



**UNIVERSIDAD PRIVADA DE TACNA**

**FACULTAD DE INGENIERIA**

**Escuela Profesional de Ingeniería de Sistemas**

**INFORME TAREA N.º 01**

**“Taller de reconocimiento de huella dactilar  
biométrica”**

Curso: Seguridad Informática

Docente: Ing. Oscar Jiménez Flores

**Aquino Huallpa, Yaneth Virginia (2017059286)**

**Tacna – Perú  
2020**

## **I. MARCO TEÓRICO**

*La biometría es un campo de la ingeniería que durante los últimos años está en una fase de expansión en aplicaciones que requieran seguridad en la autenticación del individuo autorizado.*

*Existen muchos tipos de biometría: cara, iris, firma, voz... aunque la que predomina es la de huella dactilar debido a sus inherentes ventajas, como son la elevada distintividad entre individuos, invariabilidad con el tiempo, comodidad de uso o existencia de sensores (no ópticos) de bajo coste e integrables en Si.*

*Hay dos etapas ejecutadas durante una autenticación biométrica:*

***la etapa de extracción de las características que se suponen distintivas entre individuos, y la etapa de 'matching' o comparación de estas características con una plantilla del individuo autorizado.** Las características extraídas de la huella suelen ser las minutias, que es el conjunto de puntos singulares donde las colinas de la huella acaban o bifurcan.*

***La etapa de extracción es, generalmente, bastante compleja y requiere de algoritmos de procesado que frecuentemente hacen uso intensivo de números flotantes con operaciones aritméticas y trigonométricas.***

*Los algoritmos biométricos han sido tradicionalmente implementados en software para ser ejecutados por plataformas basadas en microprocesadores de altas prestaciones y también coste. Sin embargo, en los últimos años, se han desarrollado sistemas embebidos de reconocimiento biométrico, generalmente de huella dactilar, que usan algoritmos propietarios para simplificar el procesado biométrico, con objeto de poder ser ejecutados en tiempo real sobre microprocesadores embebidos de medio coste, a costa de poder empeorar las prestaciones FAR/FFR del reconocimiento.*



### **Comparación de huellas digitales**

*Cada dedo presenta al menos una de estas características. Por otro lado, en determinados puntos las líneas de la huella dactilar se cortan bruscamente o se bifurcan. Estos puntos reciben el nombre de minucias, y juntos suman casi el 80% de los elementos singulares de una huella.*



*En el momento en que un usuario solicita ser identificado, coloca su dedo sobre un lector y su huella dactilar es escaneada y analizada con el fin de extraer los elementos característicos y buscar su homóloga en la base de datos. El resultado es un diagnóstico certero en más del 99% de los casos.*

### **Algoritmo de Esqueletización**

*La esqueletización (ver Fig. 1) busca representar una imagen binarizada (formada por pixels blancos y negros) por un grafo de grosor fino (típicamente 1 pixel de grosor) cuyos puntos cumplen que la distancia local respecto a los bordes de la imagen binarizada es máxima. La transformada MAT (Medial Axis Transformation) calcula para cada pixel de la imagen binarizada la distancia mínima respecto a los bordes, pudiéndose definir el esqueleto a partir de los pixels con máximas distancias locales. Sin embargo, la obtención del esqueleto usando esta transformada requiere de un coste computacional prohibitivo, por lo que se*

han desarrollado algoritmos de esqueletización que consiguen resultados similares a una velocidad muy superior.



Figura 1. Imagen binarizada de la huella (derecha), y su esqueletizado (izquierda)

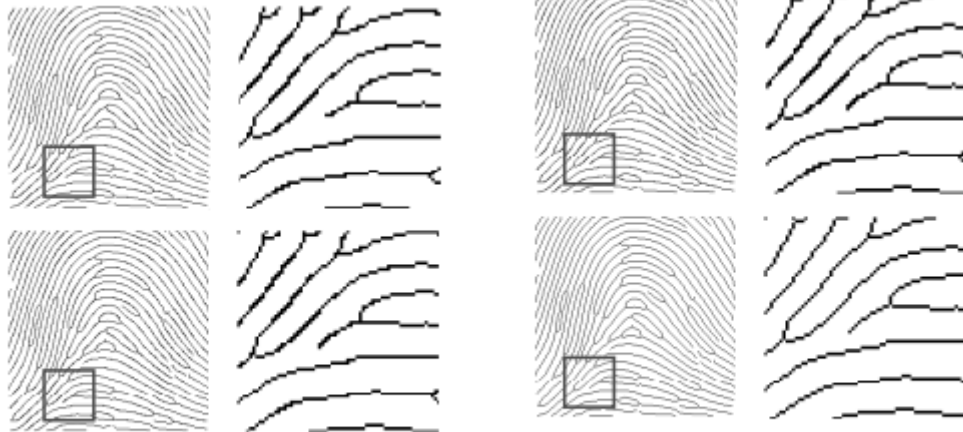


Figura 4. Esqueletización para el algoritmo [1] (arriba) o la variante [3] (abajo)

