitirá aumentar el contraste entre las crestas, lo que facilitara las siguientes etapas de binarización y adelgazamiento de las crestas.

Por último, podemos observar en la figura 14 un ejemplo de los resultados obtenidos luego de aplicar un filtro Gabor. Aquí las crestas se ven más claras permitiendo encontrar de mejor manera las minucias.

Figura 14. Filtro gabor aplicado a la imagen.



3.1.5. BINARIZACIÓN

La binarización de una imagen consiste en comparar los niveles de gris presentes en la imagen con un valor (umbral) predeterminado [14] . Si el nivel de gris de la imagen es menor que el umbral predeterminado, al pixel de la imagen binarizada se le asigna el valor 0 (blanco), y si es mayor se le asigna un 1 (negro). Este valor de umbral se determinó de manera experimental como cero, ya que al aumentarlo o disminuirlo, agregaba o eliminaba información de la huella respectivamente. De esta forma se obtiene una imagen en blanco y negro. En la figura 15 se muestra un ejemplo de imagen binarizada.

Figura 15. Binarización aplicada a la imagen.



3.1.6. PRUEBAS

En esta etapa, se decidió realizar unas pruebas para determinar que parámetros debían tener las etapas de normalización y segmentación para obtener mejores resultados en la aplicación del filtro. En la etapa de la normalización, se modificaron las variables de media y varianza requerida, que, según la literatura, varían dependiendo del sensor implementado. Los demás parámetros del proceso se mantuvieron fijos.

Para la prueba, se utilizaron tres valores diferentes, el valor ideal de la normalización y los otros dos por encima y por debajo del valor ideal. Se analizaron los tiempos de procesamiento y el cómo se ven afectados estos cambios en el resultado del proceso. En la tabla 4 podemos encontrar los resultados para la etapa de la normalización.

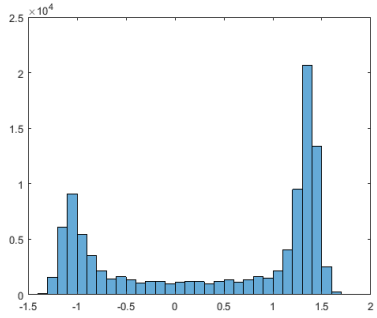
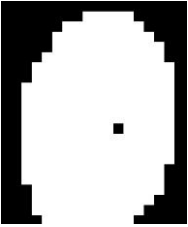

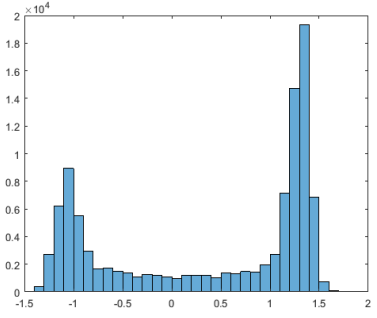


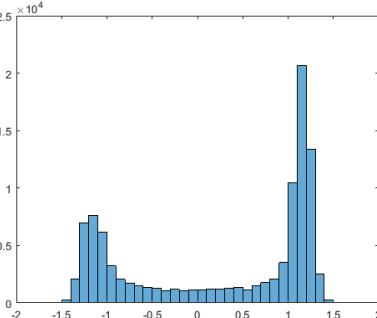
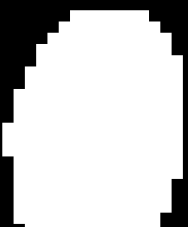





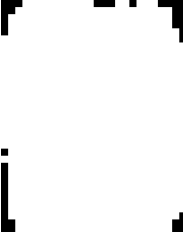

	Histograma	Segmentada	Filtrada
Varianza requerida= 0.5 Media requerida= - 0.5 Tiempo de procesamiento= 2.04 segundos			
Varianza requerida= 1 Media requerida= 0 Tiempo de procesamiento = 1.99 segundos			
Varianza requerida= 2 Media requerida= 5 Tiempo de procesamiento = 2.39 segundos			

Tabla 4. Resultados de las pruebas con la normalización aplicada a la imagen.

Como podemos ver, aunque no se puedan ver cambios a simple vista, el proceso si se ve afectado por los cambios en la normalización, afectando los resultados de la segmentación. Lo que podría borrar información de la huella.

En la tabla 5, encontramos la prueba con la segmentación. En esta se varió el tamaño del bloque que recorre la imagen al momento de hallar la varianza, arrojando los siguientes resultados:

	Normalizada	Segmentada	Filtrada
Blocksize= 4 pixeles Tiempo de procesamiento = 1.71 segundos			
Blocksize= 9 pixeles Tiempo de procesamiento = 1.99 segundos			




<p>Blocksize= 18 pixeles</p> <p>Tiempo de procesamiento = 2.09 segundos</p>			
---	---	--	---

Tabla 5. Resultados de las pruebas con la segmentación aplicada a la imagen.

Como podemos observar, al ser muy pequeño el bloque, tomara parte de la huella como si fuese parte del fondo, sin información, mientras que, si es muy grande, no delimitara bien la zona de trabajo.

3.2. ADELGAZAMIENTO Y DETECTOR DE MINUCIAS

Despues de aplicar el filtro, podemos entonces continuar con el proceso de adelgazamiento y detección de las minucias. En esta etapa se reducirán las crestas de la huella hasta que tengan un grosor de un pixel, para con esto poder hallar más fácilmente las minucias utilizando el método de Crossing Number.

3.2.1. ADELGAZAMIENTO

En este caso para el adelgazamiento de la huella se hace un proceso de esqueletización de las crestas [15]. Para esto se utilizó la función de thinning presente en Python, la cual utiliza el método de thinning desarrollado por Guo-Hall [16]. Este método es un algoritmo de dos iteraciones usando operadores con un soporte paralelo, es decir un soporte 3x3.

Figura 16. Ventana de 3x3 pixeles para el adelgazamiento.

P1	P2	P3
P8	P_i	P4
P7	P6	P5

Como vemos en la figura 16, nos referimos a P2, P4, P6 y P8 como vecinos laterales, y P1, P3, P5 y P7 como vecinos diagonales. También se definen algunas variables como $C(p)$, la cual se define como el número de distintos pixeles 8-conectados en los vecinos de P_i . Igualmente se define $B(p)$ como el número de pixeles con valor de 1 en los vecinos de P_i . Se utilizan los símbolos \neg , $\&\&$ y \parallel como operadores lógicos, de negación, AND y OR, respectivamente. Por último, se tiene la variable $N(p)$, la cual es útil para obtener un mejor adelgazamiento. Con lo anterior tenemos entonces que:

$$N_1(p) = (p_1 \parallel p_2) + (p_3 \parallel p_4) + (p_5 \parallel p_6) + (p_7 \parallel p_8) \quad (16)$$

$$N_2(p) = (p_1 \&\& p_2) + (p_3 \&\& p_4) + (p_5 \&\& p_6) + (p_7 \&\& p_8) \quad (17)$$

$$N(p) = \min(N_1(p), N_2(p)) \quad (18)$$

Con lo anterior se tiene entonces que un pixel con valor de uno se eliminara (se cambiara su valor por cero) si cumple las 3 condiciones siguientes.

- a. $C(p)=1$;
- b. $2 \leq N(p) \leq 3$;
- c. Aplica una de las siguiente:
 1. $(p_2 \parallel p_3 \parallel \overline{p_4}) = 0$ en una iteración impar
 2. $(p_6 \parallel p_7 \parallel \overline{p_1}) = 0$ en una iteración par.

El proceso se realiza entonces hasta que no se eliminan más pixeles después de una iteración.

En la figura 17 encontramos los resultados finales del proceso de adelgazamiento de la huella, como podemos ver

Figura 17. Resultado del adelgazamiento.



3.2.2. DETECTOR DE MINUCIAS

El método más utilizado para la extracción de minucias hace uso del concepto de Crossing Number (CN) [2]. Este método implica la utilización de la imagen esqueletizada, donde las crestas tienen una conectividad 8, es decir que están conectados por un solo pixel. Para extraer las minucias se escanean los vecinos locales de cada pixel, utilizando una ventana de 3x3 pixeles.

Figura 18. Ventana de 3x3 pixeles para la detección de minucias.

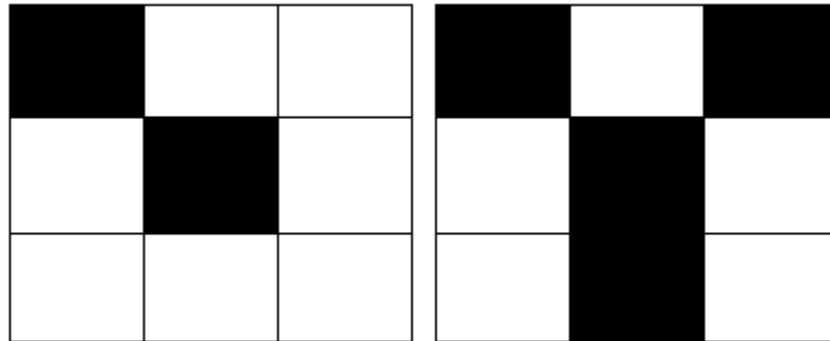
P2	P3	P4
P1	P_i	P5
P8	P7	P6

El valor de CN se computa usando los valores de cada pixel y utilizando la siguiente formula que en otras palabras es la mitad de la suma de las diferencias entre pares de pixeles adyacentes en la ventana:

$$CN = 0.5 \sum_{i=1}^8 |P_i - P_{i+1}| \quad (19)$$

A partir de este valor se puede determinar si la minucia es una terminación o una bifurcación. Si el valor de CN es igual a 1 quiere decir que la minucia es una terminación y si es igual a 3 quiere decir que esta es una bifurcación. Son estas minucias las que posteriormente en la comparación se rotaran y se hallaran los matches según su distancia euclídea y su orientación. En la figura 19 podemos ver a la izquierda lo que seria una terminación, y a la derecha una bifurcación.

