Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Identificação Biométrica Utilizando Impressão Digital

Igor de Carvalho Coelho

Monografia apresentada como requisito parcial
para conclusão do Curso de Computação — Licenciatura

Orientador
Prof. Dr. Pedro de Azevedo Berger

Brasília
2011

Universidade de Brasília — UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Curso de Computação — Licenciatura

Coordenador: Prof. Dr. Homero Luiz Piccolo

Banca examinadora composta por:

Prof. Dr. Pedro de Azevedo Berger (Orientador) — CIC/UnB
Prof. Dr. Flávio Vidal — CIC/UnB
Prof. Dr. Camilo Dorea — CIC/UnB

CIP — Catalogação Internacional na Publicação

Coelho, Igor de Carvalho.

Identificação Biométrica Utilizando Impressão Digital / Igor de Carvalho Coelho. Brasília : UnB, 2011.
60 p. : il. ; 29,5 cm.

Monografia (Graduação) — Universidade de Brasília, Brasília, 2011.

1. Biometria, 2. Impressão digital, 3. Processamento de Imagens

CDU 004.4

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro — Asa Norte
CEP 70910-900
Brasília–DF — Brasil

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Identificação Biométrica Utilizando Impressão Digital

Igor de Carvalho Coelho

Monografia apresentada como requisito parcial
para conclusão do Curso de Computação — Licenciatura

Prof. Dr. Pedro de Azevedo Berger (Orientador)
CIC/UnB

Prof. Dr. Flávio Vidal
CIC/UnB

Prof. Dr. Camilo Dorea
CIC/UnB

Prof. Dr. Homero Luiz Piccolo
Coordenador do Curso de Computação — Licenciatura

Brasília, 27 de Janeiro de 2011

Dedicatória

Dedico este trabalho a todos os que tem me ajudado nessa jornada. A minha família, namorada, professores e amigos que tem me ajudado em tudo.

Agradecimentos

Agradeço ao Professor Pedro Berger por ter me indicado os caminhos para a realização desse trabalho. Bem como o co-orientador Bruno Rolim que me ajudou nos pontos que não percebia. Agradeço também a Isabel que me ajudou na confecção do texto, sempre corrigindo meu péssimo português.

Resumo

Este trabalho visa a implementação de um sistema de certificação usando como meio a impressão digital do usuário. O objetivo principal é a implementação e teste de eficiência nas diversas fases do tratamento da imagem e da identificação.

Para isso o processo de identificação foi dividido em diversas etapas para o melhor entendimento. Em várias etapas diversos algoritmos diferentes foram testados a fim de escolher o melhor resultado. Masi adiante no texto, há os resultados de algums desses algoritmos tentados.

Palavras-chave: Biometria, Impressão digital, Processamento de Imagens

Abstract

This work implements an certification system using the user's fingerprint matching as the evaluation method. The main objective is just the implementation and benchmark of the various phases in the image enhance and identification.

For that, the identification process was divided into several steps for a better understanding. In various steps several algorithms were tested in order to choose the one with better results. Later in the text, there the results for some of the others algotrithm attempts.

Keywords: Biometrics, Fingerprint, Image Processing

Sumário

Lista de Figuras

Capítulo 1

Introdução

1.1 Motivação e Justificativa

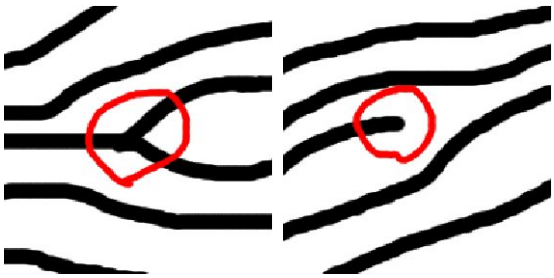
No mundo de hoje devemos estar sempre atentos a segurança. Seja de bens ou dados, sempre temos que proteger algo de modo que apenas o dono, ou quem ele autorizar, tenha acesso. E para identificar as pessoas que possuem essa autorização geralmente usamos senhas.

Porém, há diversos problemas com as senhas: em geral são relativamente fáceis de adivinhar e, quando são seguras, são difíceis de lembrar [20]. Além disso, tem-se o problema da Engenharia Social que, por meio de técnicas de interação com as pessoas, pode extrair informações e, consequentemente, adivinhar as senhas [17]. Por isso, é necessário a utilização de uma forma infalível de se identificar as pessoas, com esse objetivo usa-se a Biometria.

A Biometria consiste em usar as características únicas do indivíduo para identificá-lo, como por exemplo: retina, íris, forma da face, disposição das veias da mão e etc. Neste trabalho a forma de biometria focada são as impressões digitais. Formada por pequenos vales nos dedos, podem ser usados como se fossem carimbos para revelar um labirinto de linhas com uma disposição única para cada pessoa. Devido a sua unicidade e facilidade de extração, as impressões digitais têm sido usadas através dos anos como a principal forma de reconhecimento biométrico, muito usado na ciência forense, são uma forma barata, rápida e segura de se identificar uma pessoa. Outra vantagem é que as impressões digitais de uma pessoa permanecem inalteradas durante toda a sua vida [18]. Salvo, é claro, de eventuais ferimentos que danifiquem a impressão.

Existem diversas técnicas para comparar duas impressões digitais através de suas diversas características únicas definidas [6], porém poucas delas são viáveis computacionalmente. A técnica implementada nesse trabalho é a de identificação por meio das minúcias. As minúcias são pontos especiais formados pelas linhas que estão presentes em todas as impressões digitais. Para a identificação foram usados em especial dois tipos de pontos: as bifurcações e os términos de linha, como demonstrado na Figura 1.1. Apesar de não existirem estudos estatísticos suficientes para mostrar a unicidade das minúcias, elas são aceitas como uma forma bastante confiável de comparar impressões digitais.

Contudo, as imagens capturadas por meios digitais raramente tem uma boa qualidade, podendo conter ruídos e distorções que dificultam o processo automático de comparação. Por isso, para fazer um sistema de identificação, deve-se levar esse problema em considera-



(a) Bifurcação (b) Término de Linha

Figura 1.1: Tipos de minúcias Usados

ção. Assim, antes de buscar as minúcias na imagem devemos primeiro prepará-la através de processamentos na imagem. Nesse trabalho, a principal técnica usada será o Filtro de Gabor [7]. Com a utilização dessa técnica, passa-se a ter uma imagem com linhas bem claras disfarçando as imperfeições do meio de aquisição. E através dessas linhas, pode-se identificar as minúcias, formando uma matriz de pontos, e compará-las com um banco de outras matrizes para identificar quem é a pessoa.

Este trabalho está organizado da seguinte maneira: no capítulo sobre o referencial teórico serão explicadas as técnicas para o melhoramento da imagem (pré-processamento), afinamento das linhas, extração das minúcias, remoção das minúcias falsas (pós-processamento) e duas técnicas diferentes de *Matching* (comparação entre impressões digitais).

No capítulo sobre os resultados, serão mostradas as constantes obtidas experimentalmente para cada fase explicada no referencial teórico, bem como os resultados esperados para elas.

Por fim, o capítulo sobre a conclusão possui um apanhado sobre as técnicas implementadas e ideias para trabalhos futuros.

Assim, como hipótese tem-se que a utilização de apenas a técnica das minúcias como característica da impressão digital resulta em um identificador biométrico confiável.

1.2 Objetivos gerais e especificos

1. Objetivo Geral: Criar um sistema autônomo, com interface com *hardware* já pronta, que consegue captar a impressão digital, extrair suas características e compará-las às características previamente salvas.
2. Objetivo Específico: Por meio de pesquisa bibliográfica, adquirir o conhecimento de diversas técnicas para o aprimoramento da imagem e comparação da matriz de minúcias.
3. Objetivo Específico: Testar diversas técnicas para algumas das fases e, assim, determinar qual é a melhor.

1.3 Metodologia

A metodologia a ser aplicada nesse projeto possui conotação quase que completamente prática. Para cada estágio será feita:

1. Uma pesquisa bibliográfica sobre as diversas técnicas diferentes que poderiam ser aplicadas;
2. Implementação de uma ou várias dessas técnicas;
3. Teste objetivos e subjetivos com relação a essa técnica;
4. Incorporação da melhor técnica ao projeto final.

Ao final desse processo com todas as técnicas será realizado um teste do sistema todo. Nesse teste será usado um banco de dados de impressões digitas montado a partir de voluntários. Visto que, em tese, o que se está tentando identificar são apenas as impressões digitais e não o usuário em si, cada voluntário poderá colaborar com dez impressões digitais diferentes o que reduziria em dez vezes o tempo para se formar o banco de dados.

Ao término desse teste de velocidade e veracidade para o sistema inteiro, ele terá então seus dados comparados com os das revistas de biometria [15]. Caso esteja ao menos perto de um dos valores dessa tabela, o programa será considerado correto e a hipótese inicial será confirmada. Caso o programa esteja muito distante da tabela em [15], então será necessário voltar para os algoritmos usados e testar outros diferentes. Dessa forma espera-se selecionar os melhores algoritmos para o problema da identificação por impressão digital. A tabela com os valores de velociadade e falsos-positivos pode ser visualizada na Figura 1.2

Table 3. Average Performance Over all Four Databases in the Light Category (Top 15 only)

Algorithm	*Avg EER*	*Avg Match Time*	*Avg Model Size*
Suprema Inc	3.51%	0.22 sec	1.2Kb
Ji Hui*	3.69%	0.13 sec	0.2Kb
Beijing HanWang Technology Co.	3.96%	0.23 sec	1.6Kb
Bioscrypt	4.29%	0.17 sec	1.1Kb
Testech	4.33%	0.14 sec	1.2Kb
NITGEN	4.86%	0.19 sec	2.0Kb
Institute of Automation, The Chinese Academy of Sciences	4.91%	0.18 sec	1.2Kb
Deng Guoqiang*	5.26%	0.19 sec	1.4Kb
IDENCOM Germany	5.29%	0.18 sec	2.0Kb
Neurotechnologija	5.64%	0.14 sec	0.5Kb
Miaxis Biometrics Co.	6.21%	0.12 sec	2.0Kb
ActivCard	6.82%	0.16 sec	0.8Kb
DATAMICRO Co.	8.04%	0.10 sec	0.6Kb
Anonymous	9.05%	0.05 sec	0.2Kb
Changsha XingTong Technology Dev. Co.	9.79%	0.24 sec	0.9Kb

* Independent developer

Figura 1.2: Tabela com velocidades e erros esperados

Capítulo 2

Referencial teórico

As linhas em uma imagem de impressão digital são de vital importância visto que nelas estão contidas as informações para a biometria.

A extração das características da impressão digital é divida em quatro partes: Pré-processamento, Afinamento das linhas, Extração das minúcias e Pós-processamento. Depois de extraídas as características podemos usa um algoritmo de identificação (*Matching*) [18]. Dentre essas etapas o pós-processamento é feito para diminuir os falsos negativos.

O pré-processamento consiste no melhoramento da imagem removendo ruídos e acentuando as linhas da impressão digital. A parte de afinamento das linhas diminui a espessura dessas, deixando-as com apenas um pixel de espessura. A parte de extração de minúcias marca os pontos possíveis e aceita apenas os de bifurcação e fim de linha. A parte de pós-processamento remove as minúcias falsas que foram criadas pela imperfeição dos métodos usados no pré-processamento. A parte da identificação compara a matriz de minúcias extraída com uma pré-existente para identificar se esta impressão digital pode ser gerada pela mesma matriz. A Figura **??** mostra um diagrama para melhor ilustrar todo o processo.

2.0.1 Equipamento utilizado

O hardware utilizado para este trabalho é o *Digital Persona* da *Microsoft*. Ele produz imagens de tamanho 384x289 pixels com cada pixel tendo o tamanho de um byte.

Há um pré-processamento do próprio aparelho esticando a imagem conseguida. Dessa forma não é aconselhável fazer a aquisição das imagens com o aparelho na posição lateral, pois a imagem aparecerá distorcida. O aparelho pode ser visto na Figura 2.2

Porém o CD de instalação não possui nenhuma biblioteca de desenvolvimento. Por isso foi utilizado a coletânia de drivers para linux *fprint*. Nos sistemas operacionais derivados do *Debian* a instalação é bastante trivial, já que o pacote se encontra no repositório oficial. Basta digitar a linha:

```
$ sudo apt-get install libfprint0-dev
```

Os arquivos de cabeçalho estarão na pasta */usr/include* para consulta de quais funções são disponibilizadas para o uso.

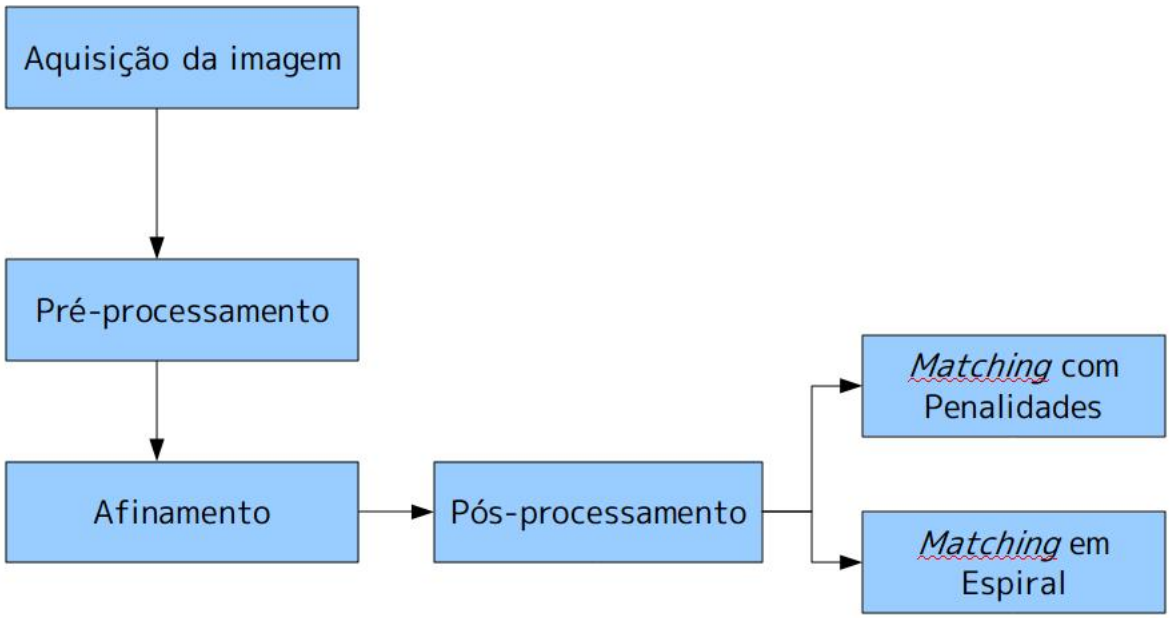


Figura 2.1: Diagrama de todo o processo

2.1 Pré-processamento

O pré-processamento consiste no melhoramento da imagem e preparação para a extração das minúcias. Para o melhoramento da imagem, o filtro utilizado é o Filtro de Gabor [13]. Este é um filtro direcional baseado em frequência que consegue uniformizar as linhas da impressão digital. Depois desse processo deve-se binarizar a imagem para a fase de afinamento das linhas para a espessura de um pixel, só então será possível extrair as minúcias.