Figura 5. Efecto escalera (arriba) y su eliminación usando [4] (abajo)

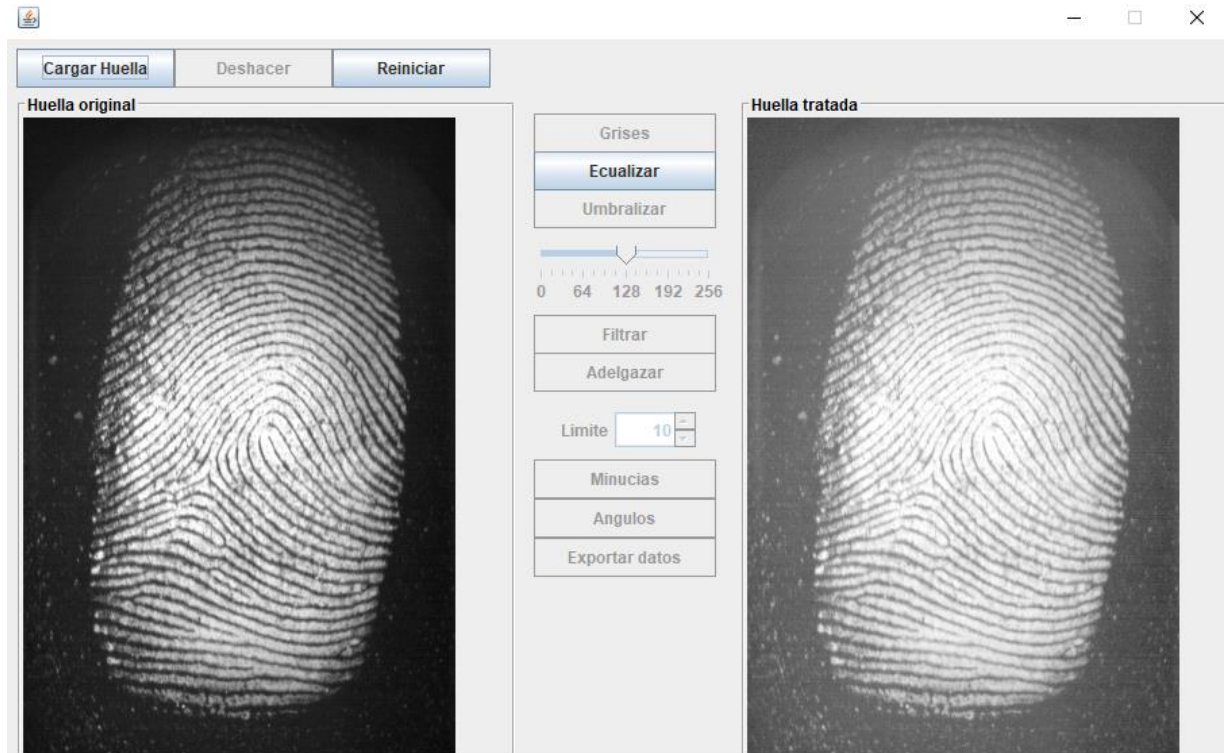
## II. DESARROLLO

El objetivo de la aplicación es realizar un preprocesado a la imagen de la huella dactilar para posteriormente identificar los rasgos que la caracterizan y diferencian, estas son las minucias. Con los datos obtenidos de la aplicación, se podría hacer una búsqueda en una base de datos de huellas dactilares para proceder a su identificación.

### III. RESULTADOS

Los tratamientos previos que realiza incluyen:

- **Conversión a escala de grises**



```

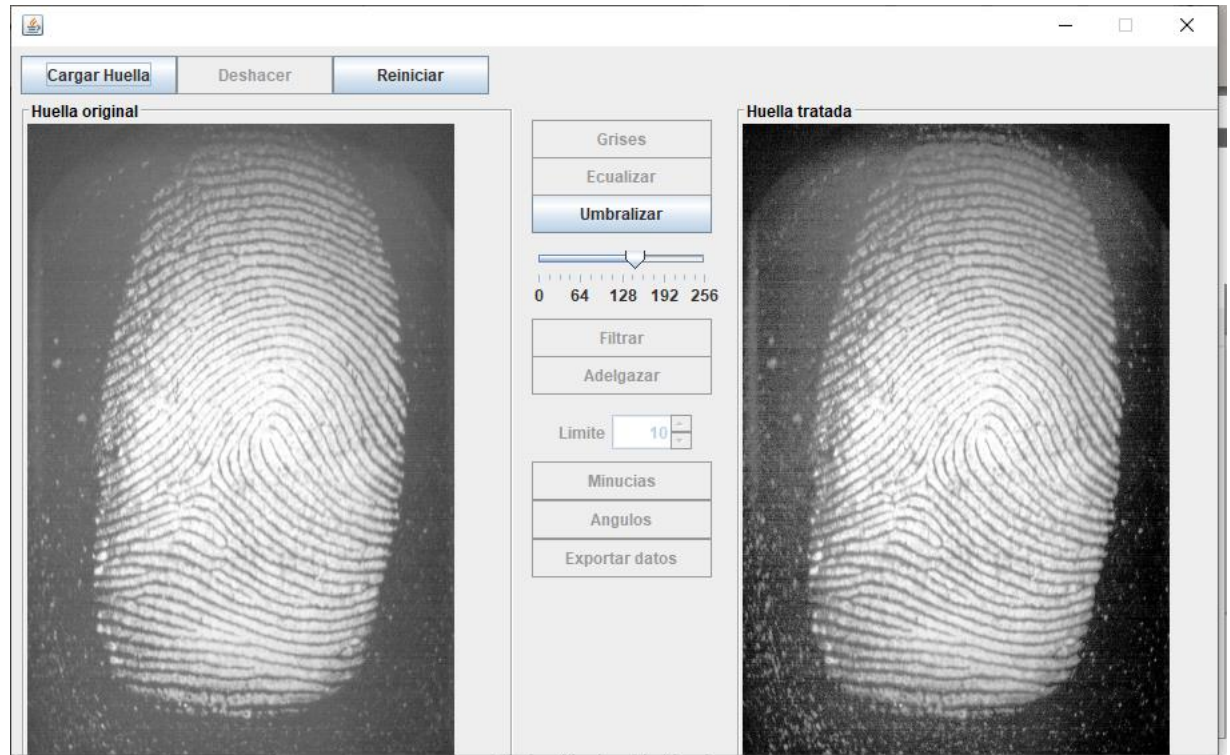
/**
 * Convierte una huella dactilar a BufferedImage para su visualizacion por pantalla
 * @param huellaEntrada la HuellaDactilar que se desea tratar
 * @param modo 0 si la huella esta en Blanco y Negro
 *             1 si la huella esta en escala de grises
 * @return BufferedImage que contiene al huella para visualizar
 */
public BufferedImage convertirRGB( HuellaDactilar imgEntrada , int modo ) {

    BufferedImage imgSalida = new BufferedImage(imgEntrada.getWidth(), imgEntrada.getHeight())

    for (int x = 0; x < imgEntrada.getWidth(); ++x) {
        for (int y = 0; y < imgEntrada.getHeight(); ++y){
            int valor = imgEntrada.getPixel(x, y);
            if(modo == 0){
                valor=valor*255;
            }
            int pixelRGB =(255<<24 | valor << 16 | valor << 8 | valor);
            imgSalida.setRGB(x, y, pixelRGB);
        }
    }
    return imgSalida;
}