Figura 19. Detección de minucias en una imagen.



3.3. COMPARACIÓN

El método de comparación de huellas basado en minucias, es uno de los métodos más populares, utilizados en aplicaciones comerciales, ya que tiene un buen desempeño y un tiempo de computación bajo. Este método lo que busca es alinear las minucias de la imagen de entrada con las minucias de la huella base,

y encontrar el número de minucias coincidentes. Luego de alinearlas, tanto en orientación como en posición, dos minucias se consideran coincidentes si su distancia y orientación, es menor al valor de tolerancia asignado.

Posición en X	Posición en Y	Tipo de Minucia	Orientación en radianes
330	223	1(Terminación)	-0.976
340	126	1(Terminación)	0.9
410	233	1(Terminación)	-0.832
460	165	1(Terminación)	1.37
460	217	1(Terminación)	-1.03
510	340	3(Bifurcación)	-0.624
530	200	3(Bifurcación)	-1.31
540	380	1(Terminación)	0.906
540	229	1(Terminación)	-0.811
550	258	1(Terminación)	-0.605

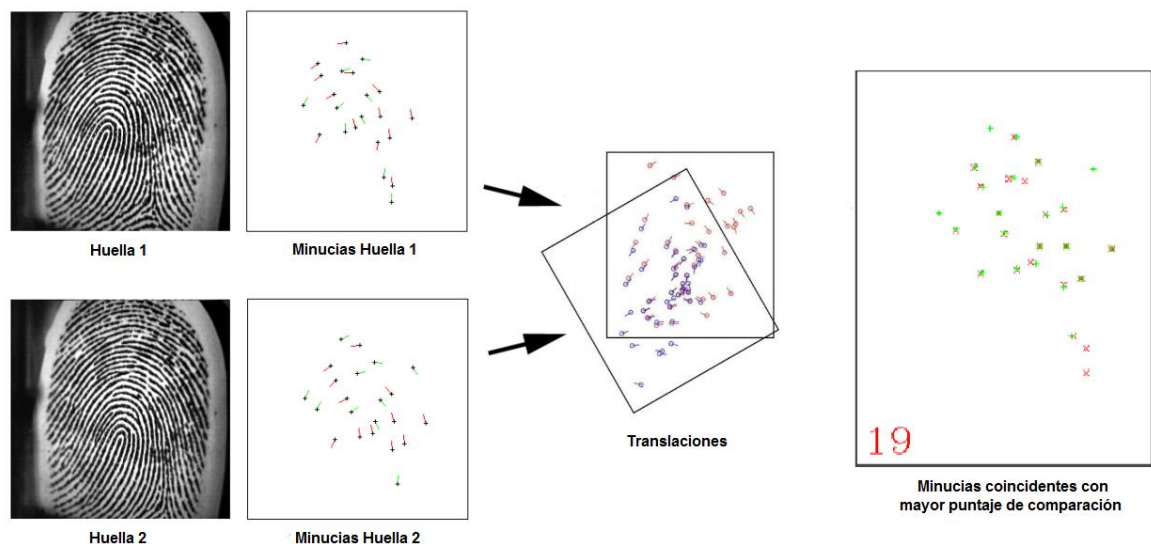
Tabla 6. Información de la minucia.

En la tabla anterior, encontramos la manera en que se guardan las minucias, para así tener un mejor entendimiento del proceso.

Para cada minucia, se guarda su posición en x,y, el tipo de minucia y la orientación en radianes, esto será lo que en ultimas se va a comprar entre las huellas. Teniendo esta tabla y siendo cada fila una minucia, se realiza el proceso de comparacion.

Este consiste en realizar una translación en rotación y posición a la huella de entrada, tomando como eje diferentes minucias, para así encontrar la comparación con un mayor puntaje de comparación, entre todas las relaciones posibles. Estas minucias llegan a formar segmentos o grafos, con su orientación y posición, y son estos los que posteriormente entraran a compararse dándole un margen de error de 5 pixeles y 5 grados. De esta manera, con los segmentos coincidentes, podemos obtener el puntaje de comparación, que será en ultimas, el que determinara si las huellas son o no coincidentes.

Figura 20. Método de comparación basado en minucias.



La buena alineación de la huella es importante para este método, por lo que es importante tener un buen procesamiento para las transformadas de traslación, rotación y escalamiento de las imágenes.

Según la cantidad de minucias se dará un puntaje de similitud, y se repetirá el proceso con las demás minucias, al final se toma la comparación con mayor puntaje y con esta se determinará si es o no la persona. Este puntaje está dado por la ecuación:

$$\sqrt{\frac{\beta^2}{\alpha_I * \alpha_f}} \quad (20)$$

Donde β es el índice de similitudes encontradas y donde α_I y α_f representan el número de minucias de cada una de las huellas (inicial y final). Para este caso, se determinó que un puntaje de comparación mayor a 0.8, determina que la huella base y la de entrada, coinciden siendo el mismo individuo.

4. IMPLEMENTACIÓN DEL SISTEMA BIOMÉTRICO

Para la implementación del proceso anterior, se utilizó Python con la librería de opencv. A continuación, se presenta una tabla con los respectivos anexos según su etapa en el procesamiento de la huella.

Pre-procesamiento	En estos códigos podemos encontrar la normalización, segmentación, estimación de frecuencia y orientación, y el filtro Gabor.	Anexo B, Anexo C, Anexo D, Anexo E y Anexo F.
Detección de Minucias	En esta sección podemos encontrar los códigos para hacer el proceso de binarización, adelgazamiento y detección de minucias.	Anexo B y Anexo G
Comparación	Este proceso, contiene los códigos para realizar la	Anexo A, Anexo B,

	comparación, incluyendo los encargados de realizar las rotaciones en la huella.	Anexo H, Anexo I, Anexo J y Anexo K, Anexo L
--	---	--

Tabla 7. Algoritmos realizados para el desarrollo del sistema biométrico.

4.1. INTERFAZ GRAFICA

Haciendo uso de Qt y PyQt, se desarrolló una interfaz gráfica que le permite al usuario interactuar de manera dinámica con el sistema, esta interfaz, cuenta con seis vistas, una vista Inicio, que le permite al usuario hacer el ingreso como usuario o como administrador.

Figura 21. Vista inicial de la GUI.



La vista de administrador, que permite agregar o eliminar un usuario y crear un nuevo administrador.

Figura 22. Vista Administrador de la GUI.



La vista de nuevo registro, donde se crea un nuevo usuario, tiene las opciones de capturar huella y guardarla como se ve en la figura 23.

Figura 23. Vista Nuevo registro de la GUI.



La vista de eliminar registro presente en la figura 24, permite ingresar el ID del usuario que se va a eliminar posteriormente.

Figura 24. Eliminar registro de la GUI.



La vista nuevo administrador, permite crear un nuevo administrador, posee las opciones de capturar huella y guardar el registro de la misma.

Figura 25. Vista Nuevo administrador de la GUI.



La vista del "ingreso" con las opciones de capturar huella y compararla.

Figura 26. Vista ingreso de la GUI.



















Todas las vistas poseen la opción de cancelar, a excepción de la vista inicial, que permite salir de la interfaz.

4.2. VALIDACIÓN DEL SISTEMA

En esta etapa del desarrollo, se hicieron las pruebas para evaluar el funcionamiento del sistema y su respuesta a ciertas variaciones a la hora de tomar una muestra.

Se tomó una huella base, se realizaron las pruebas de comparación con la misma huella girada levemente a la izquierda, a la derecha y cuando la huella no correspondía a la original.

Es importante tener en cuenta que esta prueba puede dar cuatro resultados diferentes. True Positive (TP), se da cuando una huella es correctamente identificada, True Negative (TN), cuando una huella no es reconocida correctamente por el sistema, False Positive (FP), cuando el sistema da un resultado positivo a dos huellas diferentes y False Negative (FN), cuando el procesamiento da un resultado negativo a pesar de ser la misma huella. Esta prueba se realizó con cinco huellas diferentes y se obtuvieron las imágenes, que se muestran a continuación:

Huella Original	Girada a la derecha	Girada a la izquierda	Huella diferente
	 Match: Si. TP	 Match: Si. TP	 Match: No. TN
	 Match: Si. TP	 Match: Si. TP	 Match: No. TN
	 Match: Si. TP	 Match: Si. TP	 Match: Si. FP
	 Match: Si. TP	 Match: Si. TP	 Match: No. TN





			
	Match: No. FN	Match: Si. TP	Match: No. TN

Tabla 8. Pruebas y validación del funcionamiento del sistema biométrico.