O filtro de Gabor necessita de duas informações sobre a imagem em uma determinada janela: a direção e a frequência das linhas. Porém, realiza-se um tratamento inicial na imagem, através da segmentação e da normalização, melhorando o processo de filtragem [18].

A ordem em que as etapas é realizada foi retirada do artigo (18) e pode ser melhor visualizada no diagrama da Figura 2.3.

Figura 2.2: Digital Persona

2.1.1 Segmentação

O processo de segmentação no processamento de imagens consiste em separar áreas da imagem, no caso de impressões digitais, o fundo é separado da impressão em si criando a região de interesse contendo apenas as linhas da impressão digital. Em uma imagem de impressões digitais se está interessado apenas nas linhas da impressão. O fundo que aparece não é de interesse e pode ainda comprometer os processos futuros criando minúcias falsas na hora da extração.

Uma peculiaridade nas imagens de impressões digitais é que o fundo, geralmente, possui uma variância estatística ,conhecida como $sigma^2$, menor na escala de cinza. Com isso, pode-se dividir a imagem em blocos e calcular a variância dos valores de pixels desse, caso estejam abaixo de um limiar deve-se excluir o bloco da imagem. Para excluí-los basta tornar todo o bloco preto. Esse método é conhecido como *Variance Thresholding*[16]. A variância de um bloco k de WxW pixels pode ser obtida através da equação [9]:

$$V(k) = \frac{1}{W^2} \sum_{i=0}^{W-1} \sum_{j=0}^{W-1} (I(i,j) - M(k))^2 \tag{2.1}$$

Onde $M(k)$ é a média dos pixels do bloco k, $V(k)$ será a variância do mesmo bloco. E $I(i,j)$ é o valor do pixel com coordenadas (i,j). O limiar para esta etapa foi definido empiricamente com diversas imagens.

O resultado dessa etapa pode ser conferido na Figura 2.4

2.1.2 Normalização

Após a segmentação é realizado o processo de normalização, o qual consiste em padronizar os valores dos pixels na escala de cinza uniformizando a diferença entre os mais escuros e os mais claros. Para isso, uma técnica adequada é a equalização do histograma da imagem [3].

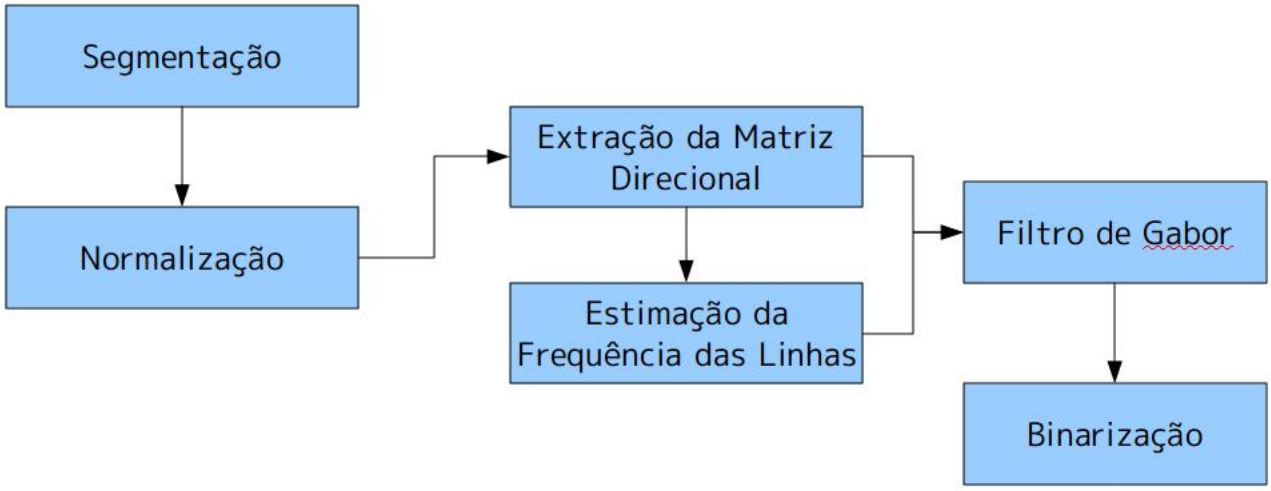
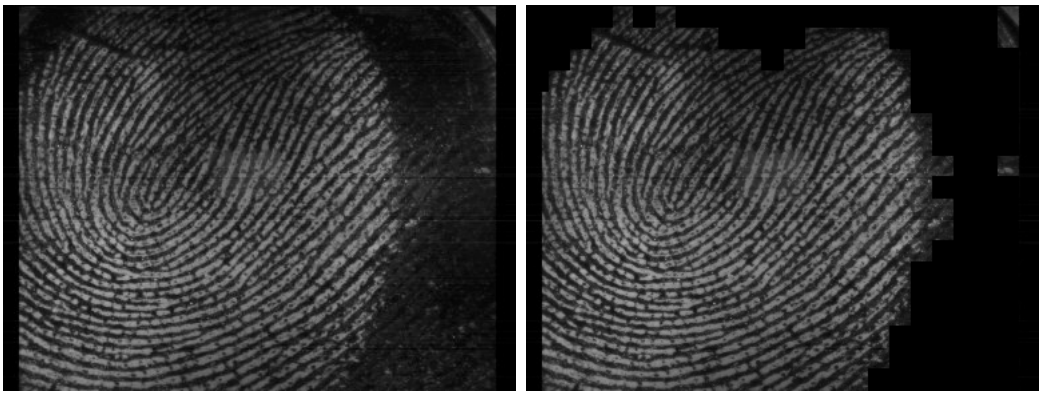


Figura 2.3: Diagrama do processo de pré-processamento

Um histograma, dentro do processamento de imagens, é uma representação gráfica da disposição dos valores dos pixels dentro da imagem, dando uma ideia da frequência desses. Para criar o histograma deve-se criar um vetor unidimensional com um tamanho igual ao número de valores que cada pixel pode atingir, no caso da imagem usada, cada pixel corresponde a um *byte*, assim os valores podem variar de 0 a 255. Todavia, deve-se lembrar de que a imagem já está segmentada, então caso seja feito o histograma da imagem inteira, esse ficará comprometido, pois a imagem ganha muitos pixels da mesma cor por ter separado a área de interesse do fundo da imagem tornando-a preta. Para contornar esse problema basta ignorar as janelas em que todos os valores são pretos (zero).

Para equalizar o histograma deve-se diluir as probabilidades de cada valor de pixel por todos os valores possíveis. Para tal, precisa-se primeiro normalizar o histograma. Considere uma imagem de NM pixels:

$$P_r(r_k) = \frac{n_k}{NM} \tag{2.2}$$



(a) Imagem Original (b) Região de interesse após a segmentação

Figura 2.4: Segmentação

Onde r_k é o k-ésimo nível de cinza e n_k o número de pixels que possuem esse nível na imagem. Assim os valores de $P_r(r_k)$ irão variar de 0 a 1 e representarão a probabilidade de um pixel aleatório na imagem ser do tom r_k. Como essa é uma probabilidade, fica evidente que

$$\sum_{k=0}^{k=r_{max}} P_r(r_k) = 1 \tag{2.3}$$

Assim o objetivo é espalhar os valores de probabilidade para que eles fiquem uniformes, melhorando o contraste da imagem. Para tal, deve-se alterar o tom dos pixels para que todos os níveis tenham a mesma probabilidade. Assim, o primeiro passo é obter uma função de distribuição acumulada.

Suponha uma transformação $s = T(r)$ em que a transformada $T(r)$ mapeia o pixel de nível r para o nível s e satisfazendo a duas propriedades, a saber:

1. $T(r)$ é monotonicamente crescente para garantir o ordenamento crescente do preto para o branco.
2. Se $0 \leq T(r) \leq 1$ então $0 \leq r \leq 1$ para garantir que o resultado esteja no mesmo intervalo de níveis de cinza.

Se for considerado o nível de cinza r como uma variável aleatória, é possível formar a função de distribuição de probabilidade de r. Com esse objetivo, primeiro considere $p_r(r)$ e $p_s(r)$ as funções de densidade de probabilidade antes e depois da modificação de $T(r)$, respectivamente. $p_r(r)$ e $p_s(r)$ estarão relacionados por:

$$p_s(s) = p_r(r)\frac{dr}{ds} = 1 \tag{2.4}$$

E assim, teremos a transformação $T(r)$, que é a função de probabilidade acumulada, lemos:

$$\int_0^s ds = \int_0^r p_r(r)dr \tag{2.5}$$

$$s = T(r) = \int_0^r p_r(r)dr \tag{2.6}$$

Contudo, como as imagens digitais trabalham no domínio discreto, pode-se considerar:

$$P_r(r_k) = \frac{n_k}{NM} \tag{2.7}$$

E define-se o "histograma acumulado da imagem"a Equação 2.8:

$$s_k = T(r_k) = \sum_{k=0}^{k=r_{max}} P_r(r_k) = \sum_{k=0}^{k=r_{max}} \frac{n_k}{NM} \tag{2.8}$$

E terá como inversa:

$$r_k = T^{-1}(s_k) \tag{2.9}$$

No apêndice B pode ser encontrado um código em linguagem C usado na equalização do histograma.

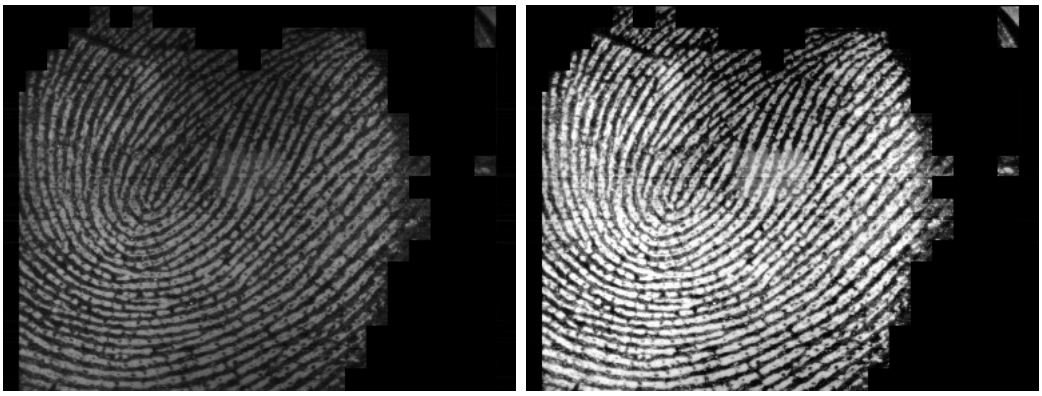
Um exemplo do resultado desta etapa pode ser visto na Figura 2.5.

2.1.3 Extração da Matriz Direcional

A matriz direcional consiste em uma representação das direções predominantes em cada segmento da imagem, como exemplo tem-se a Figura 2.6. Nela foi usada uma figura com linhas bem definidas. Existem diversos métodos para a extração dessa matriz, dentre eles há: as Grandezas Direcionais, o Operador Sobel, entre outros.

Grandezas Direcionais

Esse algorítimo baseia-se na diferença que existe quando se passa linhas em certos ângulos perto de uma borda. Por exemplo se for uma linha na vertical, uma linha traçada na horizontal terá a maior diferença, visto que os pixels estarão parte no interior do objeto e parte fora.



(a) Imagem Segmentada (b) Imagem Equalizada

Figura 2.5: Normalização

Por se tratar de um algorítimo mais empírico do que efetivamente matemático, ele é de mais fácil compreensão, porém não possui um resultado muito bom quando comparado, por exemplo, ao operador de Sobel que utiliza as derivadas parciais, o qual será discutido mais a frente.

Usando uma janela 9x9 mede-se as diferenças entre os pixels em determinados ângulos como mostra a Figura 2.7. Essa diferenças são calculadas de 8 formas diferentes, expressas pelas equações $S(0)$ a $S(7)$, em que cada uma corresponde a uma linha na Figura 2.7, essas estão definidas na Equação **??**. Depois de medidas essas diferenças são selecionados o maior e o menor elemento. Assim dependendo do resultado da Equação 2.11 é escolhido o menor, definido pelo índice p, ou o maior, definido pelo índice q.

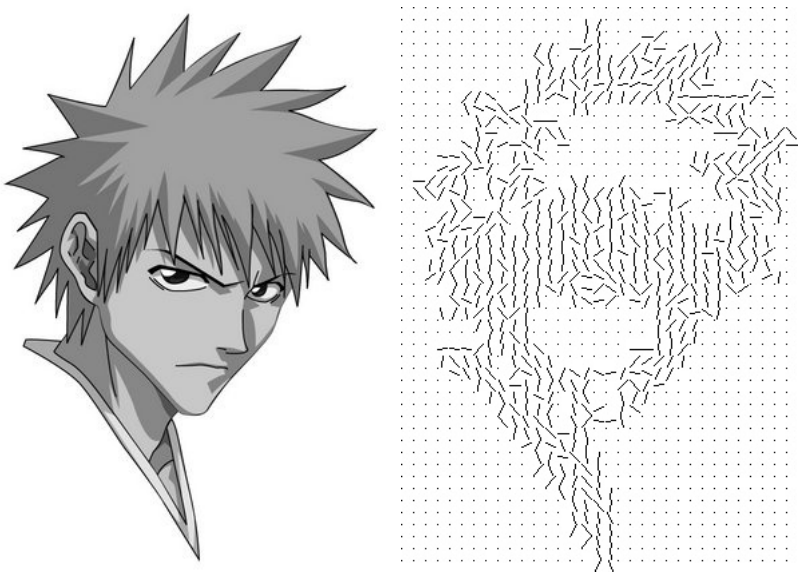
As formulas para cada S são:

$$\begin{aligned}
S(0) &= \textstyle\sum_{k=-2}^{2} I(i, j+2k) - I(i,j), \\
S(1) &= \textstyle\sum_{k=-2}^{2} I(i+k, j-2k) - I(i,j), \\
S(2) &= \textstyle\sum_{k=-2}^{2} I(i+2k, j-2k) - I(i,j), \\
S(3) &= \textstyle\sum_{k=-2}^{2} I(i+2k, j-k) - I(i,j), \\
S(4) &= \textstyle\sum_{k=-2}^{2} I(i+2k, j) - I(i,j), \\
S(5) &= \textstyle\sum_{k=-2}^{2} I(i+2k, j+k) - I(i,j), \\
S(6) &= \textstyle\sum_{k=-2}^{2} I(i+2k, j+2k) - I(i,j), \\
S(7) &= \textstyle\sum_{k=-2}^{2} I(i+k, j+2k) - I(i,j).
\end{aligned} \tag{2.10}$$

O teste a ser feito é:

$$d = \begin{cases} p & \text{se } (4C + S_p + S_q) < \frac{3}{8} \sum_{i=0}^{7} S(i) \\ q & \text{caso contrário} \end{cases} \tag{2.11}$$

Com S_p sendo o $\min S_i | i = 0, 1, 2..7$ e S_q sendo o $\max S_i | i = 0, 1, 2..7$ e C uma constante definida experimentalmente. Então basta selecionar o ângulo correspondente a $S(d)$, onde d é o índice escolhido pela Equação 2.11.