```

## ■ Ecualizado



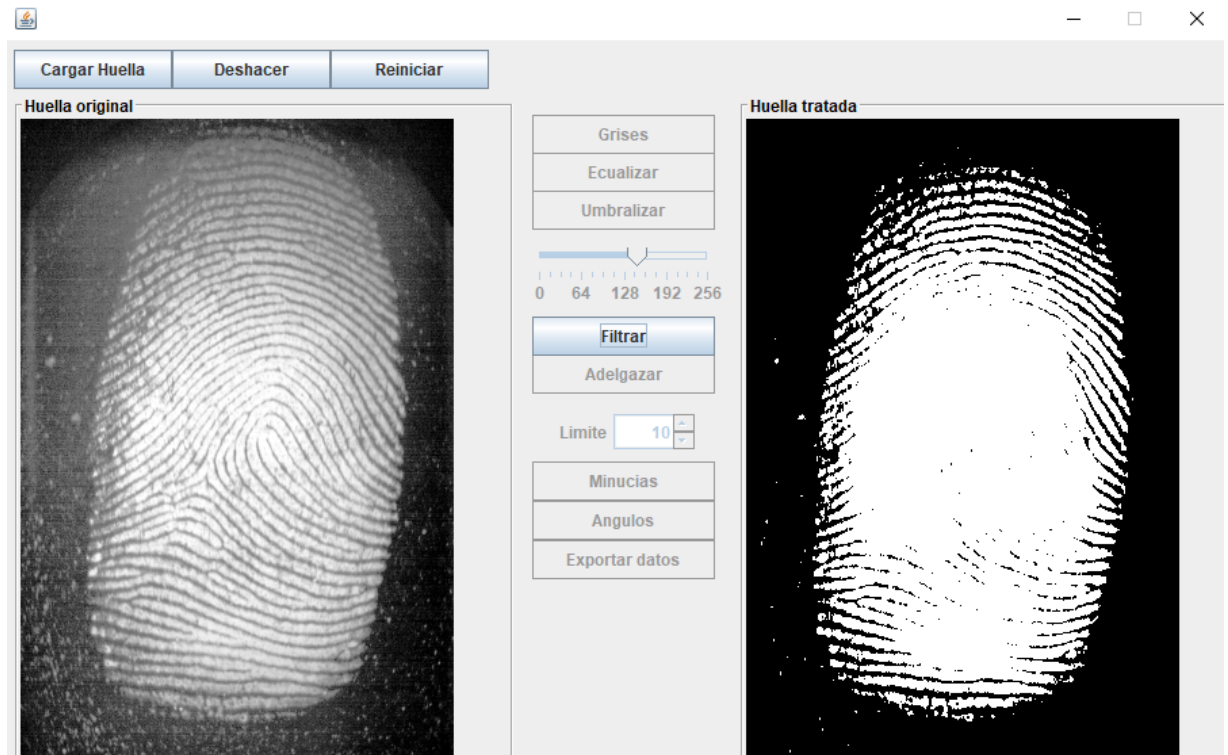
```

195  * Realiza el ecualizado de la huella de entrada
196  * @param huellaEntrada la HuellaDactilar que se desea tratar
197  * @return HuellaDactilar con los valores de grises normalizados según histograma
198  */
199  public HuellaDactilar ecualizado( HuellaDactilar imgEntrada ){
200
201      HuellaDactilar imgSalida = new HuellaDactilar(imgEntrada.getWidth(), imgEntrada.getHeight());
202      int width = imgEntrada.getWidth();
203      int height = imgEntrada.getHeight();
204      int tamPixel= width*height;
205      int[] histograma = new int[256];
206      int i =0;
207      // Calculamos frecuencia relativa de ocurrencia
208      // de los distintos niveles de gris en la imagen
209      for (int x = 0; x < width; x++){
210          for (int y = 0; y < height; y++){
211              int valor= imgEntrada.getPixel(x, y);
212              histograma[valor]++;
213          }
214      }
215      int sum =0;
216      // Construimos la Lookup table LUT containing scale factor
217      float[] lut = new float[256];
218      for ( i=0; i < 256; ++i ){
219          sum += histograma[i];
220          lut[i] = sum * 255 / tamPixel;
221      }
222      // Calculamos el umbral medio
223      int acumulador = 0;
224      int numPixels = width*height;
225      for (int x = 0; x < width; x++){
226          for (int y = 0; y < height; y++){
227              acumulador += imgEntrada.getPixel(x, y);
228          }
229      }
230      umbralMedio = acumulador/numPixels;
231      // Se transforma la imagen utilizando la tabla LUT
232
233      for (int x = 0; x < width; x++) {

```



### ■ Eliminación de ruido binario



```
/**
 * Metodo que convierte la imagen de la huella a una matriz de blancos y negros segun el umbral fijado
 * @param imgEntrada objeto HuellaDactilar que se desea umbralizar
 * @param umbral entero que representa el umbral a utilizar
 * @return objeto HuellaDactilar con la huella umbralizada a blancos y negros segun el umbral
 */
public HuellaDactilar umbralizar( HuellaDactilar imgEntrada , int umbral ){

    HuellaDactilar imgSalida = new HuellaDactilar( imgEntrada.getWidth() , imgEntrada.getHeight() );

    for( int x = 0 ; x < imgEntrada.getWidth() ; x++ ){
        for( int y = 0 ; y < imgEntrada.getHeight() ; y++ ){
            int valor = imgEntrada.getPixel( x , y );
            if( valor < umbral ){
                imgSalida.setPixel( x , y , 0 );
            } else {
                imgSalida.setPixel( x , y , 1 );
            }
        }
    }

    return imgSalida;
}
```