Con las pruebas anteriores se determinó un porcentaje de 80% como umbral que definía si la huella correspondía o no a la huella base. En caso de que el puntaje no superara este porcentaje, se tomó que la huella no correspondía. Este umbral hallado en estas pruebas, se utilizó para la posterior validación con 25 huellas. En esta etapa del desarrollo también se determinó que, para crear un nuevo usuario, se harían tres tomas, de tal manera que en el proceso de comparación se validara que la huella que estaba intentando ingresar corresponda con al menos dos de las 3 huellas en la base de datos para determinado usuario, lo que aumenta el porcentaje de efectividad final.

Para la siguiente prueba al sistema, se decidió hacer el mismo realizado en la prueba anterior, pero haciendo uso de una mayor cantidad de huellas. En este caso se utilizaron 25 huellas diferentes, como se puede ver en la tabla 9, las cuales se cotejaron entre ellas, siendo un total de 625 comparaciones.

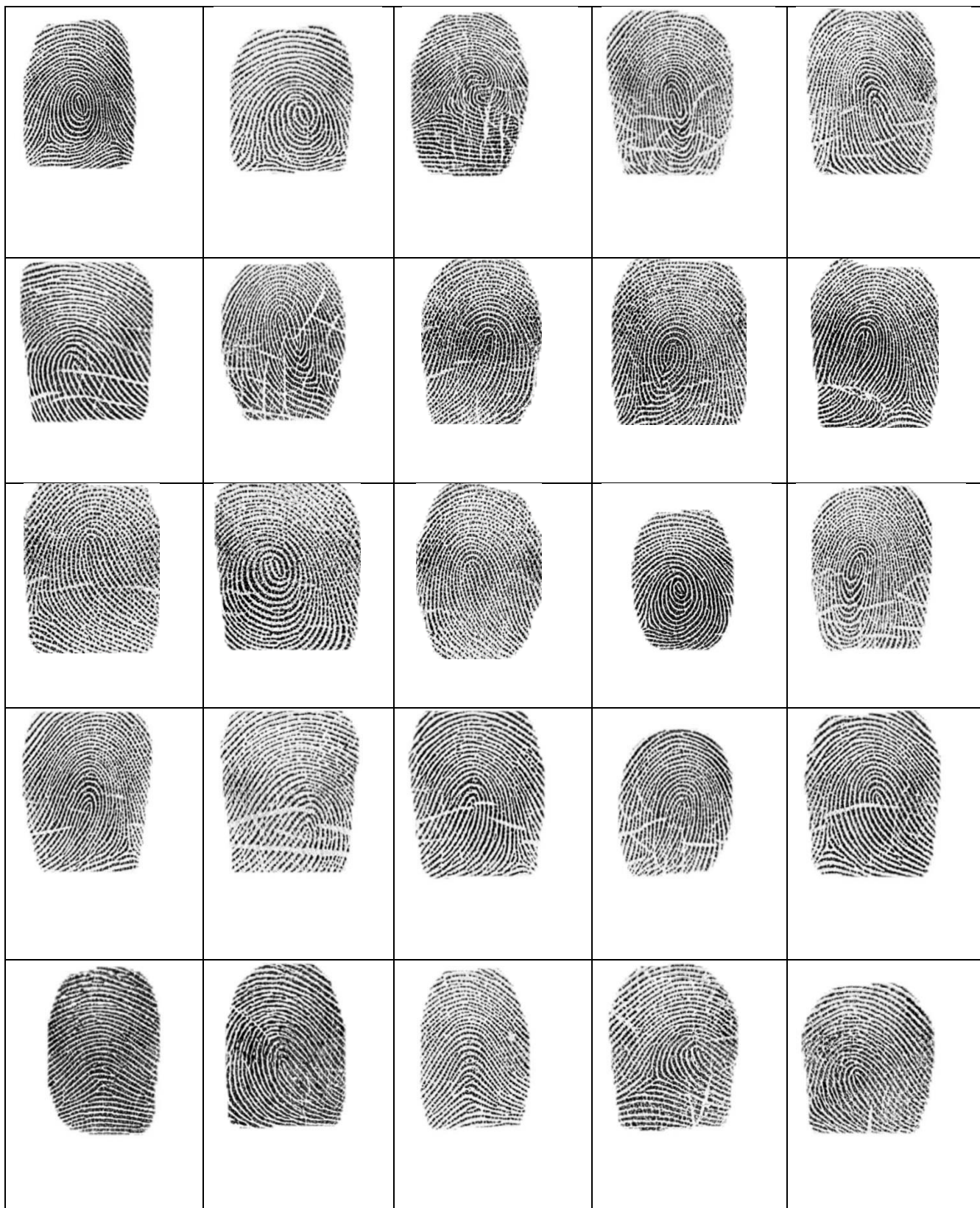


Tabla 9. Base de huellas utilizadas para la validación del sistema.

Ante esta prueba se obtuvo los siguientes resultados:

True Positive (TP)	21
False Negative (FN)	4
True Negative (TN)	502
False Positive (FP)	98

Tabla 10. Resultados de las pruebas de validación.

Siguiendo los datos obtenidos, se siguieron los estándares propuestos por la National Institute of Standards and Technology, quienes, usando una matriz de coincidencias como en este caso, hallan los valores de True Acceptance Rate y el False Acceptance Rate, que son las medidas utilizadas comúnmente para la evaluación de sistemas biométricos.

Entonces, tomando que los True Positive (TP) fueron de 21 y que se encontraron 4 False Negative (FN), tenemos que la sensibilidad o True Positive Rate es de:

$$TPR = \frac{TP}{TP+FN} = 0.84 \quad (21)$$

Lo que nos daría que el sistema tiene una sensibilidad del 84%. De igual manera la especificidad o True Negative Rate (TNR) se determina usando los True Negative (TN), que en este caso fueron 502, y los False Positive (FP), que en esta prueba fueron 98, tenemos:

$$TNR = \frac{TN}{FP+TN} = 0.84 \quad (22)$$

También tenemos que hallar la precisión del sistema que se hace con los TP y los FP. Esto nos da como resultado:

$$PPV = \frac{TP}{TP+FP} = 0.84 \quad (23)$$

Teniendo entonces estos parámetros procedemos a hallar el valor del False Acceptance Rate (FAR) y el False Negative Rate (FNR), los cuales son los índices estándar para medir el sistema.

$$FAR = \frac{FP}{FP+TN} = 1 - TNR = 0.16 \quad (24)$$

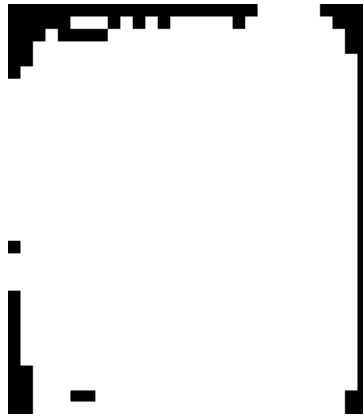
$$FNR = \frac{FN}{TP+FN} = 1 - TPR = 0.16 \quad (25)$$

Por último, se tomó uno de los registros que dio como resultado falso positivo, para tratar de determinar en qué parte del proceso se podría mejorar el algoritmo de procesamiento de la imagen. En la tabla 11 encontramos tomas que permitieron hacer un análisis mas detallado del proceso y determinar que etapa se podría optimizar.

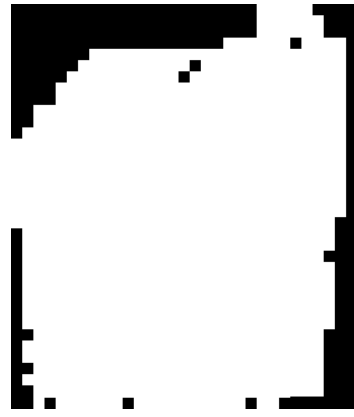
HUELLA BASE	FALSO POSITIVO
Normalización	Normalización