(a) Figura Original (b) Matriz Direcional

Figura 2.6: Matriz Direcional, usando Sobel

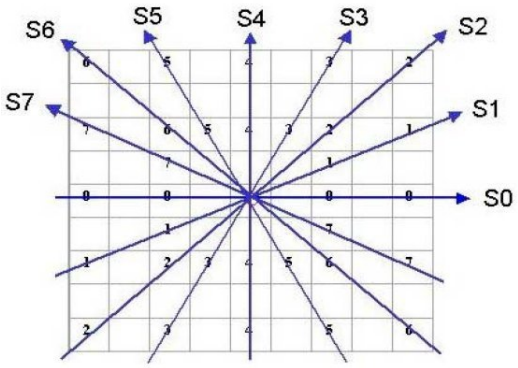


Figura 2.7: Direções para cada dado S[i]

Por se tratar basicamente de uma quantidade constante de operações feitas em cada grupo de pixel, pode-se dizer que o crescimento assintótico também será $O(n^2)$, para uma imagem quadrada de lado n. Contudo vale lembrar que nas imagens direcionais não há necessidade de percorrer toda a imagem pixel a pixel, como normalmente é feito na convolução. Por isso pode-se dizer que na prática essas formas de extração de imagens direcionais tem um crescimento de $O\left(\frac{n^2}{B^2}\right)$ em que B é o tamanho da janela utilizada.

Operador Sobel

O Operador Sobel é composto por duas matrizes que devem usadas na operação de convolução sobre a imagem produzindo as derivadas em X e Y que, se unidas, criam uma imagem com as bordas em destaque [3]. Sobre a operação de convolução vide Apêndice A. Com esses valores de derivadas, é possível descobrir para que direção uma linha está tendendo.

As matrizes para a convolução do Operador Sobel são:

$$G_x \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} G_y \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Onde G_x é o gradiente no eixo X e G_y, por sua vez, é o gradiente no eixo Y. Para gerar a imagem com as bordas destacadas basta usar a expressão $G^2 = \sqrt{G_x^2 + G_y^2}$ em que G é a imagem resultante. Porém para extrair a matriz direcional não há a necessidade dessa imagem. A expressão a seguir serve para extrair o ângulo da linha ao mesmo tempo em que dilui seus valores de modo a ter um campo mais consistente [14];

$$\theta = \frac{90}{\pi} \arctan\left(\frac{\sum_{j=0}^{n}\sum_{i=0}^{m} 2G_x(i,j)G_y(i,j)}{\sum_{j=0}^{n}\sum_{j=0}^{m} |G_x(i.j)^2 - G_y(i,j)^2|}\right) \tag{2.12}$$

O ângulo predominante para as linhas é dado por θ. Deve-se repetir esse processo para cada bloco de tamanho mxn da imagem.

Como já visto anteriormente, esse método é apenas uma convolução acrescida de um número de operações constante para diluir os ângulos, assim pela convolução o custo assintótico deste método será $O(n^2)$.

Filtro Gaussiano

Um detalhe interessante é que as linhas de uma impressão digital, em geral, não mudam repentinamente de direção, mantendo uma curva mais suave. Graças a isso, é possível usar um filtro atenuador na matriz direcional para eliminar eventuais mudanças bruscas de direção causadas por ruídos. Para esse fim foi escolhido o filtro Gaussiano.

O filtro Gaussiano para duas dimensões é definido por:

$$g(i,j) = \frac{1}{2\pi\sigma^2} \exp^{-\frac{i^2+j^2}{2\sigma^2}} \tag{2.13}$$

Com x e y sendo as coordenadas, esta expressão pode montar uma matriz para a convolução colocando um limite para x e y.

Para utilizar o filtro, primeiro deve-se converter a matriz direcional em um vetor contínuo e separando as componentes x e y usando as expressões:

$$\begin{aligned} \Phi_x &= \cos(2\theta(i,j)), \\ \Phi_y &= \sin(2\theta(i,j)), \end{aligned} \tag{2.14}$$

Onde Φ_x e Φ_y são as componentes em x e y para o filtro Gaussiano. Depois a atenuação é feita da seguinte forma:

$$\begin{aligned}\Phi'_x(i,j) &= \sum_{u=-\frac{\omega}{2}}^{\frac{\omega}{2}} \sum_{v=-\frac{\omega}{2}}^{\frac{\omega}{2}} G(u,v)\Phi_x(i-u\omega, j-v\omega),\\ \Phi'_y(i,j) &= \sum_{u=-\frac{\omega}{2}}^{\frac{\omega}{2}} \sum_{v=-\frac{\omega}{2}}^{\frac{\omega}{2}} G(u,v)\Phi_y(i-u\omega, j-v\omega),\end{aligned} \tag{2.15}$$

Com $G(u,v)$ sendo a posição (u,v) no filtro Gaussiano de tamanho ωxω. Depois usa-se esses componentes para calcular a nova direção para o pixel (i,j):

$$\Theta(i,j) = \frac{1}{2}\arctan\frac{\Phi'_y(i,j)}{\Phi'_x(i,j)}. \tag{2.16}$$

Assim a nova direção do pixel (i,j) será $\Theta(i,j)$.

2.1.4 Frequência das Linhas

Tendo a matriz direcional já pronta, é possível agora verificar a frequência das linhas para passá-la como o outro parâmetro para o Filtro de Gabor. A frequência das linhas representa o inverso da distância entre as linhas da impressão digital capturada em um dado bloco mxn[18].

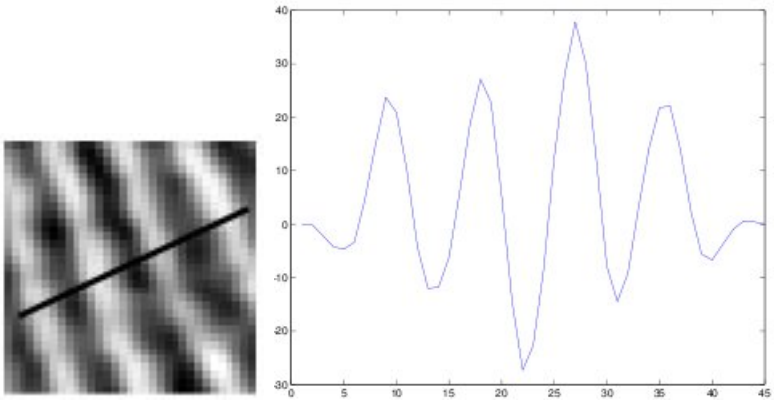
Para calcular a frequência das linhas deve-se primeiro criar um gráfico da energia por espaço, energia esta representada pelo valor do pixel na escala de cinza, em uma direção ortogonal às linhas da imagem. Na seção passada foi calculada a matriz direcional, logo basta somar 90º à direção recebida para obter a ortogonal e então, salvar o valor dos pixels no caminho em um vetor. Como a representação gráfica desse vetor segue de forma de uma senoidal, basta calcular a distância média entre os picos, sendo que um pico pode ser representado por um incremento em relação aos valores anteriores do vetor com um decremento em relação aos valores posteriores. Para uma melhor visualização do processo vide a Figura 2.8

2.1.5 Filtro de Gabor

Tendo a Matriz Direcional e uma matriz com a frequência de linhas de cada bloco, agora pode-se passar na imagem o Filtro de Gabor.

O filtro de Gabor, desenvolvido por Dennis Gabor em 1946[4], no contexto de imagens de impressões digitais, consegue melhorar as senoidais aumentando o contraste entre os picos e vales. Depois de acertadas as constantes, ele também permite uma melhor caracterização da direção e da espessura das linhas, visto que foi modelado de forma similar à visão humana. O filtro de Gabor é empregado em impressões digitais por ser um seletor de frequência e de orientação. Essas características tem feito com que ele venha sendo usado em diversos outros aplicativos que precisem de uma acentuação de bordas [18].

A equação do filtro de Gabor possui uma componente real e uma imaginária. Para o processamento no domínio espacial (direto na figura) considera-se apenas a parte real do filtro que é definida como um cosseno modulado por uma Gaussiana. [4] A equação de



(a) Linha na direção ortogonal (b) Gráfico da energia

Figura 2.8: Frequência de linhas, nesse caso são observados 4 picos[18]

Gabor no domínio real está definida em 2.17.

$$G(x, y, \theta, f) = \left\{ -\frac{1}{2} \left[\frac{x_\theta^2}{\sigma_x^2} + \frac{y_\theta^2}{\sigma_y^2} \right] \right\} \cos 2\pi f x_\theta, \tag{2.17}$$

Onde x_θ e y_θ são definidos por:

$$\begin{aligned} x_\theta &= x \cos\theta + y \sin\theta \\ y_\theta &= -x \sin\theta + y \cos\theta \end{aligned} \tag{2.18}$$

A variável θ representa a orientação da linha no pixel (x, y) e f a frequência no mesmo pixel. σ_x e σ_y são os desvios padrões nos eixos x e y, respectivamente e, podem ser obtidos também de forma empírica. Esses servem para controlar a largura de banda do filtro denotando o que retirar. Porém, o ideal é que esses parâmetros variem de acordo com a frequência porque do contrario poderia-se produzir áreas com aprimoramento desigual da imagem [18]. Com isso, para esse trabalho os parâmetros σ_x e σ_y serão funções expressas na Equação 2.19.

$$\begin{aligned} \sigma_x &= k_x F(i, j) \\ \sigma_y &= k_y F(i, j) \end{aligned} \tag{2.19}$$

Onde $F(i, j)$ denota a frequência no pixel (i, j), e k_x e k_y, constantes. Logo, o valor dos parâmetros σ_x e σ_y serão dependentes da frequência.

O filtro de Gabor é aplicado através da convolução utilizando a frequência e a orientação do pixel central. O pixel (i, j) será representado pela Equação 2.20.

$$E(i,j) = \sum_{u=-\frac{w_x}{2}}^{\frac{w_x}{2}} \sum_{v=-\frac{w_y}{2}}^{\frac{w_y}{2}} G(u,v,O(i,j),F(i,j))N(i-u,j-v) \tag{2.20}$$

Em que $E(i,j)$ será o novo pixel e $O(i,j)$, $F(i,j)$ e $N(i,j)$ serão: a orientação, a frequência e o valor no pixel (i,j), respectivamente. w_x e w_y representam o tamanho da janela para a convolução, que normalmente é fixo. Porém, para acomodar melhor o filtro é ideal que a janela de convolução também tenha um valor variável com relação aos limitadores de largura de banda. Essa relação pode ser definida na Equação 2.21 [18].

$$\begin{aligned} w_x &= 6\sigma_x \\ w_y &= 6\sigma_y \end{aligned} \tag{2.21}$$

A Figura 2.9 mostra o resultado da aplicação do filtro de Gabor.

(a) Imagem original

(b) Depois do filtro de Gabor, sem binarização

Figura 2.9: Filtro de Gabor

2.1.6 Binarização

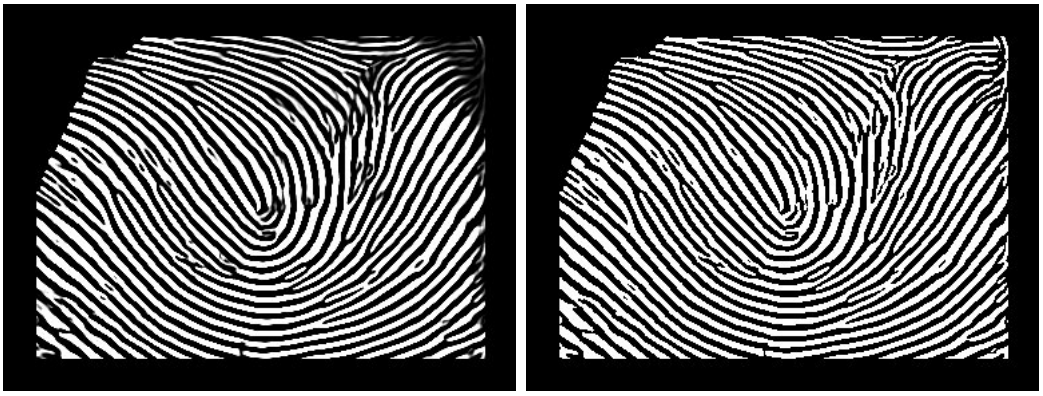
O processo de binarização é relativamente simples. Depois de estabelecido um limiar, todo pixel maior que ele passa a ser igual ao valor máximo possível e todo pixel com um valor menor passa a ter o valor mínimo. Abaixo há uma representação matemática para valores de 0 a 255 e com um limiar L, em que neste trabalho foi definido experimentalmente:

$$I(i,j) = \begin{cases} 255 & \text{se } I(i,j) > L \\ 0 & \text{se } I(i,j) \leq L \end{cases} \tag{2.22}$$

Assim, haverá uma imagem com fronteiras exatamente definidas entre os vales e picos, possibilitando a etapa de afinamento.

Na Figura 2.10 há uma binarização de uma imagem melhorada com filtro de Gabor. Percebe-se que as partes borradas somem em uma imagem binarizada.

Figura 2.10: Imagem binarizada



(a) Imagem melhorada com filtro de Gabor (b) Imagem anterior binarizada

2.2 Afinamento

O processo de afinamento é uma operação do processamento de imagens que consiste em deixar as linhas de uma imagem, já binarizada, com a espessura de apenas um pixel, mas mantendo as características das curvas e linhas. Após esse afinamento pode-se aplicar outros algoritmos para a retirada de características, no caso de impressões digitais, as minúcias. Foram usados nesse trabalho os algoritmos de afinamento de Zhang-Suen e Holt [5] [8].

2.2.1 Algorítmo de Zhang-Suen

Esse algoritmo é uma otimização do famoso algoritmo de afinamento que ficou conhecido como Zhang-Suen por seus autores [21]. Ele é essencialmente composto de dois passos.

Seja $N(p)$ o número de vizinhos da vizinhança de oito pixels de p que não são nulos, $S(p)$ é o número de transições de preto para branco (de valor nulo para não nulo) nos vizinhos de p no sentido horário e $p1-p8$ sejam os oito-vizinhos de p colocados da seguinte maneira:

$$\begin{bmatrix} p1 & p2 & p3 \\ p8 & p & p4 \\ p7 & p6 & p5 \end{bmatrix}.$$

No primeiro passo do algoritmo o pixel p é removido (tem seu valor nulificado) quando ele atende as seguintes condições:

1. $2 \leq N(p) \leq 6$
2. $S(p) = 1$
3. $p2.p4.p6 = 0$
4. $p4.p6.p8 = 0$

A primeira condição verifica se existem ao menos dois vizinhos de p, mas não mais do que seis, assim sempre serão removidos pixels que estejam na borda e não na parte interna de uma linha. A segunda condição verifica se esse pixel faz parte de apenas uma linha ou mais de uma, como por exemplo em uma bifurcação ou um cruzamento, assim remove-se o pixel apenas se esse estiver dentro de apenas uma linha. A terceira condição verifica se os vizinhos $p2$,$p4$ e $p6$ são fundo da imagem, verificando se possuem o valor zero. A quarta condição é igual a terceira, mas para os pixels $p4$,$p6$ e $p8$.

Para a segunda fase, o pixel p é eliminado se ele atender as seguintes condições:

1. $2 \leq N(p) \leq 6$
2. $S(p) = 1$
3. $p2.p4.p8 = 0$
4. $p2.p6.p8 = 0$

Se o pixel satisfazer todas as condições ele deverá ser eliminado.