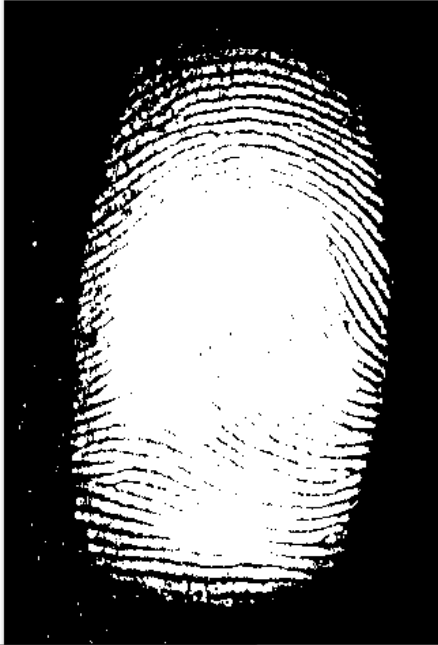
## ■ Filtrado de convolución

Cargar Huella

Deshacer

Reiniciar

Huella original



Grises

Ecualizar

Umbralizar

0 64 128 192 256

Filtrar

Adelgazar

Limite

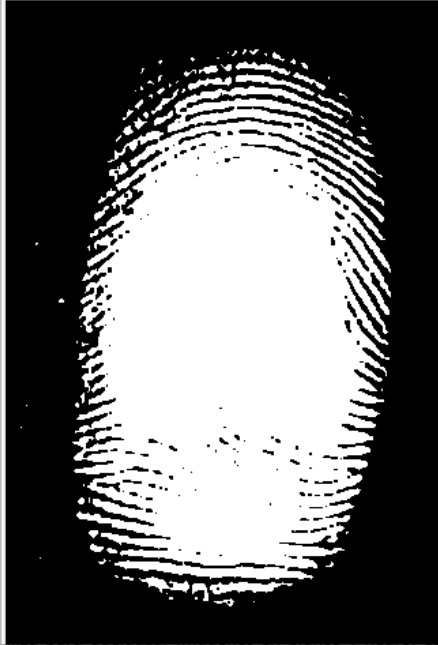
10

Minucias

Angulos

Exportar datos

Huella tratada



```

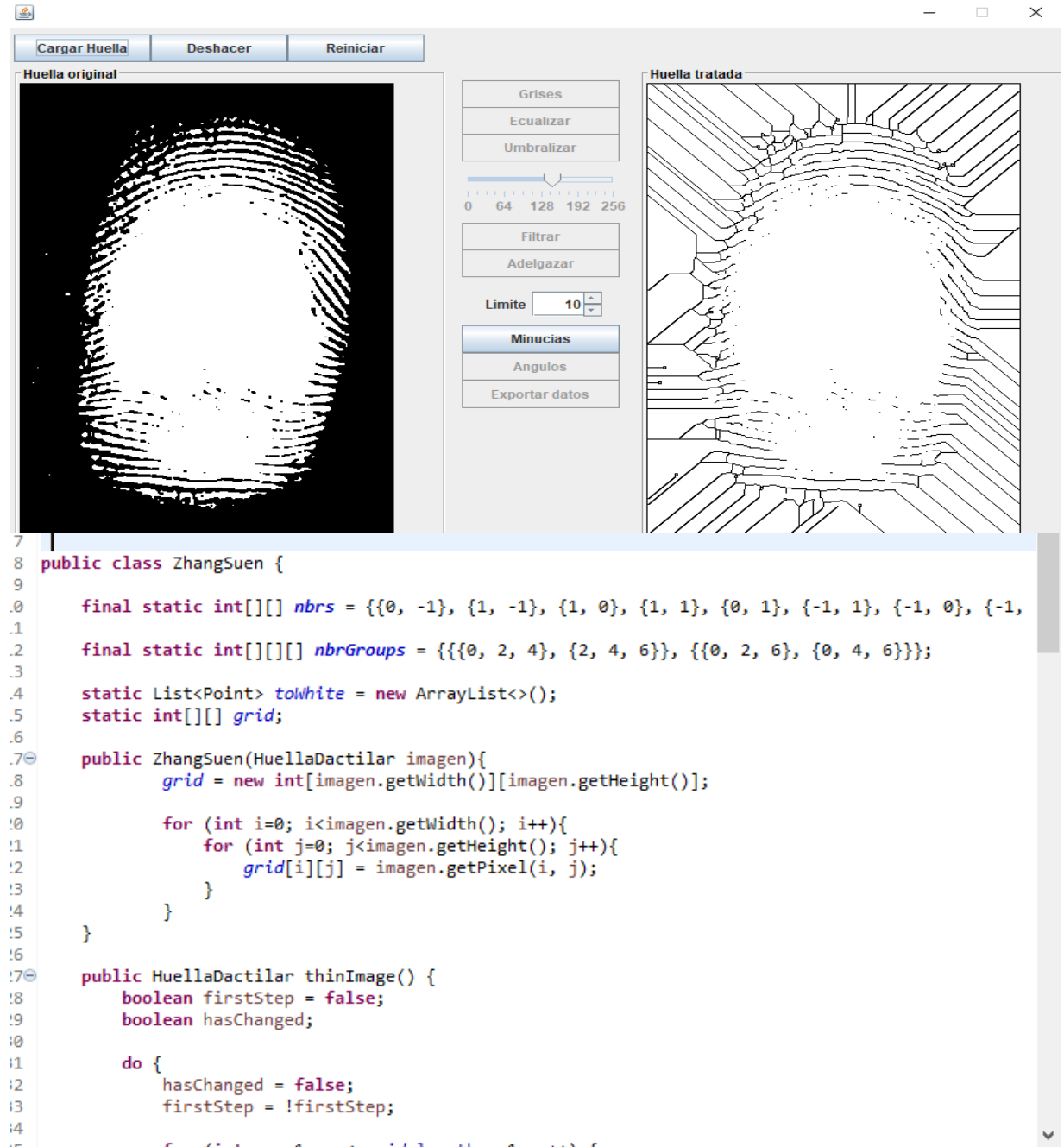
58  * Metodo para filtrar la HuellaDactilar
59  * @param imgEntrada la HuellaDactilar a filtrar
60  * @return objeto HuellaDactilar con la huella filtrada
61  */
62  public HuellaDactilar filtrar( HuellaDactilar imgEntrada ){
63
64      HuellaDactilar imgSalida = new HuellaDactilar( imgEntrada.getWidth() , imgEntrada.getHeight() );
65      HuellaDactilar imgAux = new HuellaDactilar( imgEntrada.getWidth() , imgEntrada.getHeight() );
66
67      int a, b, c ,d, e, f, g, h, p, B1, B2;
68
69      for( int x = 0 ; x < imgEntrada.getWidth() ; x++){
70          for( int y = 0 ; y < imgEntrada.getHeight() ; y++){
71              if( x > 0 && x < imgEntrada.getWidth()-1 && y > 0 && y < imgEntrada.getHeight()-1 ){
72
73                  a = imgEntrada.getPixel( x-1 , y-1 );
74                  b = imgEntrada.getPixel( x , y-1 );
75                  c = imgEntrada.getPixel( x+1 , y-1 );
76                  d = imgEntrada.getPixel( x-1 , y );
77                  e = imgEntrada.getPixel( x+1 , y );
78                  f = imgEntrada.getPixel( x-1 , y+1 );
79                  g = imgEntrada.getPixel( x , y+1 );
80                  h = imgEntrada.getPixel( x+1 , y+1 );
81                  p = imgEntrada.getPixel( x , y );
82
83                  //B1 = p+b.g.(d+e)+d.e.(b+g) + --> OR . --> AND
84                  B1 = p | b & g & ( d|e ) | d & e & ( b|g );
85              } else {
86                  B1 = imgEntrada.getPixel( x , y );
87              }
88              imgAux.setPixel( x , y , B1 );
89          }
90      }
91  }