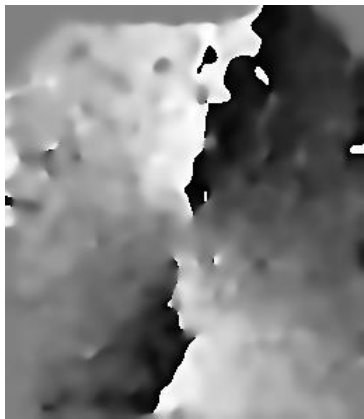
Segmentación



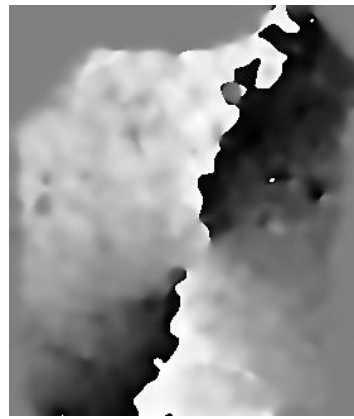
Segmentación



Estimación de la orientación

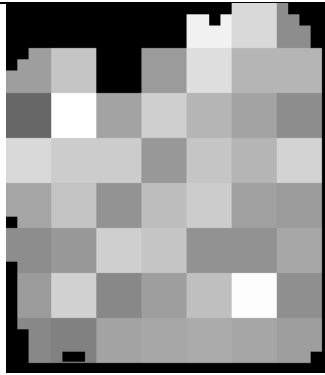


Estimación de la orientación



Estimación de la Frecuencia

Estimación de la Frecuencia



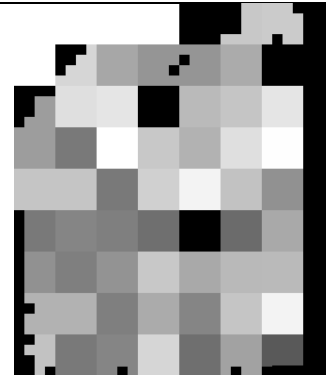
Filtro Gabor



Thinning



Detección de Minucias



Filtro Gabor



Thinning



Detección de minucias

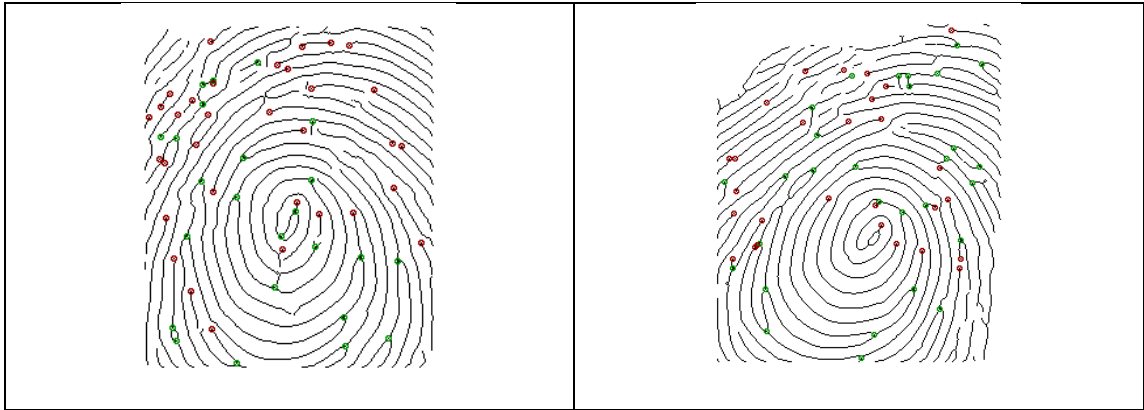


Tabla 9. Comparación del procesamiento de la imagen para un falso negativo.

Como podemos ver de las imágenes, el proceso es directo y varia poco dentro de este. Sin embargo, si podemos ver que la manera en que se ingresa la huella es la que verdaderamente afecta el proceso de la identificación. La orientación es un factor a considerar pero que no afecta tanto el proceso. No obstante, la presión aplicada en el sensor al momento de la toma, si tiene consecuencias en el resto del proceso. Como podemos ver de la huella inclinada, al habérsela mayor presión al momento de aplicarla, las líneas blancas se reducen y el proceso alcanza a agregar minucias o pasar de considerarlas terminaciones a considerarlas bifurcaciones. También es importante resaltar que la suciedad en el sensor como resultado de varias pruebas a diferente presión, puede terminar agregando información no deseada a la muestra.

A continuación, en la tabla 12, podemos ver en una tabla una serie de imágenes que podrían llegar a dar un resultado falso. Como vemos hay diferentes factores

que afectan la toma de la huella. La presión por un lado hará que las líneas de la huella queden más marcadas, mientras que la posición del dedo sobre el sensor, no solo afectara la inclinación, si no que puede hacer que el sensor no tome partes de la huella. De igual forma es importante la limpieza del sensor luego de una serie de tomas, ya que la presión y la suciedad presente en los dedos, puede dejar marcas, lo que puede hacer que parte de la huella de alguien, quede en la toma de la huella de otra persona, afectando así el proceso.



Tabla 12. Resultados falsos de la huella.

Por otro lado, en la tabla 13, se presenta lo que se considera como una buena toma por parte del sensor, lo que ayudaría que la efectividad del proceso aumentara.



Tabla 13. Resultados positivos de la huella.

5. CONCLUSIONES

- El procesador del sistema embebido afecta enormemente los tiempos de procesamiento, llegando a reducir mas de la mitad los tiempos de procesamiento de la raspberry Pi B+ a la raspberry 3.
- El tiempo de procesamiento se verá afectado también si los parámetros como los de normalización y segmentación, no son adecuados, ya que el algoritmo tendrá que hacer más ciclos para realizar el trabajo.
- Se recomienda probar otros métodos de comparación de huellas basados en minucias para tratar de optimizar todo el proceso en general y obtener mejores resultados.

BIBLIOGRAFÍA

- [1] G. Gabriela, «Extracción de características y comparación de una huella digital,» Escuela Superior politécnica del litoral, Guayaquil, Ecuador, 2009.
- [2] J. A. A. Molano, Verificación de correspondencia en huellas dactilares aplicando técnicas de procesamiento y análisis digital de imágenes para la disminución del tiempo de cotejo, Popayán, Cauca: Universidad del Cauca, facultad de Ciencias Naturales, Exactas y de la Educación. , 2010.
- [3] D. Persona, «2fa.com,» [En línea]. Available: http://2fa.com/wp-content/uploads/Datasheets/UareU_4500Reader20100416.pdf. [Último acceso: 14 10 2016].
- [4] B. U. Miñarro, «Sistemas Embebidos,» 2006. [En línea]. Available: <http://ocw.um.es/ingenierias/sistemas-embebidos/material-de-clase-1/ssee-t01.pdf>. [Último acceso: 6 Septiembre 2016].
- [5] T. Klosowski, «El rendimiento de la Raspberry Pi 3, comparado contra los modelos anteriores,» 3 marzo 2016. [En línea]. Available: <http://es.gizmodo.com/el-rendimiento-de-la-raspberry-pi-3-comparado-contra-l-1762657409>. [Último acceso: 14 Octubre 2015].

- [6] A. Kopytov, «Sysbench,» Gizmodo, 2016. [En línea]. Available: <https://github.com/akopytov/sysbench>. [Último acceso: 29 11 2016].
- [7] Python, «What is Python? Executive Summary,» Python, 2016. [En línea]. Available: <https://www.python.org/doc/essays/blurb/>. [Último acceso: 29 Octubre 2016].
- [8] OpenCv, «Open Source Computer Vision,» 2016. [En línea]. Available: <http://opencv.org/>. [Último acceso: 26 Julio 2016].
- [9] SQLite, «About SQLite,» [En línea]. Available: <https://sqlite.org/fileformat2.html>. [Último acceso: 21 Julio 2017].
- [10] K. M. M, «Comparison of fingerprint enhancement techniques through Mean Square Error and Peak-Signal to Noise Ratio,» *International Journal on Computer Science and Engineering* , vol. 3, nº 1, 2011.
- [11] R. Thai, «Fingerprint Image Enhancement and Minutiae Extraction,» The university of Western Australia , Stirling Hwy, Australia, 2003.
- [12] M. Kaas, «Analyzing Oriented Patterns,» Schlumberger Palo Alto Research , Palo Alto, 2008.
- [13] S. N. Prasad, «Gabor filter visualization,» Maryland, 2010.

- [14] J. J. Esqueda, Fundamentos de procesamiento de imagenes, Baja California: Universidad Autonoma de Baja California, 2005.
- [15] NTNU, «Hit-or-miss transform,» [En línea]. Available: <http://www.cs.tau.ac.il/~turkel/notes/hit-or-miss.pdf>. [Último acceso: 8 Febrero 2016].
- [16] Z. G.-R. Hall, «Parallel Thinning with two-subiterations algorithms,» *Image Proccessing and computer vision*, vol. 32, p. 359, 1989.
- [17] Griaule, «Minutae Based Matching,» Griaule Biometrics , 2014. [En línea]. Available: <http://www.griaulebiometrics.com/en-us/book/understanding-biometrics/types/matching/minutiae> . [Último acceso: 04 Octubre 2015].

ANEXOS

Anexo A. Código de Comparación.

Nombre Archivo: Comparacion.py

```
# -*- encoding: utf-8 -*-  
  
import FP  
  
import Match  
  
from PIL import Image  
  
import numpy as np  
  
  
im1=Image.open('fp1.jpg','r')#Leo la primera imagen  
im1=im1.convert('L')#La convierto a escala de grises  
  
im1=np.asarray(im1.getdata(),dtype=np.uint8).reshape((im1.size[1],im1.size[0]))#convierto el objeto imagen  
en una matrix de pixeles  
  
im2=Image.open('fp1.jpg','r')#Leo la primera imagen  
im2=im2.convert('L')#La convierto a escala de grises  
  
im2=np.asarray(im2.getdata(),dtype=np.uint8).reshape((im2.size[1],im2.size[0]))#convierte el objeto imagen  
en una matrix de pixeles  
  
Matrix1=FP.extraerMinucias(im1)#Extraigo la matriz de minucias de la primera imagen  
  
#np.savetxt('Matrix1.dat', Matrix1, fmt='%%.4e') #Guardo la matriz de minucias en un archivo .dat  
  
#Matrix1 = np.loadtxt('Matrix1.dat') #Leo la matriz de minucias desde un archivo .dat  
  
Matrix2=FP.extraerMinucias(im2)#Extraigo la matriz de minucias dela segunda imagen  
  
S=Match.matching(Matrix1, Matrix2)#Calculo la similitud de las matrices de minucias  
  
print S#Imprimo el valor de la similitud de las matrices de minucias
```

Anexo B. Código Base.