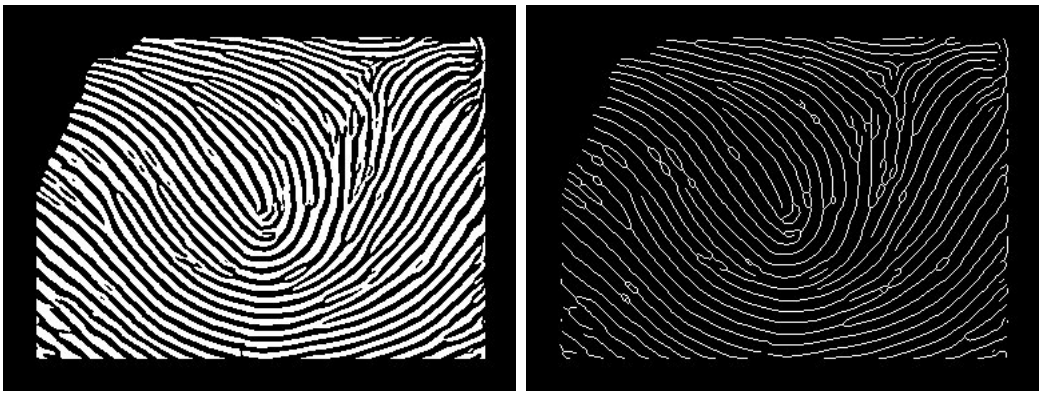
Porém, há um detalhe a ser lembrado: os dois passos não são executados paralelamente. Deve-se inicialmente fazer o primeiro passo em toda a imagem removendo os pixels que satisfazerem as condições para só então fazer o segundo passo por toda a imagem.

2.2.2 Algorítmo de Holt

Holt e outros autores fizeram uma melhoria no algoritmo de Zhang-Suen, o qual nos dá linhas mais finas e corretas removendo o serrilhamento que acontece em linhas diagonais [10]. Holt observou que metade dos pontos de uma linha no formato de escada pode ser removido sem afetar a conectividade dos objetos. Depois de passar o algoritmo de Zhang-Suen, Holt propôs o uso de quatro máscaras para retirar o serrilhamento. As máscaras estão demonstradas a seguir. O pixel central é eliminado se um dos valores do vizinho marcado com um X é nulo e todo o resto obedece a máscara.

$$\begin{bmatrix} 0 & 1 & X \\ 1 & 1 & X \\ X & X & 0 \end{bmatrix} \begin{bmatrix} X & 1 & 0 \\ X & 1 & 1 \\ 0 & X & X \end{bmatrix} \begin{bmatrix} 0 & X & X \\ X & 1 & 1 \\ X & 1 & 0 \end{bmatrix} \begin{bmatrix} X & X & 0 \\ 1 & 1 & X \\ 0 & 1 & X \end{bmatrix}$$

O resultado do processo de afinamento pode ser observado na Figura 2.11.



(a) Imagem binarizada (b) Imagem com as linhas afinadas

Figura 2.11: Imagem Afinada

2.3 Extração de Minúcias

Agora que a imagem já está melhorada e suas linhas estão de uma forma na qual é possível a análise, pode-se começar a extração das características. O que no caso de impressões digitais, são as minúcias.

Como explicado anteriormente, existem diversos tipos de minúcias, mas duas delas são fundamentais e dão origem as outras minúcias, elas são a bifurcação e o término de linha e podem ser observadas na Figura **??**. Todavia agora com a imagem afinada essas minúcias podem ser observadas como as matrizes abaixo, no qual o valor zero representa o nulo e o valor um representa o valor ativo:

$$\text{Bifurcação} \begin{bmatrix} 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \text{e} \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix} \text{Final de linha} \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

Poderia-se apenas procurar por essas matrizes dentro da imagem, porém esses pontos podem estar rotacionados e, então, haveria a necessidade de procurar por três matrizes adicionais para cada um dos tipos de minúcias, o que nos obrigaria a comparar oito matrizes a cada pixel.

Uma solução melhor é utilizar o algoritmo do *Crossing Numbers* [1]. Os *Crossing Numbers* podem ser expressos na Equação 2.23.

$$CN = \sum_{i=1}^{8} |p_i + 1 - p_i| \tag{2.23}$$

Onde p_i é o pixel de vizinhança, ver sessão 2.2. Assim o *Crossing Number*-CN será a contagem de quantas transições de preto para branco e de branco para preto há em volta

do pixel p.

Dessa forma fica simples definir as minúcias.

$$\text{O pixel p será do tipo} \begin{cases} \text{Bifurcação} & \text{se } CN = 6 \\ \text{Fim de linha} & \text{se } CN = 2 \\ \text{Pixel normal} & \text{Caso contrario} \end{cases}$$

Note que essa configuração desconsidera um tipo de minúcia de cruzamento que teria $CN = 8$, para simplificar o processo de pós-processamento que será visto na Sessão 2.4.

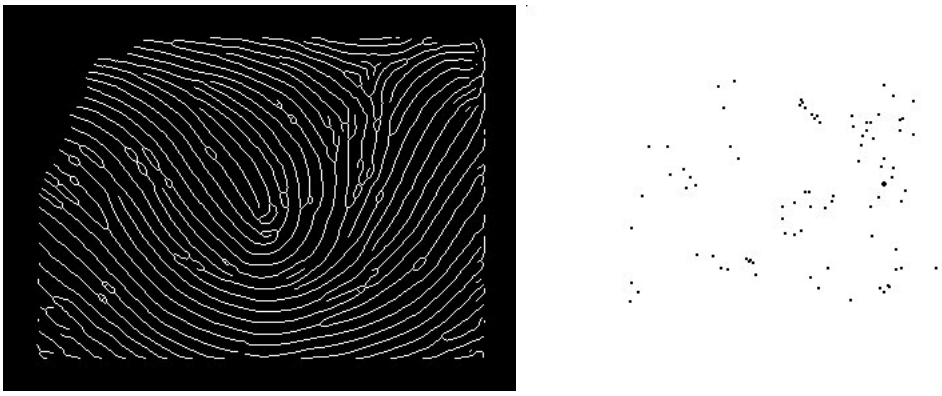
Para o gerenciamento das minúcias é ideal criar uma lista encadeada de modo a simplificar os algoritmos de reconhecimento. Para cada pixel identificado como minúcia é necessário guardar: suas coordenadas, o tipo de minúcia e a direção da linha naquele ponto da imagem. Quanto menos informações sobre a minúcia, que são utilizadas no processo de reconhecimento, menores as chances de dar um falso-positivo, porém quanto mais informações são consideradas sobre as minúcias, maiores as chances de dar falsos-negativos. Quando se está tratando de segurança é preferível ter falsos-negativos do que um falso-positivo.

É importante lembrar também que as linhas da impressão digital não se mantem até o final da imagem e, assim, podem ser confundidas com minúcias. A Figura 2.12 mostra alguns pontos que produziriam minúcias falsas de final de linha. Por isso é bom focar-se apenas nas minúcias centrais da imagem. Para amenizar este problema pode-se usar o algoritmo descrito na Sessão 2.4.1.

Figura 2.12: Minúcias falsas decorrentes do fim da impressão

Depois de ter a lista com as posições das minúcias, a imagem já não é mais necessária para o processo de identificação. Porém por algumas minúcias serem falsas, deve-se manter a imagem para o pós-processamento.

Na Figura 2.13 há na imagem (b) uma imagem gerada pela lista de minúcias conseguida pela imagem (a).



(a) Imagem melhorada (b) minúcias extraidas

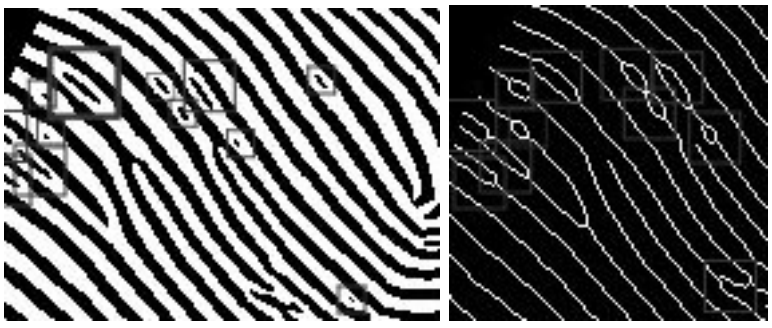
Figura 2.13: Extração e minúcias

É possível notar a grande quantidade de pontos. Para o processo de identificação essa quantidade de pontos implicariam em muito tempo gasto para cada comparação. Contudo vários desses pontos não são realmente minúcias, são imperfeições geradas nas linhas devido aos processos de melhoramento e afinamento. Na sessão 2.4 será apresentado um algoritmo para ignorar as minúcias provocadas pelas imperfeições.

2.4 Pós-processamento

O pós-processamento consiste em remover as minúcias falsas decorrentes do processo de melhora da imagem.

O filtro de Gabor apesar de demonstrar uma boa melhora na qualidade da imagem, produz pequenas manchas que, depois do processo de afinamento, parecem uma bifurcação com uma junção, criando uma volta. Esse erro produz duas minúcias falsas. A Figura 2.14 mostra essas divergências e como eles afetam o afinamento.



(a) Manchas causadas pelo filtro de Gabor (b) Erros fabricados pelo afinamento

Figura 2.14: Manchas do filtro de Gabor

Fora as divergências, também são formadas pequenas linhas isoladas que produzem duas minúcias que não condizem com a impressão digital original, como pode ser observado na Figura 2.15.

Figura 2.15: Pequenas ilhas na imagem

Um terceiro problema causado pelo pré-processamento é a junção de linhas que originalmente não possuíam intersecções. Esse problema também gera duas minúcias do tipo bifurcação. Essa configuração pode ser observada na Figura 2.16.

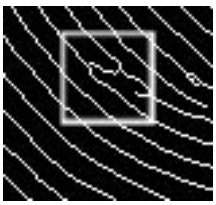


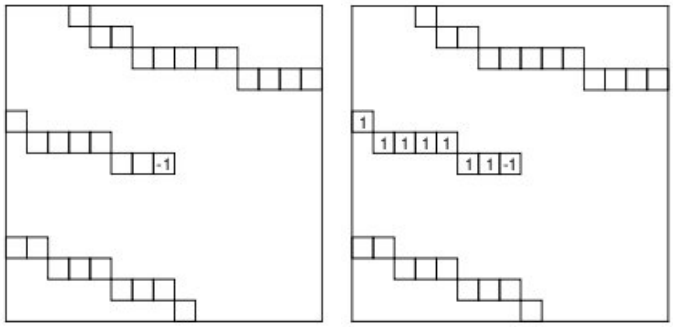
Figura 2.16: Junção de linhas

A solução para identificar e eliminar ambas minúcias minúcias falsas é acompanhar as linhas marcando-as [19]. Primeiro escolha uma área de atuação quadrada ao redor da minúcia a ser testada, (geralmente é um quadrado de 16x16 pixels) mas pode-se escolher um maior ou um menor. Mas deve-se ter em mente que um quadrado grande demais pode acabar eliminando minúcias verdadeiras enquanto um pequeno demais pode não eliminar as minúcias falsas. Tendo a área de atuação definida, deve-se copiar essa área para uma imagem auxiliar para alterar seus pixels sem modificar a imagem original. Deve-se considerar que pontos não marcados na linha terão o valor zero, ou o do próprio pixel no caso da imagem binarizada trabalhada que é de 255. Agora deve-se marcar o ponto onde está a minúcia com o valor -1, porém como -1 é 255 no tipo *unsigned char*, é melhor colocar o valor -2 (como a imagem está binarizada os únicos valores são 0 e 255 e um valor como -2 não prejudicará o algoritmo). Agora deve-se percorrer cada linha, e suas bifurcações, da imagem ligada a minúcia marcando seus pixels, para cada tipo de minúcia o algoritmo procede de maneira diferente.

Para testar as minúcias de final de linha, deve-se analisar vizinhança de 8 do pixel onde se encontra a minúcia identificando a linha em que termina na minúcia:

1. Siga a linha marcando seus pixels com 1 até que a linha acabe ou chegue no final da imagem copiada;
2. Agora percorra as bordas da imagem copiada e conte o numero de transições do pixel de valor 0 para o valor 1;
3. Se esse valor for igual a 1 então essa minúcia é verdadeira e não precisa ser deletada, agora se o valor for 0 isso significa que ela era uma minúcia do tipo ilha como citado anteriormente.

Um exemplo de execução desse algoritmo pode ser observado na Figura 2.17.



(a) Passo inicial (b) Percorrimento da linha

Figura 2.17: Exemplo do algoritmo para a minúcia de fim de linha. Imagem de [18]

Agora para testar se a minúcia do tipo bifurcação é verdadeira o processo é um pouco mais complicado.

1. Primeiro percorre-se os 8-vizinhos da minúcia e marque os pixels de linha que estejam imediatamente ligados a minúcia com os valores 1, 2 e 3;
2. Agora a partir do pixel marcado com 1, percorre-se sua linha marcando todos os pixels com o mesmo valor. Caso haja outra bifurcação deve-se seguir para os dois lados marcando os dois caminhos com o mesmo valor. Isso pode ser programado facilmente usando a técnica da recurção, ao chamar a mesma função de marcação sempre que achar uma bifurcação;
3. A marcação das linhas deve parar quando chegar a margem do bloco da imagem, quando chegar a um pixel já marcado ou o fim da linha;
4. Agora repete-se o processo com os outros pixels marcados com os valores 2 e 3;
5. Depois de percorrer todas as linhas ligadas a minúcia, percorra a borda da imagem no sentido horário e conte o número de transições de 0 para 1, de 0 para 2 e de 0 para 3. Se o número de transições for igual a 1 para cada um desses valores a minúcia é considerada verdadeira e pode ser mantida. Caso o valor de qualquer um deles seja 0 então isso significa que ouve um "bolsão". Caso o valor de algum deles seja maior que 1 então houve uma junção de linhas.

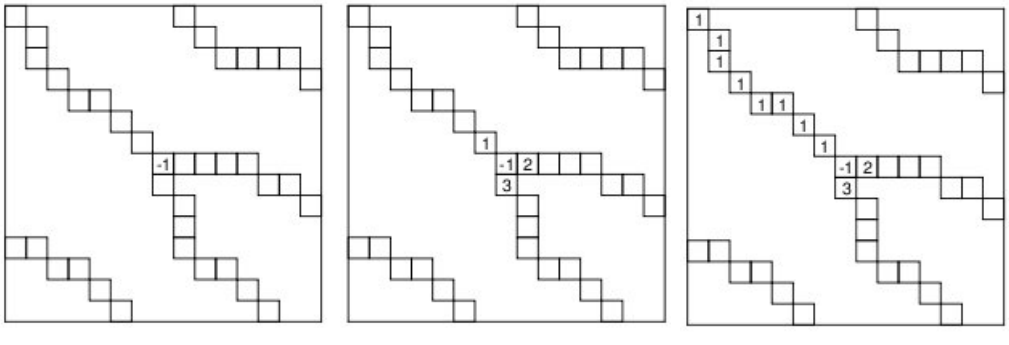
Conta-se apenas número de transições e não o número de pixels com o valor determinado pois perto da borda a linha pode seguir de forma paralela a borda criando uma margem de pixels do mesmo valor.

