Biometria por Imagem com Avaliação de Deslocamento e Iluminação

Damiana Costa Érick Oliveira Leonardo Carvalho Thiago Malheiros

Universidade Federal Fluminense Niterói, Brasil

Resumo

Este relatório visa descrever algoritmos que possam analisar de uma forma geral um tipo de biometria baseada na análise de imagens, assim como propor uma aplicação que realiza graficamente todos os algoritmos aqui descritos.

Palavras chave: Análise de imagens, reconhecimento, digitais, biometria, finger-knuckle-print.

1. Introdução

Como trabalho de implementação da cadeira de Análise de Imagens da Universidade Federal Fluminense escolhemos construir um sistema que fosse capaz de analisar características biométricas da articulação superior de qualquer dedo da mão e de qualquer pessoa dentro de certos parâmetros de entrada, trabalhando impreterivelmente em escala de cinza.

2. Banco de Imagens

Como passo inicial para análise e desenvolvimento da aplicação existia a necessidade da utilização de um banco de imagens já preparado e com algumas significantes instâncias operáveis. Logo, após algumas buscas, o banco escolhido foi um banco de imagens cecido pela Universidade Politécnica de Hong Kong [1]. Os pesquisadores desta Universidade coletaram imagens de 165 voluntários, sendo destes 125 homens e 40 mulheres, todos entre 20 a 50 anos de idade. Posteriormente, desenvolveram um identificador biométrico que realiza a identificação do indivíduo através da superfície externa em torno da articulação da falange medial de um dedo.

Figura 1. Dispositivo de Captura biométrica da Universidade Politécnica de Hong Kong [1].

2.1 Padrão de Imagem Utilizada

A partir do banco de imagens cedido pela Universidade Politécnica de Hong Kong [1] foram selecionadas 30 imagens para a realização das transformações, aplicações de filtros e análises diversas requeridas ao longo do desenvolvimento deste trabalho, esses detalhes serão melhores descritos nos tópicos seguintes.

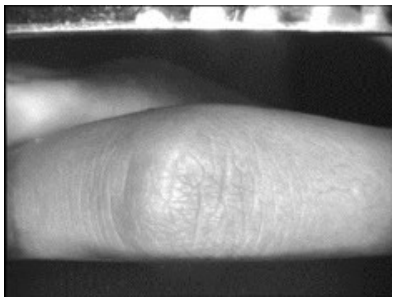

Figura 2. Uma das imagens que constituem o banco de dados cedido pela Universidade Politécnica de Hong Kong [1].

3. Filtros e Análise

Após a seleção do banco de imagens e das imagens em si, nossa equipe aplicou diversas transformações na imagem original com o objetivo realçar e avaliar os padrões de ocorrência das linhas de expressão da imagem de entrada. Os seguintes filtros foram aplicados às imagens, e, seus respectivos histogramas também foram gerados:

- Imagem equalizada (equalização do histograma)
- Imagem com equalização adaptativa (AHE)
- Imagem com equalização adaptativa com limitação de contraste (CLAHE)

Para aplicar estes filtros, a linguagem de programação escolhida foi o Python [2], a biblioteca de processamento de imagens Skimage [3] também foi utilizada.

3.1 Equalização do Histograma

O objetivo aqui é tentar distribuir a ocorrência dos diferentes tons na imagem de uma forma mais linear, e, portanto, originando uma nova imagem com maior contraste. Realizamos a equalização global do histograma e o aumento no contraste pode ser notado no quadro da direita na Figura 3 a seguir:

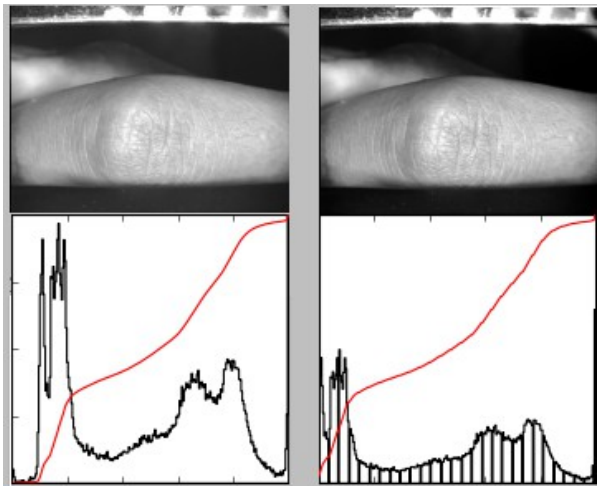

Figura 3. Comparação da imagem original com a imagem equalizada.