Nombre archivo: FP.py

Este código contiene el llamado a la mayoría de archivos y clases del proceso. La clase `extraerminucias` contiene el proceso completo desde el pre procesamiento hasta el llamado a las clases para la comparación.

```
# -*- encoding: utf-8 -*-  
  
#Importo las librerías necesarias para el proceso  
import numpy as np#Librería para trabajar con matrices y vectores  
import matplotlib.image as mpimg#Librería para guardar imágenes  
import matplotlib.pyplot as plt#Librería para mostrar imágenes  
from PIL import Image#Librería para leer imágenes y hacer conversiones  
import thinning#Librería para hacer thinning a imágenes  
import time#Librería para calcular el tiempo de ejecución del código  
import cv2#Librería para el tratamiento de imágenes  
  
#importo las demás clases creadas para el proceso  
import RidgeFreq #Cálculo de la frecuencia de la imagen  
import Seg #Segmentación de la imagen  
import RidgeFilter #Filtro Gabor de la imagen  
import Minucias #Busqueda de minucias  
import OrS#Cálculo de la orientación  
import CorreccionM#Corección de las minucias  
  
# -----  
  
#Función para invertir una imagen (valores) que está en escala de 0 a 255  
def invertir (im):  
    im2=(255-im)
```

```

    return im2

#Función para invertir una imagen (valores) que está en escala de 0.0 a 1.0
def invertird (im):
    im2=(1-im)
    return im2

#Proceso completo de extracción de minucias
def extraerMinucias(image):
    # Normalizacion de la imagen
    # La imagen se normaliza con media y desviación estándar requerida
    t0=time.time() #Calculo el tiempo de ejecución en esta sección del código
    reqmean = 0 #Media que quiero que tenga mi imagen
    reqvar = 1 #Varianza que quiero que tenga mi imagen
    thresh = 0.25 #Threshold permitido para el umbral de varianza mi zona de trabajo (cambia para cada sensor)

    filas, columnas = image.shape #Obtengo el tamaño de la imagen
    image = Seg.im2double(image) #Convierto la imagen a double
    means = image.mean() #Calculo la media de la imagen.
    stds = image.std() #Calculo la desviación de la imagen
    im = image - means #Le resto a la imagen la media
    im = im / stds #Divido la imagen sobre la desviación estándar
    a=reqmean + im * np.sqrt(reqvar) #Creo mi nueva imagen con los parámetros dados
    t11=time.time() #Calculo el tiempo de ejecución hasta este punto del código
    #Imprimo el tiempo de ejecución del código

    # Segmentación de la imagen
    B1 = np.zeros((filas, columnas)) #Creo una imagen de ceros
    blksize = 36 #Defino el tamaño de mi bloque para realizar la segmentación
    imagen, B = Seg.segmentation(a, filas, columnas, blksize, B1, thresh, im) #Calculo la segmentación de mi imagen
    t1=time.time() #Calculo el tiempo de ejecución hasta este punto del código
    #Imprimo el tiempo de ejecución del código

    # Orientacion de la imagen

```

```
OrientSmoothSigma = 5 #Determino mi desviación estándar para el filtro gaussiano que se aplicará a la matriz de orientación
```

```
Or = OrS.Orient(imagen, OrientSmoothSigma) #Calculo la orientación de la imagen.
```

```
#print "Image orientation calculated = {}".format(time.time()-t1) #Imprimo un mensaje en pantalla para avisar que ya se hizo la orientación
```

```
t2=time.time() #Calculo el tiempo de ejecución hasta este punto del código
```

```
#Imprimo el tiempo de ejecución del código
```

```
# Cálculo de la frecuencia de la imagen
```

```
blksize = 36 #Defino el tamaño de mi bloque para realizar la segmentación
```

```
freq, medfreq = RidgeFreq.rf(imagen, B, Or, blksize, 3, 2, 15) #Calculo la matrix de frecuencias y la mediana de la frecuencia de la imagen
```

```
t3=time.time() #Calculo el tiempo de ejecución hasta este punto del código
```

```
#Imprimo el tiempo de ejecución del códigotime.time()-t2go
```

```
freq = medfreq * B/3.5 #Se demostró empíricamente que el filtro funciona mejor con la mediana de la frecuencia de la imagen sobre 3,5
```

```
#mpimg.imsave("Frecuencia.png", freq, cmap="gray") #Se guarda la matrix de frecuencia como una imagen
```

```
# Filtrado de la imagen
```

```
newim = RidgeFilter.ridgefilter(imagen, Or, freq, 0.5, 0.5, 1) #Se hace el filtro gabor de la imagen
```

```
#print "Gabor filter done = {}".format(time.time()-t3) #Imprimo un mensaje en pantalla para avisar que ya se hizo el filtro
```

```
t4=time.time() #Calculo el tiempo de ejecución hasta este punto del código
```

```
#Imprimo el tiempo de ejecución del código
```

```
#Para binarizar la imagen se aplica un threshold
```

```
mt = np.nonzero(newim > 0)
```

```
newim[mt]=255
```

```
mt = np.nonzero(newim <= 0)
```

```
newim[mt] = 0
```

```
# Thining de la imagen
```

```
newim = np.asarray(newim, dtype='uint8') #Convierto la imagen en uint8
```

```
imthin = thinning.guo_hall_thinning(invertir(newim)) #Hago el thinning de la imagen con la función que tiene python
```

```
#print "Thining done = {}".format(time.time()-t4) #Imprimo un mensaje en pantalla para avisar que ya se hizo el thinning
```

```

t5=time.time() #Calculo el tiempo de ejecución hasta este punto del código
#Imprimo el tiempo de ejecución del código
img=invertir(imthin*B) #Invierto la imagen ya que el thinning la entrega invertida

# Cálculo de minucias
result, PosT, PosB = Minucias.BuscarMinucias(img, B) #Busco las minucias en la imagen
#print "Minutiaes calculated = {}".format(time.time()-t5) #Imprimo un mensaje en pantalla para avisar que
ya se encontraron las minucias
t6=time.time() #Calculo el tiempo de ejecución hasta este punto del código
#Imprimo el tiempo de ejecución del código

#En un bloque de tamaño blksizeC solo quiero critB bifurcaciones y critT terminaciones
blksizeC=36
critB = 1
critT = 1

#Defino un bloque de búsqueda y defino mis criterios de búsqueda, siguiendo el ejemplo, solo voy a
permitir una bifurcación y o una terminación dentro de ese bloque

#Las repetidas las elimino
PosT, PosB = CorreccionM.correccion(PosT, PosB, blksizeC, critB, critT) #Corrijo las minucias con los
criterios anteriormente dados

#Creo una variable temporal para guardar mi matrix corregida
tempos=np.nonzero(PosT)

#En una matrix aparte guardo todas las bifurcaciones y en otra todas las terminaciones.

#Col 1 Pos X
#Col 2 Pos Y
#Col 3 Tipo de minucia 1 Terminación, 3 Bifurcación
#Col 4 Orientación de la minucia
MatrixT=np.zeros((tempos[0].size,4))
MatrixT[0:len(tempos[0]),0]=tempos[0]
MatrixT[0:len(tempos[0]),1]=tempos[1]
MatrixT[0:len(tempos[0]),2]=1
MatrixT[0:len(tempos[0]),3]=Or[tempos]

tempos=np.nonzero(PosB)
MatrixB=np.zeros((tempos[0].size,4))

```

```
MatrixB[0:len(tempos[0]),0]=tempos[0]
MatrixB[0:len(tempos[0]),1]=tempos[1]
MatrixB[0:len(tempos[0]),2]=3
MatrixB[0:len(tempos[0]),3]=Or[tempos]

#Nótese que la T y la B de las matrices de arriba son Terminación y Bifurcación, respetivamente.
MatrixM=np.concatenate((MatrixT, MatrixB), axis=0) #Concateno dos matrices.

#result = minucias.BuscarMinuciasC(img,PosT,PosB)

print "Total time = {}" .format(time.time()-t0) #Imprimo un mensaje en pantalla para avisar que ya se se
realizó todo el proceso

#Imprimo el tiempo de ejecución del código

#result.show() #Muestro en pantalla la imagen con minucias corregidas

return MatrixM #Retorno la matrix de minucias.
```

Anexo C. Segmentación.

Nombre archivo: Seg.py

```
# -*- encoding: utf-8 -*-
import numpy as np
import matplotlib.image as mpimg

# -----

#for i in range(0, int(np.fix(filas/blksize)*blksize)+blksize, blksize):
#    for j in range(0, int(np.fix(columnas/blksize)*blksize)+blksize, blksize):
#
def im2double(im):
    #Función para hacer la conversión de una imange uint8 a double
    min_val = np.min(im.ravel())
    max_val = np.max(im.ravel())
    out = (im.astype('float') - min_val) / (max_val - min_val)
    return out

def im2int(im):
    #Función para convertir una imagen a uint8
    out = (im.astype('int') * 255)
    return out

# Segmentación de la imagen
# Cálculo de la zona de trabajo
def segmentation(a, filas, columnas, blksize, B, thresh, im):
    fil=int(np.fix(filas/blksize)*blksize)+blksize
    col=int(np.fix(columnas/blksize)*blksize)+blksize
```

#Rocorro la imagen completamente, bloque por bloque, calculando la varianza del bloque, ese valor de varianza lo guardo

#en una matriz, en la posición que ocupaba el bloque en la imagen original

#Además se aplica el threshold, si la varianza es menor a mi thresh, reemplace por 0, sino, reemplace con 1.