Um exemplo de execução desse algoritmo pode ser observado na Figura 2.18.

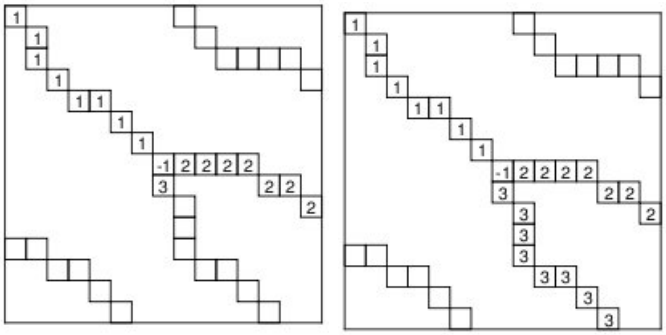
Na Figura 2.19 tem-se as situações que configuram minúcias falsas nesse algoritmo.

2.4.1 Eliminando as Minúcias das bordas

Em vários casos a impressão digital não ocupa a imagem inteira, como quando colocamos o dedo indicador ou mesmo mínimo. Assim aparecem algumas minúcias na fronteira



(a) Passo inicial (b) Marcação inicial (c) 1º percorrimento



(d) 2º percorrimento (e) 3º percorrimento

Figura 2.18: Exemplo do algoritmo para a minúcia de bifurcação. Imagem de [18]

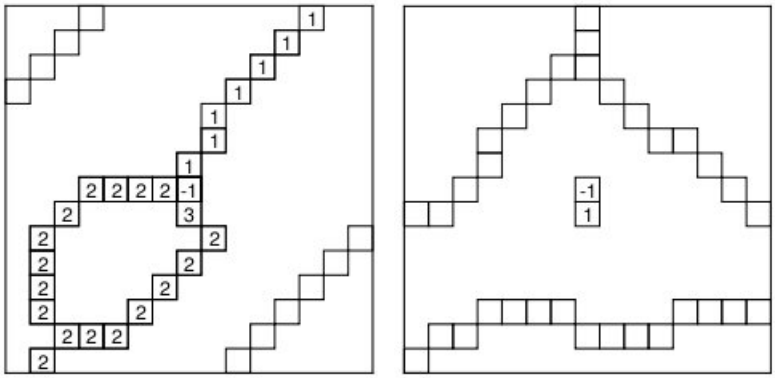
da impressão digital com o fundo. E o algoritmo de pós-processamento já apresentado não as reconhecerá como falsas.

Como temos alguns dados sobre a minúcia, podemos usá-los junto com a imagem para determinar se ela está dentro da área da minúcia ou se está na borda. Uma forma simples de resolver este problema é usar a posição da minúcia na imagem e o ângulo da linha associada à ela. Assim basta seguir esta direção até ou achar outra linha, nesse caso a minúcia é dita como verdadeira, ou achar a borda da imagem, e nesse caso a minúcia é tida como falsa. A Figura 2.20 exemplifica a diferença entre usar ou não esta estratégia. Note a quantidade de minúcias falsas retiradas.

Outra forma simples é apenas restringir a procura por minúcias a porção central da imagem. Porém essa forma pode jogar fora muitas minúcias verdadeiras e também apenas ameniza. Portanto o ideal é usar as duas formas em conjunto.

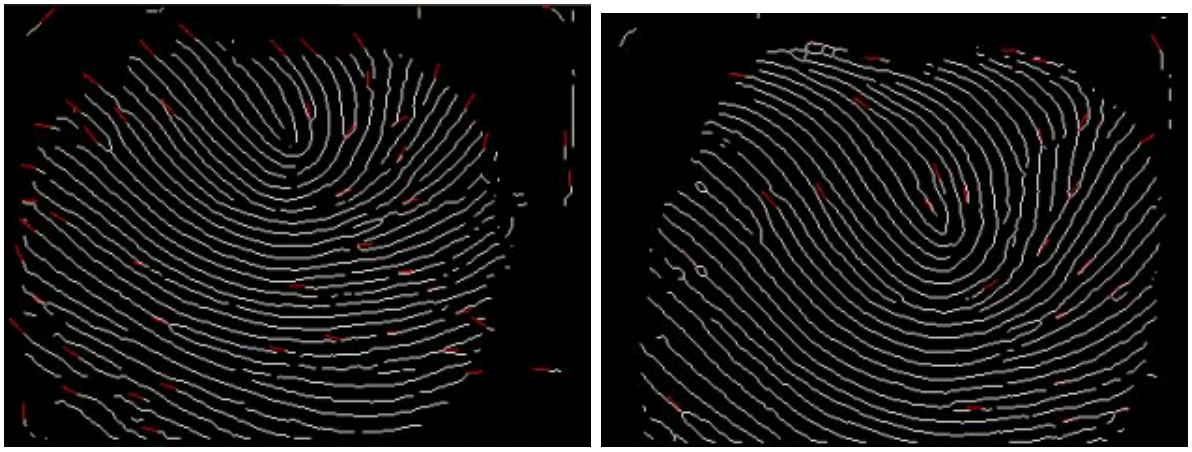
2.5 *Matching*

A etapa *matching*, ou casamento, consiste em um sistema automático que, quando dado como entrada duas matrizes de minúcias, ele responde afirmativo para o caso das duas matrizes terem uma grande probabilidade de pertencerem a mesma pessoa ou negativo caso essa probabilidade seja muito baixa e inconclusiva. Um sistema de reconhecimento ideal deve dar sempre a mesma resposta mesmo com a rotação, translação ou deformação do dedo. Os motivos para uma eventual rotação ou translação da imagem são



(a) Bolsão (b) Ilha

Figura 2.19: Identificação de minúcias falsas. Imagem de [18]



(a) Sem a remoção (b) Com a remoção

Figura 2.20: Remoção das minúcias da Fronteira

óbvios pois o usuário não conseguirá colocar o dedo sempre no mesmo lugar exato, mas agora a deformação se dá por causa da natureza elástica da pele, o que pode fazer com que algumas minúcias fiquem mais distantes ou próximas. Além disso é necessário lembrar que em se tratando de medições do corpo humano, a medida está sujeita a mudanças pelo tempo ou a um eventual ferimento, que podem fazer com que se perca a elasticidade do dedo ou uma parte da impressão digital, mudando-a por completo. No caso do ferimento, dependendo da gravidade, não há muito o que ser feito visto que uma solução determinística como um programa não pode adivinhar minúcias que não estão mais lá. Por isso no caso de ferimentos nos dedos que modifiquem a impressão digital o melhor a fazer é recadastrar o dedo ou cadastrar um novo dedo, caso este não possua mais uma impressão bem definida.

O sistema de reconhecimento pode usar as linhas e sobrepor uma impressão sobre a outra ou apenas pontos específicos característicos extraídos da imagem, como as minúcias. Na prática, a identificação pode ser feita usando a sobreposição das linhas uma sobre a outra, porém esse método se tornaria muito custoso para uma implementação computacional, pois as impressões sofrem muita deformação em virtude do meio de aquisição.

Além do mais, com essa técnica seria difícil determinar quando foi atingida a sobreposição máxima tendo em vista que mesmo que fossem alinhadas as linhas de uma região fechada da imagem, uma região oposta provavelmente estaria desalinhada, mesmo sendo a mesma impressão, por causa principalmente da deformidade causada pela elasticidade da pele. Assim para um sistema computacional o mais indicado é realmente um sistema pontual que pode ser avaliado facilmente usando relações matemáticas de similaridade.

Na prática, o processo de reconhecimento usa alguma regra para determinar o nível de similaridade entre duas matrizes. A tática mais comum é medir a distância entre uma minúcia e sua correspondente na outra matriz, porém essa abordagem não considera eventuais rotações ou mesmo deformações. Outra forma é sobrepor uma matriz na outra e ir fazendo modificações na matriz testada até se igualar ao *template* previamente adquirido, cada tipo de mudança possui um peso. Ao final do processo, somam-se as penalidades e, caso elas estejam abaixo de um limiar pré-definido, as duas matrizes são consideradas iguais. Esse processo é diferente do de apenas somar as distâncias pois leva em consideração uma pequena rotação, translação e até deformidade da impressão. Dentre as modificações possíveis tem-se para se aplicar as penalidades:

1. Rotação/Translação de todas as minúcias;
2. Translação de uma única minúcia, que pode ocorrer devido a deformação;
3. Inclusão/Remoção de minúcias.

A Figura 2.21 ilustra como duas imagens da mesma impressão digital podem ser bem diferentes e como o sistema de reconhecimento automático precisa ser robusto para ser usado nessas aplicações. Nota-se na Figura 2.21 uma clara translação da impressão digital o que provoca o desaparecimento de algumas minúcias, sendo assim é visto outro empasse: saber qual é a melhor imagem para salvar de modo a ter um bom e confiável *template* para o reconhecimento.

(a) Primeira captura (b) Segunda captura

Figura 2.21: Mesmo dedo capturado duas vezes

Uma técnica que pode ser utilizada para determinar o *template* é determiná-lo de forma iterativa [13]. O processo está descrito a seguir:

1. Salve a primeira matriz de minúcias que conseguir, essa será nosso primeiro *template*;
2. Sempre que a identificação der positiva conte o número de minúcias da matriz testada;
3. Se esse número for maior, então translade a imagem para que o centro da imagem tenha a maior quantidade de minúcias possível;
4. Agora salve essa nova matriz como o novo *template*;
5. Sempre que for identificar uma impressão digital repita o processo.

A vantagem de se usar um sistema iterativo é que este dispensa treinamento, porém no começo a matriz de minúcias ainda não é a melhor. Logo existe uma chance maior, ainda que pequena, de conseguir um falso-positivo e assim esse *template* passar a pertencer a pessoa errada. Para prevenir isso normalmente se pede para que o usuário coloque o dedo no sensor cerca de cinco vezes na hora do cadastramento.

A tática adotada nesse trabalho na formatação das matrizes para salvar no *template* ou para prepará-las para o reconhecimento consiste de:

1. Definir uma minúcia central baseada na matriz direcional;
2. Alinhar a matriz de minúcias com base nessa minúcia central. Com isso tem-se o problema de translação amenizado;
3. Se não houver *template* para esse usuário, salvar a matriz de minúcias como ela está agora e terminar;
4. Caso exista um *template* para se comparar, pode-se seguir.

Uma das técnicas utilizadas para a comparação é o algoritmo adaptativo elástico de comparação (*adaptive elastic string matching algorithm* [12] em [13]). Esse algoritmo requer duas características de cada minúcia depois de alinhadas, a saber: a distância da minúcia escolhida até minúcia central e a direção da linha associada a minúcia que fora estimada na matriz direcional. O primeiro passo é representar as minúcias como uma *string* impondo um alinhamento linear baseado na distância da minúcia central. Depois é possível compará-las usando um algoritmo inexato para quantificar a correspondência explicado a seguir.

O algoritmo inexato de comparação tenta transformar uma das *strings* na outra, no nosso caso transformar a *string* de entrada no *template*. O número de operações usadas para essa transformação é tido como a taxa de similaridade, quanto menor esse valor mais igual uma matriz é da outra. Como já dito as operações de retirada de uma minúcia verdadeira, adição de uma minúcia falsa e modificação das coordenadas possuem uma penalidade associada a cada, para assim ir definindo a similaridade entre elas. Dentre todas as operações o algoritmo escolhe a que tiver menor custo a cada passo, baseado em programação dinâmica. Ao fim deve-se somar essas penalidades e compará-las com um limiar. Para cada minúcia na entrada e sua candidata no *template* o algoritmo verifica se esta se encontra dentro de um limite de deformação, usando as características da distância da minúcia central e posição, caso ela esteja fora desse limite, haverá uma penalidade para essa minúcia para que seja removida ou inserida no *template*. Porém caso ela esteja dentro

desse limite a penalidade é proporcional a mudança das características da minúcia de entrada para que se igualem a do *template*. A Figura 2.22 ilustra a ideia desse algorítimo.

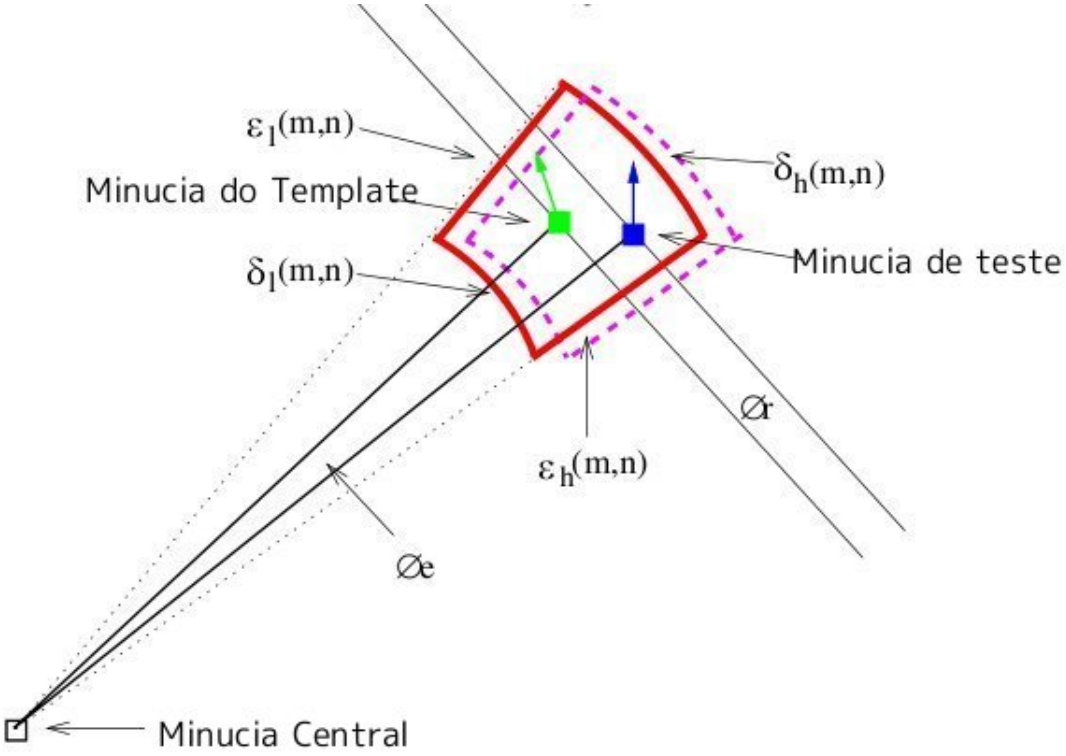


Figura 2.22: Características da minúcia para o *matching*. Imagem de [12]

2.5.1 Detecção da Minúcia Central

Um problema apresentado nessa estratégia de *matching* é a determinação de uma minúcia central para ser usada como base. Dois pontos podem ser utilizados como base para o *matching*, são o núcleo da impressão digital ou o ponto delta, *core point* e *delta point*, respectivamente. Esses pontos são reconhecidos como os pontos de maior curvatura na impressão digital. A Figura 2.23 exemplifica esses dois pontos.