```



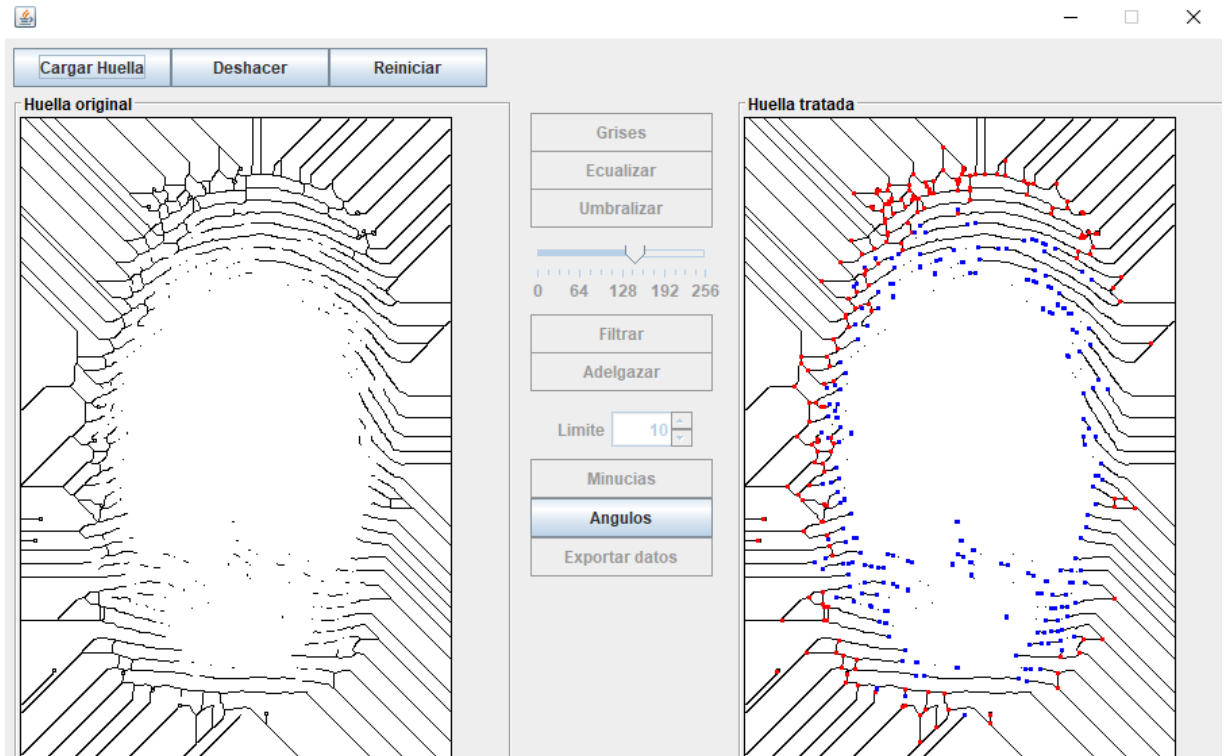
### ■ Adelgazado mediante el algoritmo de Zhang-Suen