#:[:,

for i in range(0, fil, blksize):

for j in range(0, col, blksize):

if ((i + blksize) < filas) & ((j + blksize) < columnas):#revisa bloque por bloque calculando la varianza

if (np.vectorize(a[range(i, i + blksize, 1),:][:,range(j, j + blksize, 1)].var())) <= thresh):

B[i:i+blksize,j:j+blksize].fill(0)

else:

B[i:i+blksize,j:j+blksize].fill(1)

if ((i + blksize) > filas) and ((j + blksize) > columnas):#si el bloque pasa las filas y las columnas tomamos la posición final menos blksize

if (np.vectorize(a[range(filas - blksize, filas, 1),:][:, range(columnas - blksize, columnas, 1)].var())) <= thresh):

B[filas-bksize:filas, columnas:columnas - blksize].fill(0)

else:

B[filas-bksize:filas, columnas:columnas - blksize].fill(1)

if ((i + blksize) > filas) and ((j + blksize) < columnas):#si el bloque pasa las filas, tomamos la posición final del bloque en filas

if (np.vectorize(a[range(filas - blksize, filas, 1), :][:,range(j, j + blksize, 1)].var())) <= thresh):

B[filas-bksize:filas, j:j + blksize].fill(0)

else:

B[filas-bksize:filas, j:j + blksize].fill(1)

if ((j + blksize) > columnas) and ((i + blksize) < filas):#si el bloque pasa las filas, tomamos la posición final del bloque en columnas

if (np.vectorize(a[range(i, i + blksize, 1), :][:,range(columnas - blksize, columnas, 1)].var())) <= thresh):

B[i:i + blksize, columnas:columnas - blksize].fill(0)

else:

B[i:i + blksize, columnas:columnas - blksize].fill(1)

Threshold de la segmentación

mt = np.nonzero(B > 0)

Máscara de la huella: zona de trabajo

Normalización de la zona de trabajo


```
mediah = np.mean(im[mt[0], mt[1]])
```

```
im1 = im - mediah
```

```
stdh = np.std(im1[mt[0], mt[1]])
```

```
im1 = im1 / stdh
```

```
im2 = im2double(im1)
```

#Ya teníamos normalizada la imagen, pero habia zonas basura que no nos interesan, ahora se normaliza la zona de trabajo de la imagen.

```
return im2, B
```

Anexo D. Estimacion de la orientación.

Nombre Archivo: OrS.py

```
# -*- encoding: utf-8 -*-  
  
import numpy as np  
from scipy.ndimage.filters import gaussian_filter  
import cv2  
  
def Orient(im, OrientSmoothSigma):  
    #Calculo el sobel en x y en y de toda la imagen  
    sobelx = cv2.Sobel(im, cv2.CV_64F, 1, 0, ksize=3)  
    sobely = cv2.Sobel(im, cv2.CV_64F, 0, 1, ksize=3)  
    #Aplico la fórmula del libro  
    Vx=2*np.multiply(sobelx, sobely)  
    Vy=np.multiply(sobelx, sobelx)-np.multiply(sobely, sobely)  
    #Obtengo mi matrix de orientación  
    orient=0.5*np.arctan2((Vx),(Vy))  
    #Calculo los ángulos en x y en y, esto para evitar pérdidas de datos  
    Ox = np.cos(2 * orient)  
    Oy = np.sin(2 * orient)  
    #Hago un filtro gaussiano para suavizar la matriz  
    sin2theta = gaussian_filter(Oy, OrientSmoothSigma, 0)  
    cos2theta = gaussian_filter(Ox, OrientSmoothSigma, 0)  
    #Reconstruyo mi matrix de orientación  
    orient = (np.arctan2(sin2theta, cos2theta) / 2)  
    return orient
```

Anexo E. Estimación de la frecuencia.

Nombre Archivo: RidgeFreq.py

```
# -*- encoding: utf-8 -*-  
  
import numpy as np  
import cv2  
import matplotlib.pyplot as plt  
import matplotlib.cm as cm  
# -----  
  
def ordfilt2(A, order):  
    result = np.zeros((int(len(A)),1))  
    for n in range(0, int(len(A)), 1):  
        if n + order < int(len(A)):  
            band = A[range(n, n + order, 1)]  
            result[n, 0] = band.max()  
        else:  
            band = A[range(n, int(len(A)), 1)]  
            result[n, 0] = band.max()  
    return result  
  
def rotateImage(image, angle):  
    #Función para rotar imagen con respecto al centro de la imagen  
    image_center = tuple(np.array(image.shape)/2)  
    rot_mat = cv2.getRotationMatrix2D(image_center, angle, 1.0)  
    result = cv2.warpAffine(image, rot_mat, image.shape, flags=cv2.INTER_LINEAR)  
    return result
```

```

def frequest(im, orientim, windsze, minWaveLength, maxWaveLength):
    orientim = 2 * np.array(orientim).ravel()#ravel reordena una matrix y la deja como un vector
    cosorient = np.mean(np.cos(orientim))
    sinorient = np.mean(np.sin(orientim))
    orient = np.arctan2(sinorient,cosorient) / 2 # se halla el ángulo promedio del bloque, se hace promediando
    los senos y cosenos
    #de los ángulos doblados antes de la reconstrucción del ángulo de nuevo
    rotim = rotateImage(im, orient/np.pi*180)#se rota la imagen para que las crestas queden verticales
    filas = len(im)
    columnas = len(im[0])
    #Se recorta la imagen rotada para que no tenga regiones inválidas
    cropsze = int(np.fix(filas / np.sqrt(2)))
    offset = int(np.fix((filas - cropsze) / 2))
    rotim = rotim[range(offset-1, offset + cropsze+1, 1), :][:, range(offset-1, offset + cropsze+1, 1)]
    #se suman las columnas para obtener una proyección de los valores de los pixeles en las crestas
    proj = rotim.sum(axis=1)
    #Se buscan picos en la proyección haciendo una dilatación y luego encontrando donde
    #la dilatación es igual a los valores originales
    dilation = ordfilt2(proj, windsze)
    maxpts = np.zeros((len(dilation), 1))
    for m in range(0, len(dilation), 1):
        if dilation[m] == proj[m] and proj[m] > np.mean(proj):
            maxpts[m] = 1
        else:
            maxpts[m] = 0
    maxind = np.nonzero(maxpts)
    #Se determina la frecuencia espacial de las crestas dividiendo la distancia entre el primer y último pico
    #sobre el número de picos -1. Si no hay picos en el bloque o si la longitud de onda es menor o mayor que
    el
    #rango permitido, se establece la frecuencia = 0
    if len(maxind[0]) < 2:
        freqim = 0
    else:
        NoOfPeaks = len(maxind[0])

```

```
waveLength = float((maxind[0][NoOfPeaks-1]-maxind[0][0])/float((NoOfPeaks-1)))
```

```
if (waveLength > minWaveLength) and (waveLength < maxWaveLength):
```

```
    freqim = float(1/waveLength)
```

```
else:
```

```
    freqim = 0
```

```
return freqim
```

```
def rf(im, mask, orientim, blksize, windsze, minWL, maxWL):
```

```
    #Se hace el cálculo de la frecuencia en toda la imagen, recorriendola bloque por bloque
```

```
    #Plica lo mismo que para la segmentación, no se sobrelapan los bloques, excepto al final cuando el bloque es de mayor tamaño
```

```
    #a la zona de imagen faltante
```

```
    filas, columnas = im.shape
```

```
    freq = np.zeros((len(im), len(im[0])))
```

```
    for r in range(0, filas-blksize, blksize):
```

```
        for c in range(0, columnas-blksize, blksize):
```

```
            if ((r + blksize) < filas) & ((c + blksize) < columnas):
```

```
                blkim = im[range(r, r+blksize, 1), :][:, range(c, c+blksize, 1)]
```

```
                blkor = orientim[range(r, r+blksize, 1), :][:, range(c, c+blksize, 1)]
```

```
                freq[r:r+blksize, c:c+blksize] = freqest(blkim, blkor, windsze, minWL, maxWL)
```

```
            if ((r + blksize) > filas) and ((c + blksize) > columnas):
```

```
                blkim = im[range(filas-blksize, blksize, 1), :][:, range(columnas-blksize, columnas, 1)]
```

```
                blkor = orientim[range(filas-blksize, blksize, 1), :][:, range(columnas-blksize, columnas, 1)]
```

```
                freq[filas-blksize:filas, columnas-blksize:columnas] = freqest(blkim, blkor, windsze, minWL, maxWL)
```

```
            if ((r + blksize) > filas) and ((c + blksize) < columnas):
```

```
                blkim = im[range(r, r+blksize, 1), :][:, range(columnas-blksize, columnas, 1)]
```

```
                blkor = orientim[range(r, r+blksize, 1), :][:, range(columnas-blksize, columnas, 1)]
```

```
                freq[filas - blksize:filas, columnas - blksize:columnas] = freqest(blkim, blkor, windsze, minWL, maxWL)
```

```
            if ((c + blksize) > columnas) and ((r + blksize) < filas):
```

```
                blkim = im[range(filas-blksize, blksize, 1), :][:, range(c, c+blksize, 1)]
```

```
                blkor = orientim[range(filas-blksize, blksize, 1), :][:, range(c, c+blksize, 1)]
```

```
                freq[filas - blksize:filas, columnas - blksize:columnas] = freqest(blkim, blkor, windsze, minWL, maxWL)
```

```
    freq1 = freq * mask #Elimino la frecuencia que no está en mi zona de trabajo
```

```

x1 = np.nonzero(freq1 > 0) #Busco las frecuencias mayores a 0
medianfreq = np.median(freq1[x1]) #Y calculo la mediana de las frecuencias anteriormente halladas
return freq, medianfreq

```

Anexo F. Aplicación del filtro Gabor.