Existem diversas técnicas para detectar esses pontos, mas a mais comum é a do Índice de Poincaré ou *Poincaré Index* [2]. Essa técnica utiliza uma série de multiplicações e somas para determinar se um ponto tem possibilidade de ser o ponto de maior curvatura e ainda decide se esse ponto tem mais chance de ser um *delta* ou um *core*.

Primeiramente deve-se normalizar a Matriz Direcional de modo a que os ângulos tenham apenas 8 valores possíveis: 0^o, 22.5^o, 45^o, 67.5^o, 90^o, 112.5^o, 135^o e 157.5^o, como pode ser visto na Figura 2.24.

Feito isso deve-se pegar um vetor de pontos dentro de uma curva fechada. Essa curva pode ser um circulo ou qualquer outra forma, sempre que seja fechada para depois ser percorrida no sentido anti-horário. Supondo que esse vetor tenha N elementos, o índice de *Poincaré* será dado por:

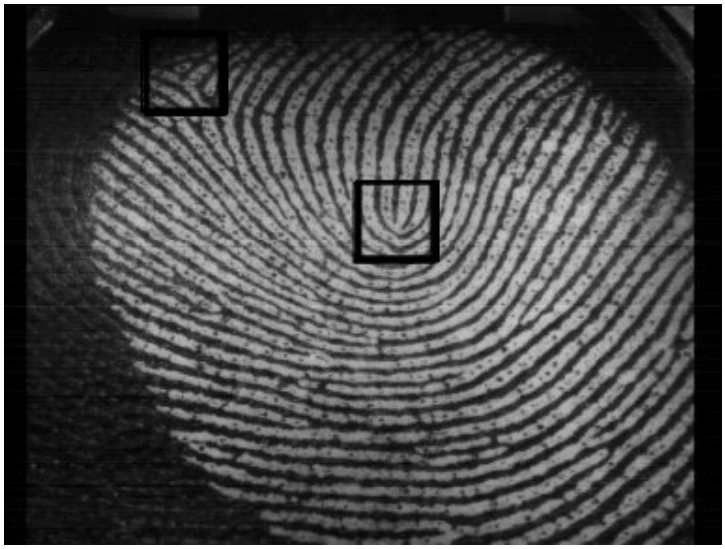


Figura 2.23: Pontos *core*(no centro) e *delta*(em cima)

$$Poincare(x,y) = \frac{1}{2\pi}\sum_{k=0}^{N-1}\Delta(k) \tag{2.24}$$

Onde

$$\Delta k = \begin{cases} \delta(k) & \text{se} \quad |\delta(k)| < \frac{\pi}{2} \\ \delta(k)+\pi & \text{se} \quad \delta(k) < -\frac{\pi}{2} \\ \pi - \delta(k) & \text{se} \quad \delta(k) > \frac{\pi}{2} \end{cases} \tag{2.25}$$

Em que $\delta(k) = \Theta(x_{(k+1)modN}, y_{(k+1)modN}) - \Theta(x_k, y_k)$.

Assim se o índice de *Poincaré* no bloco for de 0.5, esse bloco pode ser um *core point*. Se o valor do índice for de -0.5, então o bloco pode ser um *delta point*. Para qualquer outro valor, apenas consideramos que o bloco não contenha nenhum dos dois pontos procurados.

Essa abordagem, porém possui um problema: o tamanho da curva escolhida. Se ela for pequena demais, pode não conseguir detectar o ponto de curvatura, e se ela for grande demais,ela pode conter diversos pontos de curvatura e se apontar como um só, afinal existem pessoas que possuem mais de um ponto de núcleo ou delta.

O que nos leva a um outro problema, nem todas as pessoas possuem esses pontos nas digitais, então este método não abrange toda a população. Mas podemos considerar que o próprio método de identificação por digitais não abrange toda a população, pois existem pessoas que não possuem digitais.

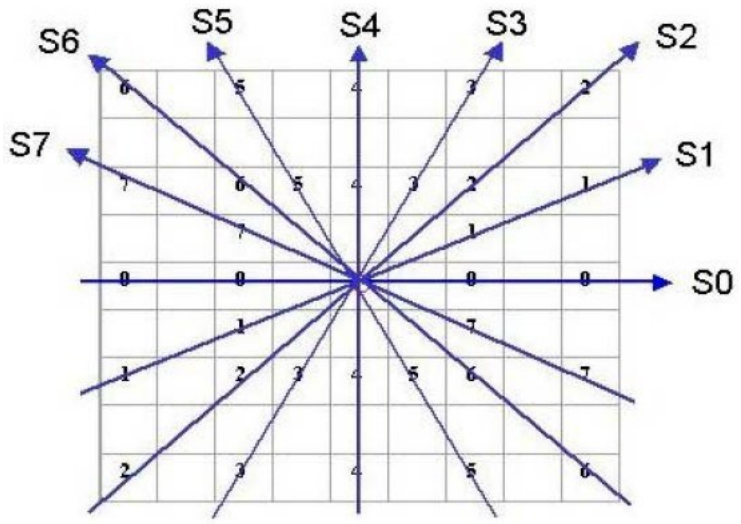


Figura 2.24: Direções para a "normalização"da Matriz Direcional

2.5.2 Técnica alternativa: *Matching* em espiral

Outra técnica de *matching* utilizada foi a do *Matching em Espiral*. Essa é uma técnica utilizada pela fabricante *Hitachi* em seus leitores de impressão digital. Possui um conceito simples: girar as minúcias em um padrão espiral e ver se elas ficam suficientemente próximas umas das outras.

O algoritmo decorre da seguinte maneira:

1. Primeiro elimina-se as minúcias que estejam muito próximas, o quão próximas pode ser decidido empiricamente;
2. Depois sobrepõe-se as minúcias e verifica se uma quantidade boa delas estão suficientemente próximas;
3. Caso não estejam, translada-se as minúcias de teste seguindo a regra da Equação 2.26:

$$\begin{cases} x = x + (-1)^i * i \\ y = y + (-1)^i * i \end{cases} \tag{2.26}$$

Em que i varia até um raio definido. Na imagem 2.25 temos uma demonstração gráfica para facilitar o entendimento.

Pode-se incluir ainda uma variação de angulo de modo a que mesmo se o usuário coloque o dedo de cabeça para baixo ele seja identificado. Como esse processo é relativamente rápido basta fazer a espiral toda, executar uma rotação e refazer a espiral toda novamente, até completar uma volta com a rotação.

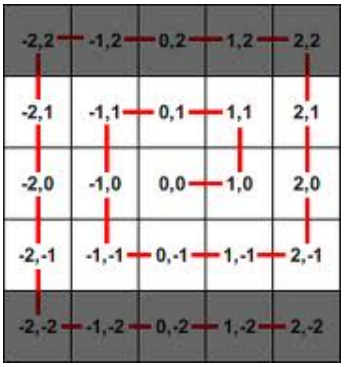


Figura 2.25: Transformação na imagem de modo espiral

Capítulo 3

Resultados

Nessa seção serão mostrados os resultados obtidos para cada parte da do programa. Serão mostrados os valores selecionados para as constantes empíricas bem como o por que da escolha entre um algoritmo em detrimento do outro no caso das partes que foram implementadas mais de um algoritmo diferente.

3.1 Pré-processamento

3.1.1 Segmentação

O componente a ser estimado na segmentação é o limiar para a variância do bloco. Caso a variância seja inferior a esse limiar então o bloco é considerado fundo e caso contrário será uma região de impressão digital.

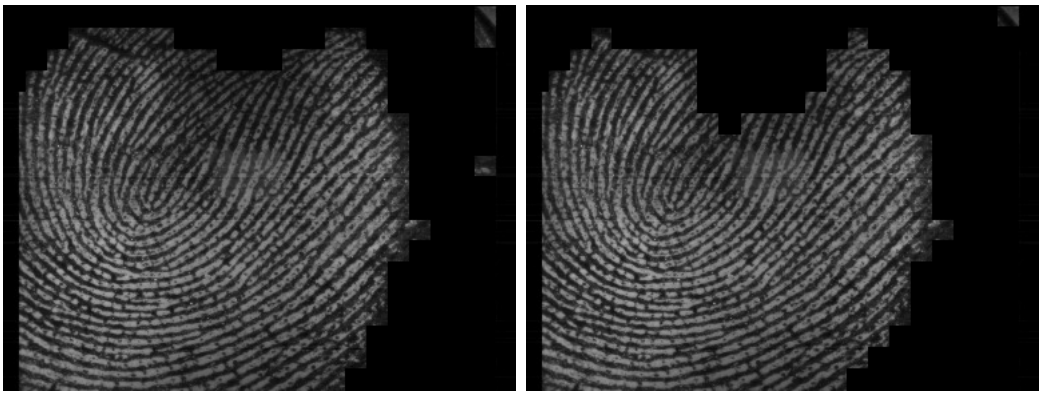
Como pode ser visto na Figura 3.1 não há uma forma matemática para definir se o limite escolhido está separando bem a impressão do fundo. Por isso, para determinar o melhor limiar foi feito um programa a parte apenas com a função de segmentação que mudava o limite de forma interativa. Assim, o usuário pode aumentar e diminuir o limite até que uma porção considerável do fundo tenha sido retirado e anotar o valor atual do limite. Depois é feita a segmentação de outras imagens usando esse limite, dessa forma pode-se verificar se o limite adquirido serve como um bom limite para a maioria, ou totalidade, das imagens ou se apenas para aquele caso específico foi bom.

A Figura 3.1 também nos mostra o resultado para alguns valores de limiar. Após testar com 20 imagens o valor escolhido para essa fase do pré-processamento foi o de 1.0. Ao comparar as imagens nota-se que esse valor separa bem o fundo, mas ainda sobra um pouco de ruido do fundo, já o valor de 2.0 retira todo o fundo, mas em contra partida retira também uma porção importante da impressão digital que é o centro dela.

Esse valor é alterado dependendo do tamanho da janela utilizada para a segmentação. Caso a janela mudasse para 9x9, por exemplo, todo esse processo deverá ser refeito.

3.1.2 Normalização

A técnica da equalização de histograma não precisa estimar nenhuma constante, como um limiar, por exemplo. Por isso apenas foi necessário aplicar o algorítimo.



(a) Limite de 1.0 (b) Limite de 2.0

(c) Limite de 3.0

Figura 3.1: Resultados da Segmentação para Diversos valores de limite

A diferença foi que como a segmentação separava o fundo da imagem colocando a janela inteira com o valor zero, o valor zero seria um dos mais, senão o mais, frequentes na imagem. Assim os blocos representado o fundo da imagem não poderiam ser considerados. Para tal bastou-se verificar se, em cada janela, a maioria dos pixels era de valor zero, caso afirmativo essa janela era ignorada.

A Figura 3.2 mostra o quão acentuada é a diferença entre considerar ou não as janelas já separadas pela segmentação. Nota-se como a imagem usando-a inteira no algoritmo parece ter menos contraste.

3.1.3 Extração da Matriz Direcional

Dentre as duas técnicas consideradas, Sobel e Grandezas Direcionais a que se saiu melhor para imagens do tipo da impressão digital foi a técnica do operador de Sobel. Os testes foram realizados nas mesmas 20 imagens citadas nos resultados da etapa de segmentação.

O teste levou em consideração os quesitos: complexidade algorítmica e uma análise subjetiva dos resultados. Como todas as duas técnicas são basicamente convoluções, com algumas operações constantes, todas elas tem complexidade algorítmica de $O(n^2)$ então foi usado quase que exclusivamente a análise subjetiva.

(a) Ignorando o fundo (b) Utilizando a imagem inteira

Figura 3.2: Resultados da Segmentação para Diversos valores de limite

Na Figura 3.3 pode-se ver as matrizes direcionais de uma imagem de impressão digital para cada técnica adotada. Nota-se que a matriz utilizando o operador de Sobel, apesar de não seguir todas as linhas com exatidão, ao menos segue a direção predominante de uma dada área, o que não acontece nas outras técnicas que em determinadas partes da figura ficam em direções completamente erradas.

Depois de decidida qual técnica melhor se aplicava ao tipo de imagem que seria utilizada, o programa foi refeito acrescentando o final estatístico que dilui as direções na imagem [14]. A Figura 3.4 ilustra as direções obtidas diretamente sobre a imagem de entrada. As linhas em branco representam as direções obtidas. Percebe-se como as linhas seguem com exatidão maior os contornos da imagem. As linhas paralelas representam o angulo zero, pois essas regiões são consideradas fundo da imagem e por isso serão ignoradas no resto do programa.

3.1.4 Frequência das linhas

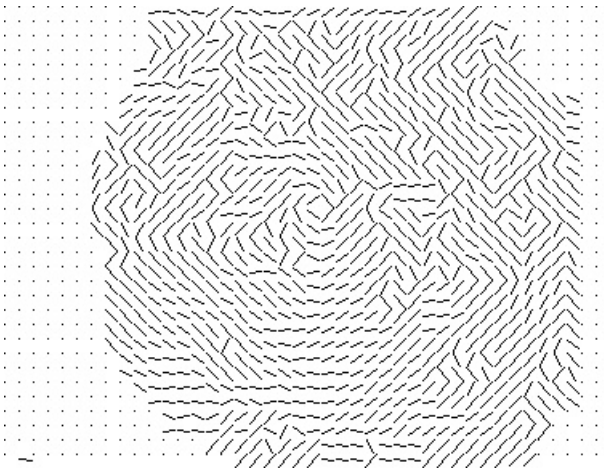
Assim como o processo de equalização de histograma, a estimação da frequência de linhas de uma janela não possuem parâmetros a serem estimados. Por isso foi necessária apenas a aplicação da técnica prontamente descrita.

Apenas dois detalhes foram alterados para o programa. O primeiro é ignorar o fundo da imagem, e o segundo é aplicar uma técnica de substituição para evitar que a frequência mude de forma muito drástica de um valor para o outro. Essa técnica sim possui um componente estimável.

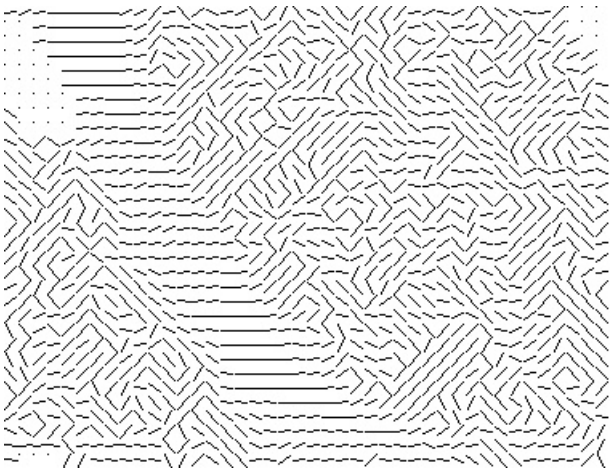
Essa técnica consiste no seguinte algoritmo:

1. Testar se a frequência encontrada está com um valor inferior a um ϵ estimado;
2. Caso esse valor enteja abaixo do ϵ estimado, teste se o valor de frequência da janela a esquerda possui um valor maior que ϵ;
3. Caso afirmativo, substitui-se o valor de frequência atual por este;
4. Caso contrário, faça o mesmo teste com a janela acima;

(a) Imagem Original



(b) Utilizando a técnica de Sobel



(c) Utilizando a técnica das Grandezas Direcionais

Figura 3.3: Resultado das diversas técnicas de extração da matriz direcional

5. Se nem a janela a esquerda e nem a janela acima tiver um valor acima de ϵ, deixe o valor como está

Utilizando essa técnica pode-se diluir a frequência das linhas visto que ela não varia de forma brusca na impressão digital.