Este es un algoritmo que se utiliza para adelgazar imágenes en blanco y negro, es decir, un bit por píxel.



Las operaciones posteriores a estos tratamientos son:

- **Detección de minucias**

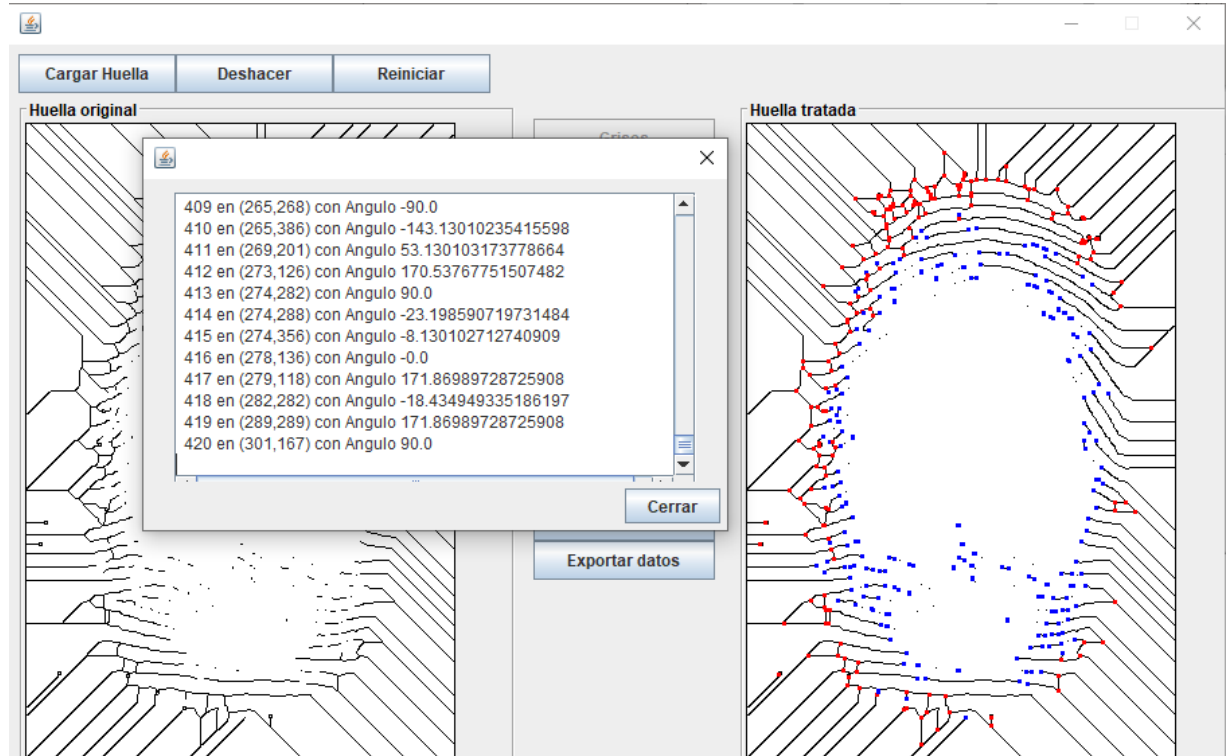


```

346  * Metodo que detecta las minucias de una huella adelgazada
347  * Las minucias devueltas tienen Crossing Numbre = 1 (corte o final) Æ 3 (bifurcaciÃ³n)
348  * @param imgEntrada la huella sobre la que se quieren detectar las minucias
349  * @param limite el lÃmite para alejarse de los bordes y descartar falsas minucias
350  */
351  public void detectarMinucias( HuellaDactilar imgEntrada , int limite ) {
352
353      int p;                // Pixel central
354      int[] Pi = new int[9]; // PÃxeles adyacentes
355      int cn;               // Crossing number
356      int aux = 0;
357
358      for( int x=limite ; x < imgEntrada.getWidth()-limite ; x++ ){
359          for( int y=limite ; y<imgEntrada.getHeight()-limite ; y++ ){
360
361              p = imgEntrada.getPixel( x , y );
362              aux = 0;
363
364              if( p == 0 ){
365
366                  Pi[0] = imgEntrada.getPixel( x+1 , y );
367                  Pi[1] = imgEntrada.getPixel( x+1 , y-1 );
368                  Pi[2] = imgEntrada.getPixel( x , y-1 );
369                  Pi[3] = imgEntrada.getPixel( x-1 , y-1 );
370                  Pi[4] = imgEntrada.getPixel( x-1 , y );
371                  Pi[5] = imgEntrada.getPixel( x-1 , y+1 );
372                  Pi[6] = imgEntrada.getPixel( x , y+1 );
373                  Pi[7] = imgEntrada.getPixel( x+1 , y+1 );
374                  Pi[8] = imgEntrada.getPixel( x+1 , y );
375
376                  for( int i = 0 ; i <= 7 ; i++ ){
377                      aux = aux + Math.abs(Pi[i] - Pi[i+1]);
378                  }
379
380                  cn = aux/2;
381
382                  if (cn == 1 || cn == 3){
383                      minucias.add(new Minucia(x, y, cn));
384                  }
385              }
386          }
387      }
388  }