Nombre Archivo: RidgeFilter.py

```

# -*- encoding: utf-8 -*-
import numpy as np
import cv2
import time
import timeit
import warnings

def fxn():
    warnings.warn("deprecated", DeprecationWarning)

with warnings.catch_warnings():
    warnings.simplefilter("ignore")
    fxn()

# -----
#@numba.jit()
def im2double(im):
    min_val = np.min(im.ravel())
    max_val = np.max(im.ravel())
    out = (im.astype('float') - min_val) / (max_val - min_val)
    return out

#@numba.jit()
def rotateImage(image, angle):
    image_center = tuple(np.array(image.shape)/2)
    rot_mat = cv2.getRotationMatrix2D(image_center, angle, 1.0)
    result = cv2.warpAffine(image, rot_mat, image.shape, flags=cv2.INTER_LINEAR)
    return result

```

```
#@numba.jit()
```

```
def ridgefilter(im, orient, freq, kx, ky, showfilter):
```

```
    #Se determina un incremento de 3° para las orientaciones del filtro
```

```
    angleInc = 3
```

```
    im = im2double(im)
```

```
    filas = len(im)
```

```
    columnas = len(im[0])
```

```
    #Se encuentra donde los valores de frecuencia son válidos
```

```
    x1 = np.nonzero(freq > 0)
```

```
    validr = x1[0]
```

```
    validc = x1[1]
```

```
    #Se redondea el array de frecuencias a la centésima más cercana (0.01) para reducir el número de frecuencias
```

```
    #presentes en el vector
```

```
    freq = np.round(freq*100)/100
```

```
    #Se genera un array de las frecuencias únicas en la imagen
```

```
    unfreq = np.unique(freq)
```

```
    unfreq = unfreq[1:len(unfreq)]
```

```
    #Se genera una tabla, dada la frecuencia multiplicada por 100 para obtener un índice entero, y se le asigna esa frecuencia
```

```
    #multiplicada al índice
```

```
    freqindex = np.ones((len(unfreq), 1))
```

```
    #Se generan los filtros correspondientes a las distintas frecuencias y orientaciones (con los incrementos mencionados antes)
```

```
    sze = np.zeros((len(unfreq), 1))
```

```
    sigmax = 1 / unfreq[0] * kx
```

```
    sigmay = 1 / unfreq[0] * ky
```

```
    sze[0] = np.round(1* max(sigmax, sigmay))
```

```
    dtype_filter = "(" + str(int(sze[0]*2+1)) + ", " + str(int(sze[0]*2+1)) + ")float64"
```

```
    sigmax = 1/unfreq*kx
```

```
    sigmay = 1/unfreq*ky
```

```
    sze = np.round(3*max(sigmax, sigmay))
```

```
    x = np.linspace(-sze, sze, sze*2+1)
```

```
    y = np.linspace(-sze, sze, sze*2+1)
```

```
    [x, y] = np.meshgrid(x, y)
```

```

reffilter = np.exp(-(x**2/sigmax**2 + y**2/sigmay**2)/2)*np.cos(2*np.pi*unfreq*x)

#Cada filtro para cada frecuencia se rota angleInc veces en sentido antihorario (por eso el -)
o=0
filter=np.zeros((180/angleInc,len(x),len(y)))
while(o<180/angleInc):
    #for o in range(1, 180/angleInc+1, 1):
        tempimg = rotateImage(reffilter, -(o-1)*angleInc)
        filter[o - 1, 0:len(x), 0:len(y)]=tempimg
        o += 1

#Se buscan los indices de la matrix de puntos mas grandes que maxsize
maxsize = size

finalind1=np.asarray(np.all((((validr>maxsize),(validr<(filas-maxsize)),(validc>maxsize),(validc<(columnas-
maxsize)))),axis=0))

finalind=np.nonzero(finalind1==True)

#Se convierten los valores de la orientación en radianes a un valor entero que corresponda o más se
acerque a los

#angulos calculados por round/degrees/angleInc
maxorientindex = np.round(180/angleInc)
orientindex = np.round(orient/np.pi*180/angleInc)
i = np.nonzero(orientindex < 1)
#for up in range(0,len(i)-1,1):
    orientindex[i] = orientindex[i]+maxorientindex
i = np.nonzero(orientindex > maxorientindex)
#for up in range(0,len(i)-1,1):
    orientindex[i] = orientindex[i]-maxorientindex

# Filtrado
newimg=np.zeros((filas, columnas))
size=int(size)
#freqindex=int(freqindex)
im1 = np.zeros((filas + 2 * size, columnas + 2 * size))
im1[size:filas + size, size:columnas + size] = im

s=size
k = 0
newim=np.zeros((s,s))
while (k < finalind[0].size):

```



```
r = validr[(finalind[0][k])]
```

```
c = validc[(finalind[0][k])]
```

#Para cada zona válida de la imagen, habrá un valor en frecuencia y orientación, de mi matrix de orientación y mi matrix de frecuencia

#obtengo los datos y realizo el filtro. No solo se toma el punto, se toma un bloque con el punto a filtrar como centro

```
newim = np.multiply(filter[int(orientindex[r, c] - 1), :, :], im1[r - s + s:r + s + s + 1, c - s + s:c + s + s + 1])
```

```
newimg[r, c] = newim.sum()#Se hace una sumatoria de los valores del bloque filtrado
```

```
k += 1
```

```
return newimg
```

Anexo G. Código para hallar minucias.

Nombre Archivo: Minucias.py

```
# -*- encoding: utf-8 -*-
# Metody biometryczne
# Przemyslaw Pastuszka

from PIL import Image, ImageDraw
import numpy as np
import Seg

cells = [(-1, -1), (-1, 0), (-1, 1), (0, 1), (1, 1), (1, 0), (1, -1), (0, -1), (-1, -1)]

def invertird (im):
    im2=(1-im)
    return im2

def BuscarM(pixels, i, j):
    #El cálculo de minucias se hace por sumatorias
    values = [pixels[i + k, j + l] for k, l in cells]
    crossings = 0
    for k in range(0, 8):
        crossings += abs(values[k] - values[k + 1])
    crossings /= 2
    if pixels[i, j] == 1:
        if crossings == 1:
            return "terminacion"
        if crossings == 3:
            return "bifurcacion"
    return "nada"
```

```

def BuscarMinuciasC(im, PosT, PosB):
    pixels = Seg.im2double(im)
    pixels = invertird(pixels)
    x, y = im.shape
    im = Image.fromarray(np.uint8(im))
    result = im.convert("RGB")
    draw = ImageDraw.Draw(result)
    ellipse_size = 2
    for i in range(5, x - 5):
        for j in range(5, y - 5):
            if PosT[i,j] == 1:
                draw.ellipse([(j - ellipse_size, i - ellipse_size), (j + ellipse_size, i + ellipse_size)],
                              outline=(150, 0, 0))
            else:
                if PosB[i,j] == 3:
                    draw.ellipse([(j - ellipse_size, i - ellipse_size), (j + ellipse_size, i + ellipse_size)],
                                  outline=(0, 150, 0))

    del draw
    return result

def BuscarMinucias(im, B):
    pixels = Seg.im2double(im)#Convierto la imagen a double
    pixels = invertird(pixels)#Invierto la imagen
    x, y = im.shape#Obtengo el tamaño de la imagen
    im = Image.fromarray(np.uint8(im))#Convierto la imagen a uint8
    #result = im.convert("RGB")
    #draw = ImageDraw.Draw(result)
    #colors = {"terminacion": (150, 0, 0), "bifurcacion": (0, 150, 0)}
    #ellipse_size = 2
    resultadoB = np.zeros((x, y))#Creo una matriz de ceros para almacenar las minucias encontradas
    resultadoT = np.zeros((x, y))
    for i in range(5, x - 5):#Recorro toda la imagen buscando minucias
        for j in range(5, y - 5):

```

#El condicional es para decir que si hay una terminación o una bifurcación, que esté alejada del borde de mi zona de trabajo, esto para eliminar falsas minucias

```
if (B[i,j]==1 and B[i-5,j]==1 and B[i,j-5]==1 and B[i+5,j]==1 and B[i,j+5]==1 and B[i+5,j+5]==1 and B[i-5,j+5]==1 and B[i-5,j-5]==1 and B[i+5,j-5]==1):
```

#Con esto evito que tome como "terminaciones" las zonas donde termina la zona de trabajo

```
minucias = BuscarM(pixels, i, j)
```

```
if minucias == "terminacion": #Clasifico mi tipo de minucia, 1 terminacion, 3 bifurcacion
```

```
    resultadoT[i,j] = 1
```

```
else:
```

```
    if minucias == "bifurcacion":
```

```
        resultadoB[i,j] = 3
```

```
#if minucias != "nada":
```

```
    # draw.ellipse([(j - ellipse_size, i - ellipse_size), (j + ellipse_size, i + ellipse_size)], outline = colors[minucias])
```

```
return result, resultadoT, resultadoB
```

Anexo H. Código para cambiar el origen de la huella y ubicarlo en la minucia a comparar.