3.1.5 Filtro de Gabor

Essa técnica por sua vez possui três componentes estimáveis. São eles: σ, γ e φ. A alteração em cada uma dessas constantes implica em um determinado efeito na imagem, a saber:

1. σ: Altera a espessura das linhas, se elas forem grossas demais podem se juntar com as linhas adjacentes e se forem muito finas elas desaparecem ou perdem ligações que existem na imagem original.
2. γ: Regula o alongamento das linhas, se for muito pequeno as linhas podem ficar muito curtas e perder ligações e se o valor for muito grande pode provocar minúcias falsas em decorrência de borrões.

(a) Imagem mesclada com a matriz direcional

Figura 3.4: Resultado da aplicação de Sobel junto do final estatístico [14]

3. φ: Por se tratar de uma função periódica, pois possui cosseno na fórmula, essa constante regula sua fase. Em termos prático ela escurece a imagem como um todo, para depois clarear até certo ponto novamente, de forma cíclica. Como se uma camada de tinta passasse pela imagem de forma vertical de um lado para o outro.

Nas Figuras 3.5, 3.6 e 3.7 tem-se algumas diferenças na imagem com a variação de cada parâmetro. Apesar de ter pouca diferença entre as imagens, essas poucas diferenças podem criar minúcias falsas e, com isso, reduzir o número de minúcias verdadeiras depois do estágio de pós-processamento. Por isso deve-se prestar bastante atenção e fazer um bom ajuste dessas constantes.

Para este projeto as constantes ficaram acertadas para:

$$\sigma = 4.5$$
$$\gamma = 1.0$$
$$\varphi = 0.0$$

3.1.6 Binarização

Para a constante de binarização foi escolhido o valor do meio do raio de valores possíveis para escala de cinza. Foi pensado em alterar esse valor, mas com o valor de 128 nenhuma das linhas foi cortada na maioria das 20 imagens testadas, o que provocaria mais minúcias falsas, então esse valor foi mantido.

Como a parte de binarização foi incluída já dentro da função de filtro de Gabor, os resultados da binarização podem ser vistos nas Figuras 3.5, 3.6 e 3.7 .

(a) Valor de 3.0 (b) Valor de 4.0 (c) Valor de 5.0

Figura 3.5: Consequência na imagem da alteração do parâmetro σ

(a) Valor de 1.0 (b) Valor de 2.0 (c) Valor de 3.0

Figura 3.6: Consequência na imagem da alteração do parâmetro γ

3.2 Afinamento

Para analisar o resultado, foi necessário aproximar a imagem melhorada com o filtro de Gabor e seguir cada uma das linhas para se certificar de que nenhuma delas fora cortada. Essa análise foi necessária pois se o algoritmo de afinamento estivesse cortando as linhas da imagem já melhorada teria que trocar por outro algorítimo de afinamento. Por ser uma análise muito custosa, ela foi feita apenas em 5 imagens, como nenhuma delas apresentou quebra de linhas então o algoritmo foi considerado bom.

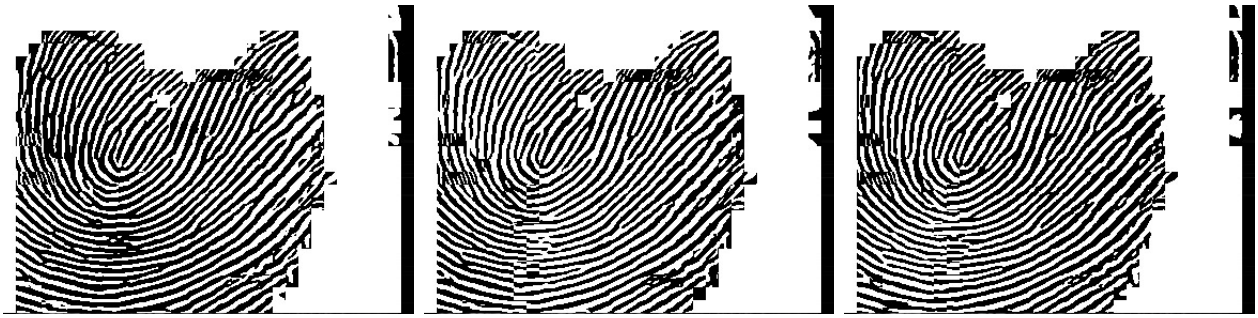
Como já mostrado na seção de Referencial Teórico, o resultado do processo de afinamento não varia muito, então o resultado pode ser conferido na Figura 2.11.

3.3 Extração das minúcias

Para essa parte do programa é esperado que se encontre toda e qualquer candidata a minúcia. Isso deverá dar por volta de quarenta a sessenta minúcias dentro da imagem.

Como pode ser notado pela Figura 2.13, o algoritmo está achando todas bifurcações ou finais de linha na imagem. Para testar a exatidão desse resultado foi necessário sobrepor as imagens e ver se os pontos coincidiram com as bifurcações e finais de linha.

Como esse é um algoritmo bastante automático e não precisa de nenhum ajuste subjetivo, sua análise termina aí. Porém para diminuir o problema de encontrar minúcias nas bordas da impressão digital, foi restringida a área de busca por minúcias para que



(a) Valor de 0.0 (b) Valor de 2.0 (c) Valor de 4.0

Figura 3.7: Consequência na imagem da alteração do parâmetro φ

se procure apenas no centro. Nesse trabalho foi desconsiderado 20% em cada borda da imagem como representado na Figura 3.8.

Figura 3.8: Restrições para evitar algumas minúcias falsas

3.4 Pós-Processamento

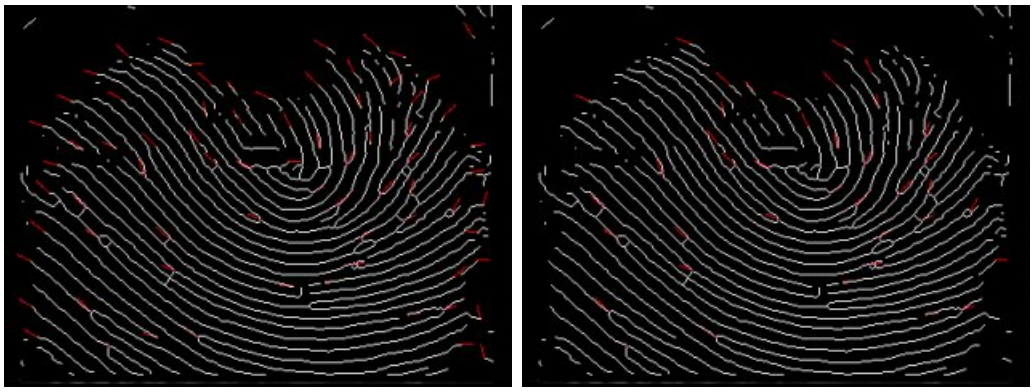
Para essa parte do programa foram testadas diversos tamanhos para as janelas. Isso se deve ao fato de que uma janela maior elimina mais minúcias falsas, mas acaba eliminando muitas verdadeiras também, em contrapartida uma janela muito pequena deixaria passar muitas minúcias falsas o que inviabilizaria a identificação pois essas minúcias falsas possuem uma tendência aleatória dependendo da posição como se escane-a o dedo.

O desejado para a identificação é uma quantidade entre dez e vinte minúcias [1].

A Figura 3.9 ilustra a quantidade de minúcias que são removidas, esse é o resultado com uma janela de $16x16$ pixels. A imagem a esquerda serve de referência para notar como as minúcias falsas, eventualmente uma ou outra minúcia verdadeira é removida.

3.5 *Matching*

Serão apresentados aqui os resultados da fase de *matching*. Em decorrência do enfoque das técnicas escolhidas serem de minimizar o falso-positivo, espera-se uma taxa de falsos-



(a) Número completo de minúcias (b) Com a retirada as falsas

Figura 3.9: Visualização da quantidade de minúcias removidas

negativos alta. Mas, como já citado anteriormente, o pior resultado quando se lida com segurança é um falso-positivo que acarreta em uma entrada não autorizada, já para o caso do falso-negativo basta o usuário refazer o teste.

Para testar as duas técnicas de *matching* foram criados dois grupos de imagens. Um grupo com imagens de impressões digitais de diferentes pessoas, e outro com várias imagens diferentes da mesma impressão. Assim o primeiro grupo será usado para testar a porcentagem de falsos-positivos enquanto o segundo grupo será usado para testar a de falsos-negativos. Como a quantidade de imagens é pequena, no caso do primeiro grupo que possui cerca de 50 imagens, haverá uma permutação entre as imagens, de modo que, cada imagem será testada com todas as outras, assim se consegue uma quantidade maior de testes. No caso do segundo grupo, caso as imagens sejam insuficientes basta capturar mais, visto que esse grupo usa sempre o mesmo dedo. Um ponto a ser notado é que essa base está livre dos erros de hardware na aquisição que as vezes acontecem, já que a aquisição foi assistida. Quando era notado um erro na aquisição, a imagem era ignorada.

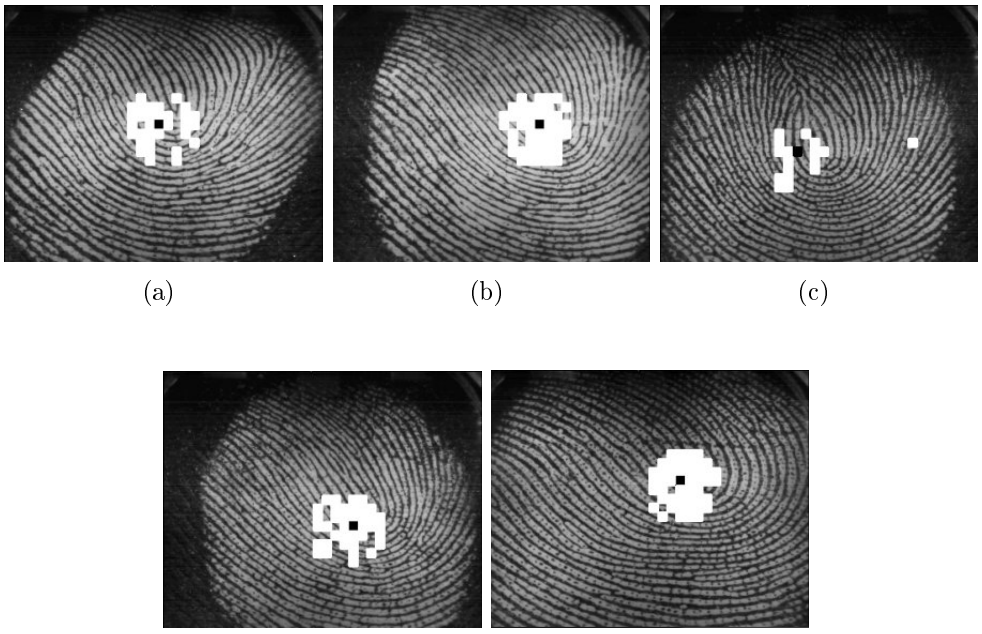
3.5.1 Detecção da Minúcia Central

O algoritmo do índice do *Poincaré* necessitou de diversos ajustes para que ele fosse melhor implementado. Uma das modificações feitas propostas por Ling Hong e Anil Jain [11], foi a inicialização de uma outra matriz que guardará os possíveis pontos. Depois de devidamente povoada os pontos de *core* ou *delta* serão o centroide de uma porção de pontos conectados.

Na Figura 3.10 temos que os quadrados brancos representam os possíveis pontos de *core* e o quadrado preto representa o ponto selecionado como sendo o verdadeiro. O ponto *delta* foi desconsiderado nessas imagens. Nota-se que apesar de bastante próximo do ponto real raramente essa técnica o acerta exatamente.

3.5.2 *Adaptive Elastic String Matching Algorithm*

Com a base de imagens de 50 imagens foi estimado a quantidade de falsos-positivos. Novamente os ajuste feitos nas técnicas de *matching* estão priorizando a diminuição dos



(a) (b) (c)

(d) (e)

Figura 3.10: Resultado do índice de Poincaré

falsos-positivos de modo a aumentar a segurança, o que provoca um aumento nos falsos-negativos. Caso o usuário queira ter maiores chances de ser identificado na primeira tentativa ele pode aumentar o limiar de penalidade de modo a aceitar impressões digitais mais diferentes.

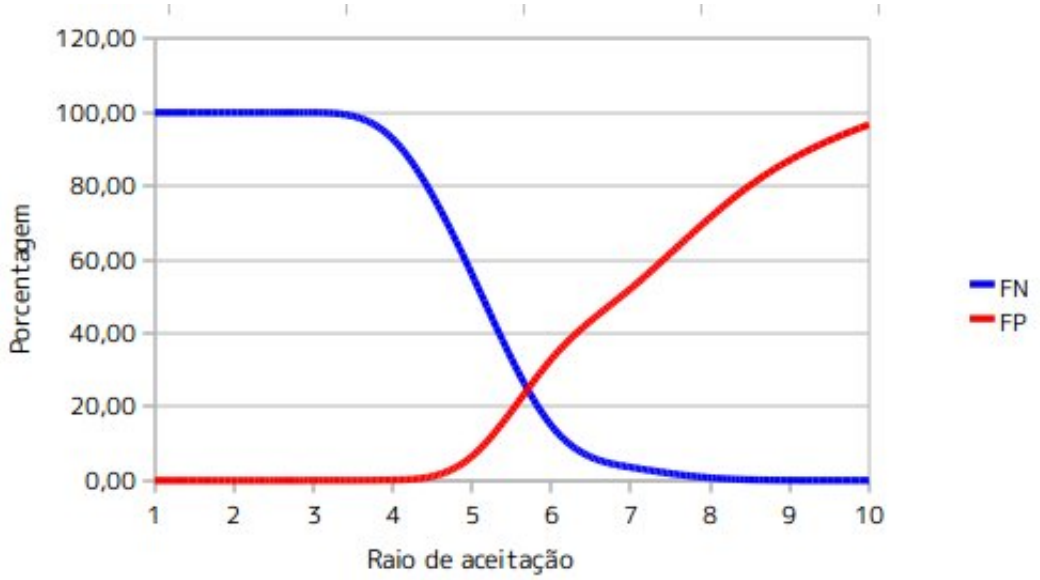
Os valores estabelecidos para cada tipo de penalidade são os seguintes:

1. Para translação das minúcias: 1 multiplicado pela distancia a transladar: 5 pixels de distância máxima permitida;
2. Para a inclusão de uma minúcia nova: 50;
3. Para a exclusão de uma minúcia antiga: 40;
4. Limite de minúcias a serem lidas: 20;
5. Limiar para identificação: 350.

O gráfico apresentado na Figura 3.11 ilustra o processo de determinação do melhor raio de aceitação para a translação. Esse raio representa a distância máxima que a minúcia pode estar deslocada para não ter que ser adicionada ou excluída. A menor penalidade deve ser para esta característica, então a alteração desse raio influi diretamente na quantidade de falsos-positivos e falsos-negativos. Quanto maior o raio, maiores as chances de um falso-positivo.