```

■ **Cálculo de los ángulos de las minucias**



```

93 //
94 /*
95  * Metodo que calcula los angulos de las minucias
96  * @param imgEntrada la huella sobre la que se quieren detectar los angulos de las minucias
97  */
98 public void calcularAngulos( HuellaDactilar imgEntrada ) {
99
100     boolean fin = false;
101     int xInicial, yInicial;
102     int x, y;
103
104     ArrayList<Point> visitados = new ArrayList<Point>();
105     ArrayList<Point> vecinos = new ArrayList<Point>();
106     ArrayList<Point> aux = new ArrayList<Point>();
107
108     for( int i = 0 ; i < this.minucias.size() ; i++ ){
109
110         // Las coordenadas iniciales son las de la minucia
111         xInicial = minucias.get( i ).getX();
112         yInicial = minucias.get( i ).getY();
113
114         // Las coordenadas actuales, son las de la minucia
115         x = xInicial;
116         y = yInicial;
117
118         // Vaciamos la lista de visitados y añadimos la minucia a visitados
119         visitados.clear();
120         visitados.add( new Point ( x , y ) );
121
122         // Añadimos los vecinos de la minucia
123         obtenerVecinos( vecinos , visitados , vecinos , imgEntrada , x , y );
124
125         // Recorremos los vecinos y los vamos procesando
126         while( !vecinos.isEmpty() ){
127
128             x = vecinos.get(0).x;
129             y = vecinos.get(0).y;
130
131             vecinos.remove(0);
132

```

## CONCLUSIONES

- *Los sistemas de seguridad biométricos nos permiten la autenticación automática de las personas, evitando así los posibles fraudes o falsificación de identidad.*
- *Este sistema de comparación de huellas dactilares se observó la importancia de los registros biométricos para aumentar la seguridad, ya sea de una empresa o en el caso de los procesos policiales y judiciales.*
- *En este sistema se empleó un patrón similar: se recogen previamente las características de unos individuos, se cargan las imágenes al sistema y se procesan mediante un algoritmo numérico. Si en el momento en que la persona que trata de acceder concuerda las características, le permitirá el acceso. Si el sistema no es capaz de emparejar los datos, se denegará el acceso.*

## REFERENCIAS BIBLIOGRAFICAS

- [1] T. Y. Zhang, C.Y. Suen, “A Fast Parallel Algorithm for Thinning Digital Patterns”, *Communications of the ACM*, pp.236-239, Vol. 27, Num. 3, March 1984 .**
- [2] L. Lam, S-W Lee, C.Y. Suen, “Thinning Methodologies-A Comprehensive Survey”, *IEEE Trans. on Pattern Analysis and Machine Intelligence*, pp. 869-885, Vol. 14, No. 9, September 1992**
- [3] P.S.P. Wang, “An Improved Fast Parallel Thinning Algorithm for Digital Patterns”, *Proc. of International Conference on Computer Vision Pattern Recognition*, pp.364-367, 1985.**
- [4] C.M. Holt, A. Stewart, M. Clint, R.H. Perrot, “An Improved Fast Parallel Thinning Algorithm”, *Communications of the ACM*, pp.156-160, Vol. 30 Iss. 2, 1987**
- [5] D. Maio, D. Maltoni, A. Jain, S. Prabhakar, “Handbook of Fingerprint Recognition”, Ed. Springer-Verlag, 2003**