Transformada.py

```
# -*- encoding: utf-8 -*-
```

```
import numpy as np
```

```
def Transform( M, i ):
```

```
    #La transformada que se hace es una transformada de rotación sobre el eje z, esto para crear un nuevo  
    sistema de referencia en mi minucia
```

```
    x1,Count,var=M.shape
```

```
    XRef=M[0,i,0]
```

```
    YRef=M[0,i,1]
```

```
    ThRef=M[0,i,3]
```

```
    T=np.zeros((Count,4))
```

```
    R=[[np.cos(ThRef),np.sin(ThRef),0],
```

```
        [-np.sin(ThRef), np.cos(ThRef), 0],
```

```
        [0, 0, 1]] #Transformation Matrix
```

```
    R = np.asarray(R)
```

```
    for i in range(0, Count-1, 1):
```

```
        B=[[M[0,i,0]-XRef],
```

```
            [M[0,i,1]-YRef],
```

```
            [M[0,i,3]-ThRef]]
```

```
        B=np.asarray(B)
```

```
        temp=np.dot(R,B)
```

```
        T[i, 0] = temp[0]
```

```
        T[i, 1] = temp[1]
```

```
        T[i, 2] = temp[2]
```

```
        T[i, 3] = M[0,i, 3]
```

```
    return T
```

Anexo I. Código para girar la huella a comparar n veces y hacer su comparación.

Transformada2.py

```
# -*- encoding: utf-8 -*-
import numpy as np

def Transform2( T, alpha ):
    #Transformada de rotación con respecto a un punto.
    Count, var=T.shape
    Tnew=np.zeros((Count,4))
    R=[[np.cos(alpha), np.sin(alpha),0, 0],
        [-np.sin(alpha), np.cos(alpha), 0, 0],
        [0, 0, 1, 0],
        [0, 0, 0, 1]]      # Transformation Matrix
    R = np.asarray(R)
    for i in range(0, Count-1,1):
        B=T[i,:]-[0, 0, alpha, 0]
        B = np.asarray(B)
        Tnew[i,:]=np.dot(R,np.transpose(B))
    return Tnew
```

Anexo J. Código que da el puntaje de match de las huellas comparadas.

Score.py

```
# -*- encoding: utf-8 -*-
```

```
import numpy as np
```

```
def Score(T1, T2):
```

```
    Count1, var1=T1.shape
```

```
    Count2, var2=T2.shape
```

```
    n=0
```

```
    T=15 #Threshold for distance
```

```
    TT=14#Threshold for theta
```

```
    for i in range(0,Count1-1, 1):
```

```
        Found=0
```

```
        j=0
```

```
        while (Found==0) and (j < Count2):
```

```
            dx=(T1[i,1]-T2[j,1])
```

```
            dy=(T1[i,2]-T2[j,2])
```

```
            d=np.sqrt(dx**2+dy**2) #Euclidean Distance between T1(i) & T2(i)
```

```
            if d<T: #Si la distancia cumple con mi criterio
```

```
                DTheta=abs(T1[i,3]-T2[j,3])*180/np.pi
```

```
                DTheta=min(DTheta,360-DTheta)
```

```
                if DTheta<TT: # y si el angulo de la minucia cumple con mi criterio
```

```
                    n=n+1      #Increase Score
```

```
                    Found=1
```

```
            j=j+1
```

```
    sm = np.sqrt(float(n)*float(n) / (float(Count1) * float(Count2))) # similarity Index
```

```
    #el indice de similitud es la raiz cuadrada del numero de similitudes encontradas al cuadrado, sobre la multiplicacion del numero de minucias de cada huella
```

```
    return sm
```

Anexo K.Codigo que elimina el exceso de minucias por bloques.

CorreccionM.py

```
# -*- encoding: utf-8 -*-
```

```
import numpy as np
```

```
def correccion(PosT, PosB, blksize, critB, critT):
```

```
    filas, columnas = PosT.shape
```

```
    for i in range(0, int(np.fix( filas /blksize ) *blksize ) +blksize, blksize):
```

```
        for j in range(0, int(np.fix( columnas /blksize ) *blksize ) +blksize, blksize):
```

```
            if ((i + blksize) < filas) & ((j + blksize) < columnas): # revisa bloque por bloque calculando el número de minucias existentes, si ese número es mayor a mi criterio
```

```
                #las elimino
```

```
                irange = range(i, i + blksize, 1)
```

```
                jrange = range(j, j + blksize, 1)
```

```
                x = PosT[irange, :][:, jrange]
```

```
                y = PosB[irange, :][:, jrange]
```

```
                x1 = np.nonzero(x)
```

```
                y1 = np.nonzero(y)
```

```
                if x1[0].size > critT:
```

```
                    PosT[i:i+blksize, j:j+blksize] = 0
```

```
                if y1[0].size > critB:
```

```
                    PosB[i:i+blksize, j:j+blksize] = 0
```

```
            # si el bloque pasa las filas y las columnas tomamos la posición final menos blksize
```

```
            if ((i + blksize) > filas) and ((j + blksize) > columnas):
```

```
                irange = range(filas - blksize, filas, 1)
```

```
                jrange = range(columnas - blksize, columnas, 1)
```

```
                x = PosT[irange, :][:, jrange]
```

```
                y = PosB[irange, :][:, jrange]
```

```
                x1=np.nonzero(x)
```

```
                y1=np.nonzero(y)
```

```
                if x1[0].size > critT:
```

```
                    PosT[filas-bksize:filas, columnas:columnas - blksize] = 0
```

```
                if y1[0].size > critB:
```



```

        PosB[filas-blksize:filas, columnas:columnas - blksize] = 0

# si el bloque pasa las filas, tomamos la posición final del bloque en filas
if ((i + blksize) > filas) and ((j + blksize) < columnas):
    irange = range(filas - blksize, filas, 1)
    jrange = range(j, j + blksize, 1)
    x = PosT[irange, :][:, jrange]
    y = PosB[irange, :][:, jrange]
    x1 = np.nonzero(x)
    y1 = np.nonzero(y)
    if x1[0].size > critT:
        PosT[filas-blksize:filas, j:j + blksize] = 0
    if y1[0].size > critB:
        PosB[filas-blksize:filas, j:j + blksize] = 0

# si el bloque pasa las filas, tomamos la posición final del bloque en columnas
if ((j + blksize) > columnas) and ((i + blksize) < filas):
    irange = range(i, i + blksize, 1)
    jrange = range(columnas - blksize, columnas, 1)
    x = PosT[irange, :][:, jrange]
    y = PosB[irange, :][:, jrange]
    x1 = np.nonzero(x)
    y1 = np.nonzero(y)
    if x1[0].size > critT:
        PosT[i:i + blksize, columnas:columnas - blksize] = 0
    if y1[0].size > critB:
        PosB[i:i + blksize, columnas:columnas - blksize] = 0

return PosT, PosB

```

Anexo L. Código que realiza el Match de las huellas.

Match.py

```

# -*- encoding: utf-8 -*-

import Transformada
import Transformada2
import numpy as np
import Score

def matching(M1, M2):
    mt = np.nonzero(M1[:,2] >=1)#Busco las minucias que sean mayores o iguales a indice 1 (o sea, todas)
    M1=M1[mt,:]
    mt = np.nonzero(M2[:, 2] >=1)
    M2=M2[mt,:]
    x1, count1, var1 = M1.shape #calculo el numero de minucias
    x2, count2, var2 = M2.shape
    bi=0
    bj=0
    ba=0 #Best i,j,alpha
    S=0      #Best Similarity Score
    for i in range(0, count1-1, 1):
        T1=Transformada.Transform(M1,i)#Hago la transformada de mi minucia acutal
        for j in range(0, count2-1, 1):
            if M1[0,i,2] == M2[0,j,2]: #Si ambas minucias son del mismo tipo
                T2=Transformada.Transform(M2,j) #Hago la transformada de la siguiente minucia a comparar
                T3=Transformada2.Transform2(T2,np.pi/180)#Nuevamente haco una transformada a mi minucia
                comparada, para ver que tanto se parece a mi minucia base
                sm=Score.Score(T1,T3)#Calculo el puntaje de similitud
                if S<sm:
                    S=sm
                    #bi=i
                    #bj=j
                    #ba=a
    return S

```