Pelo gráfico notamos que para um raio muito pequeno nenhuma minúcia é acertada, o que provoca um falso-negativo de 100%. Apenas a partir do raio de aceitação de 5 pixels, os falsos-negativos começam a cair, e caem imediatamente para 57.14%, ou seja a cada duas entradas o processo de identificação errará uma. Outro detalhe é que o falso-positivo cresce de forma muito rápida conforme o raio é aumentado. Como essa é a característica

que deve ser minimizada a todo custo, foi escolhido o raio de 5 pixels que possui uma taxa de falsos-positivos de 0.71%. O que ainda é considerada alta.



(a)

Figura 3.11: Falsos-positivos(FP) e falsos-negativos(FN) para o algoritmo das penalidades

3.5.3 Técnica alternativa: *Matching* em espiral

Essa técnica usou a mesma base de dados que a anterior e, do mesmo modo, houve prioridade para a redução de falsos-positivos em detrimento do aumento dos falsos-negativos. Caso o usuário queira diminuir a chance de falsos-negativos pode-se diminuir a porcentagem mínima de acertos para a identificação. Não é recomendável apenas aumentar o raio para considerar que as minúcias estão concêntricas pois, caso a aquisição dê errado em decorrência do hardware, haverão muitas minúcias em pouco espaço o que produzirá uma área com uma chance muito grande de coincidir erroneamente com qualquer minúcia do *template*.

As contantes definidas para esta técnica foram as seguintes:

1. Porcentagem mínima de acertos para identificação: 50%;
2. Raio para considerar que as minúcias estão concêntricas: 13 pixels.

Ao analisar o gráfico da Figura 3.12, pode-se notar uma grande diferença de performance entre os dois algoritmos. Com a mesma base de dados utilizada, o algoritmo de *matching* em espiral conseguiu a notória marca de 0.00% de falsos-negativos e falsos-positivos com o raio de 13 pixels. Em outras palavras, o algoritmo não errou nenhuma comparação, já que não obteve nenhuma identificação falsa. Na prática, no entanto, ele não se manteria nesse patamar porque, como explicado anteriormente, essa base de dados está livre do defeito de aquisição de hardware. Porém mesmo considerando esse erro o algoritmo de *matching* em espiral se mostrou o melhor em termos de acertos.

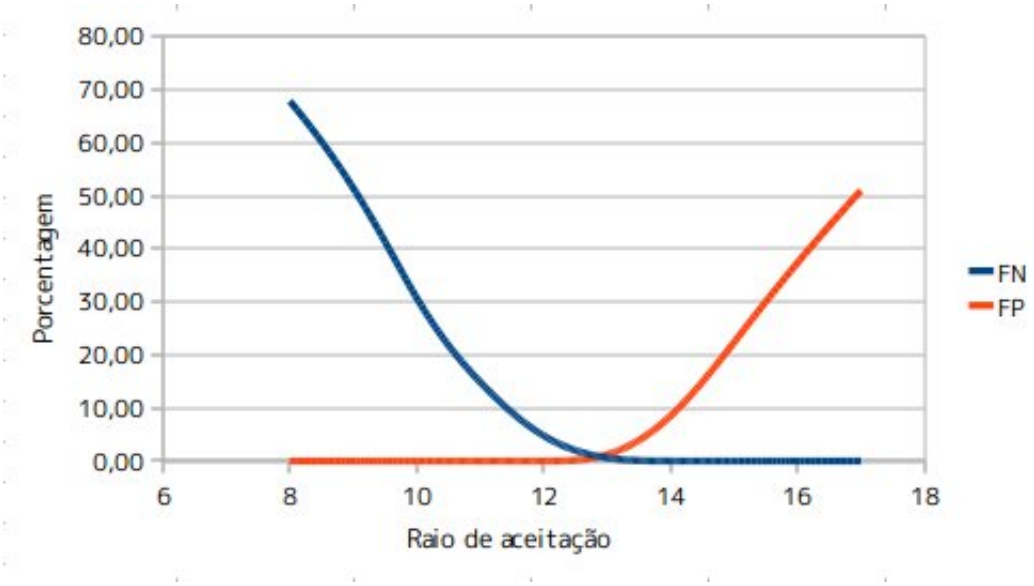


Figura 3.12: Falsos-positivos(FP) e falsos-negativos(FN) para o algoritmo em espiral

Porém como desvantagem temos o tempo de execução. O algoritmo de *matching* usando as penalidades é cerca de cem vezes mais rápido que o em espiral. Seus tempos foram em média: 55.02 e 5198.30 microssegundos respectivamente. Apesar de pela a identificação biométrica depender da entrada do usuário, um tempo de 5000 microssegundos é imperceptível. Essa demora se torna um problema caso queria comparar milhares de impressões digitais. Lembrando que estes tempos são apenas para os processos de *matching* e não estão sendo considerados os tempos do pré e pós-processamento.

Capítulo 4

Conclusão

Este trabalho apresentou resultados acima do esperado para a identificação. Era esperado possuir uma taxa de falsos-positivos entre 0.01 e 0.5, porém com a base de dados usada o número de falsos-negativos e falsos-positivos foi irrelevante, para o caso do *Matching* em Espiral. É óbvio que em uma situação real a implementação não manterá esse patamar, as fotos colocadas para teste foram escolhidas de modo a minimizar os erros de hardware gerando poucas minúcias falsas.

Porém apesar de ter um bom resultado, a técnica do *matching* em espiral é muito mais lenta que a técnica das penalidades, cerca de 100 vezes. Assim caso se tenham que fazer diversas comparações, como por exemplo buscando os dados de uma pessoa com base em sua impressão digital, o melhor é usar o *matching* das penalidades, ajustado para um alto índice de falsos-positivos, como primeira passada para só então utilizar o *matching* em espiral para determinar exatamente quem é a pessoa. Dessa forma só seria usado o algoritmo mais lento quando o conjunto de minúcias a serem testadas fosse bastante reduzido.

Assim com base nos resultados alcançados com os dois algoritmos de *matching* pode-se dizer que a hipótese inicial de que com apenas minúcias se pode identificar alguém foi confirmada. E a taxa de erros foi muito boa. Caso o pré-processamento melhore ainda mais, talvez essa taxa possa ser mantida em uma situação real também. Além disso para o conjunto de imagens usado, o programa implementado ficou proximo dos valores citados na tabela dos melhores identificadores [15].

4.1 Trabalhos Futuros

1. Estender a identificação para toda a palma da mão;
2. Classificar o tipo de impressão digital a fim de criar um índice e agilizar a busca em um banco de dados para a ciência forense;
3. Utilizar essa forma de biometria em conjunto com outra forma, como a voz por exemplo, assim teríamos um identificador biométrico bimodal, ou tri-modal.

Referências

[1] J C Amengual, A Juan, J C Pérez, F Prat, S Sáez, and J M Vilar. Real-time minutiae extraction in fingerprint images. *In Proc. of the 6th Int. Conf. on Image Processing and its Applications*, 1997. 19, 38

[2] Jin Bo, Tang Hua Ping, and Xu Ming Lan. Fingerprint singular point detection algorithm by poincaré index. 28

[3] A. Conci, E. Azevedo, and F. R. Leta. *Computação Gráfica: Teoria e Prática, Volume 2.* Editora Campus, 2008. 7, 12

[4] J.G. Daugman. Uncertainty relation for resolution in space, spatial frequency, and orientation optimized by two-dimentional visual cortical filters, July 1985. 14

[5] D. R. Faria. Reconhecimento de impressões digitais com baixo custo computacional para um sistema de controle de acesso. Master's thesis, Universisdade do Paraná, Dezembro 2005. 17

[6] F. Galton. Fingerprints, 1982. 1

[7] R. C. Gonzalez and R. E. Woods. *Digital Image Processing.* Publisher Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 2006. 2

[8] Z. Guo and R. Hall. Parallel thinning with two-subiteration algorithms. *Communications of the ACM*, 32, March 1989. 17

[9] W.W. Hines, D.C. Montgomery, D.M. Goldsman, and C.M.Borror. *Probabilidade e Estatística na Engenharia*, volume Quarta Edição. LTC, 2006. 7

[10] C.M. Holt, A. Stewart, M. Clint, and R.H. Perrot. An improved parallel thinning algorithm. *Communications of the ACM*, 30:156–160, February 1987. 18

[11] Lin Hong and Anil Jain. Classification of fingerprint images. 39

[12] A. Jain, L. Hong, S. Pankanti, and R. Bolle. *Proceedings of IEEE (Special Issue on Automated Biometrics)*, 85:1365–1388, September 1997. x, 27, 28

[13] A. Jain and S. Pankanti. Fingerprint classification and matching. 6, 26, 27

[14] S. Kasaei, M. Deriche, and B. Boashash. Fingerprint featufce extraction using block-direction on reconstructed images. *IEEE TENCON - Speech and Image Technologies for Computing and Telecommunications*, pages 303–306, 1997. x, 13, 34, 36

[15] D. Maltoni, June 2004. 3, 43

[16] D. Maltoni, D. Maio, A.K. Jain, and S. Pabhakar. Fingerprint image analysis for automatic identification. *MAchine Vision and Applications*, 6:124–139, 1993. 7

[17] K. Mitnik. *A Arte de Enganar*. Publisher Pearson Education, 2003. 1

[18] R. Thai. Fingerprint image enhancement and minutiae extraction, 2003. x, 1, 5, 6, 14, 15, 16, 23, 24, 25

[19] M. Tico and P. Kuosmanen. An algorithm for fingerprint image post-processing. *Proceedings of the Thirty-Fourth Asilomar Conference on Signals, Systems and Computers*, 2:1735–1739, November 2000. 22

[20] M. A. Tompson. *Livro Proibido do Curso de Hacker*. Editado pela ABSI - Associação Brasileira de Segurança da Informação, 2004. 1

[21] T.Y. Zhang and C.Y. Suen. Fast thinning algorithm for thinning digital patterns. *Communications of the ACM*, 27:236–239, March 1984. 17

Apêndice A

Convolução

A operação de convolução serve para passar um filtro sobre uma imagem no domínio espacial, isso significa que ela opera diretamente sobre os pixel. Para cada filtro deve ser usada uma determinada máscara para transladá-la pela imagem. As máscaras também são conhecidas como *kernels* ou *templates*.

A operação de convolução é expressa como:

$$G(i,j) = \sum_{u=-\frac{W}{2}}^{\frac{W}{2}} \sum_{v=-\frac{W}{2}}^{\frac{W}{2}} O(u,v)I(i+u,j+v) \tag{A.1}$$

Onde $G(i,j)$ é o resultado da convolução em volta do pixel $I(i,j)$ e $O(u,v)$ é a máscara de convolução de dimensões WxW. Assim na prática o que ocorre é a multiplicação de cada valor da mascara sobreposta em uma área da imagem centrada em (i,j). Para aplicar o filtro em toda a imagem, essa operação seria representada por:

$$G = O * I.$$

Com G sendo a imagem gerada pela convolução da imagem original I com a máscara de convolução O. Isso representa que a máscara foi "passada"por cada pixel da imagem deslocando um pixel por vez a máscara. Não há necessidade da máscara de convolução ser sempre quadrada, mas a maioria das máscaras é.

Para lidar com os pixels da borda em que a sobreposição da máscara acabaria saindo da imagem pode-se usar duas estratégias mais comuns:

1-Considerar os pixels fora da imagem como o valor zero;

2-Começar a convolução a partir do $\frac{W}{2}$ pixel da linha e da coluna e ir sempre até ao $T - \frac{W}{2}$ pixel com T sendo o total de pixels na linha ou coluna.

Apesar de mais correto a primeira alternativa implica em uma complexidade um pouco maior e caso se esteja mais interessado em velocidade colocar mais condicionais dentro de um loop (que são as somatórias) aumentam consideravelmente o tempo de processamento. Por isso a alternativa mais comum é ignorar os pixels da borda começando pelos mais ao centro. Assim basta mudar um pouco o parâmetro dentro da checagem no loop para tal.

Abaixo há um exemplo de função escrita em linguagem C que implementa uma convolução genérica. Para a função abaixo deve-se ter em mente apenas algumas considerações:

-A imagem está sendo representada como um vetor bidimensional de tipo *char* sem sinal;

-Os parâmetros *image_x_size* e *image_y_size* representam as dimensões da imagem em x e y, igual a sua resolução;

-A matriz da mascara de convolução é quadrada de tamanho *mask*;

```
void convulation(unsigned char **image_orig, unsigned char **image_dst, int
    **mask, int masksize, int image_x_size, image_y_size){
    int i,j,u,v;     //variaveis de indice
    int aux;     //variavel auxiliar apra guardar a soma
    for(j=masksize/2; j<image_y_size-masksize/2; j++)
        for(i=masksize/2; i<image_x_size-masksize/2; i++){
            aux=0;  //deve-se zerar a variavel auxiliar para guardar a soma
            for(v=-masksize/2; v<masksize/2; v++)
                for(u=-masksize/2; u<masksize/2; u++){
                    aux += mask[u][v] * image_orig[i-u][j-v];
                }
            //depois da soma deve-se checar se o resultado nao extrapolou
                os valores maximos e minimos do tipo char.
            if(aux>255){
                image_dst[i][j] = 255;
            }else if(aux<0){
                image_dst[i][j] = 0;
            }else{
                image_dst[i][j] = aux;
            }
        }
}
```

Alguns exemplos de matrizes de convolução e suas funções:

$$\text{Filtro de média: } \begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix}$$

$$\text{Filtro Prewitt: } G_y \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} G_x \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

O filtro de média "embaça"a imagem diminuindo o efeito do *allising* e com isso deixa as linhas mais contínuas. O filtro de Prewitt destaca as linhas da imagem e é usado exatamente como o filtro de Sobel.

Apêndice B

Equalização de Histograma

A seguir há um código escrito em linguagem C que implementa a equalização de histograma para uma imagem representada pela estrutura *"struct imagem"*.

```
typedef struct imagem{
    int height;
    int width;
    unsigned char *data;
} imagem;


void equalization(struct imagem *img){
    unsigned char aux[img->width*img->height];
    int histogram[256];
    unsigned int cdf[256];
    unsigned int cdfmin=0,cdftot=0;
    long total=0;
    float faux;
    int x,y;
    for(x=0;x<256;x++){
        histogram[x] = 0;
        cdf[x] = 0;
    }for(y=0;y<img->height;y++)
        for(x=0;x<img->width;x++){
            if(img->data[x+y*img->width]!=0){
                histogram[img->data[x+y*img->width]]++;
                total++;
            }
        }
    for(x=1;x<256;x++){
        cdftot += histogram[x];
        cdf[x] = cdftot;
    }
    x=0;
    while(cdfmin==0) cdfmin = cdf[x++];
    for(y=0;y<img->height;y++)
        for(x=0;x<img->width;x++){
            if(img->data[x+y*img->width]!=0){
                aux[x+y*img->width] = ((cdf[img->data[x+y*img->width]] -
                    cdfmin) / (float)(total - cdfmin))*255;
            }
```

```
            }
        for(y=0;y<img->height;y++)
            for(x=0;x<img->width;x++){
                if(img->data[x+y*img->width]!=0){
                    img->data[x+y*img->width] = aux[x+y*img->width];
                }
            }
}
```