3.2 Equalização Adaptativa do Histograma

A equalização adaptativa do histograma, diferente da anterior, avalia vários histogramas, cada um correspondente a uma sessão diferente da imagem, e as usa para redistribuir os valores de luminosidade da imagem. Este tipo de equalização é adequada para melhorar o contraste local de uma imagem, e assim realçar mais detalhes na mesma. A aplicação deste filtro pode ser vista na Figura 4 a seguir.

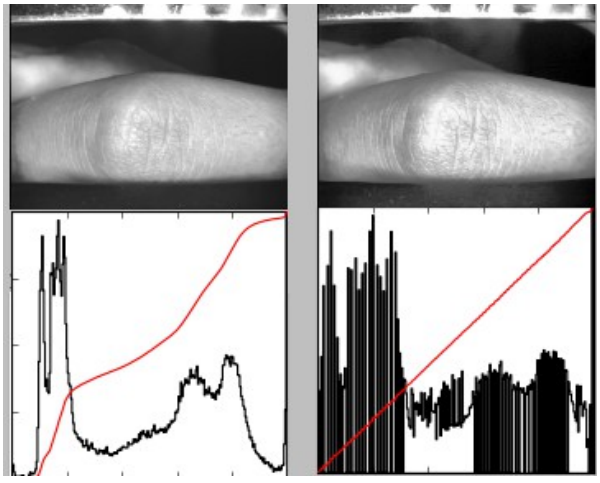

Figura 4. Comparação da imagem original com a imagem pós aplicação da equalização adaptativa.

Neste caso, foi possível verificar que após a aplicação deste filtro, se tornaram bem visíveis alguns detalhes no dedo do indivíduo que antes não eram tão notáveis.

3.3 Equalização Adaptativa do Histograma com limitação de contraste.

Este tipo de filtro é uma versão melhorada da equalização adaptativa. Podemos equalizar de forma adaptativa escolhendo um valor limite. A equalização adaptativa com limite de contraste opera em pequenas regiões da imagem, chamadas de janelas em vez de operar em toda a imagem. Cada janela de contraste é reforçada de modo que o histograma da região de saída corresponda aproximadamente ao histograma especificado pelo de distribuição. As janelas vizinhas são então combinadas através de uma interpolação bilinear a fim de realçar os limites de tons da imagem. O resultado da aplicação deste filtro pode ser visto na imagem a seguir.

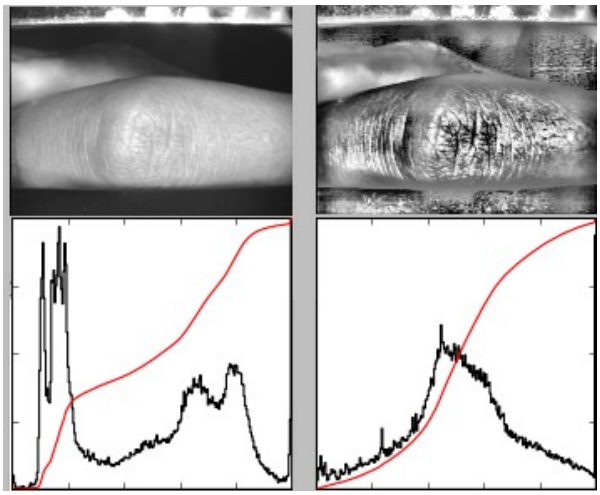

Figura 5. Comparação da imagem original com a imagem adaptada com limite de contraste.

3.4 Limiarização

A limiarização de uma imagem, de maneira rude, se dá pelo fato de avaliar e transformar os níveis de cinza de uma imagem em outra contendo apenas os tons pretos e brancos (binária). A imagem a seguir exibe o resultado obtido com a aplicação do filtro de limiarização:

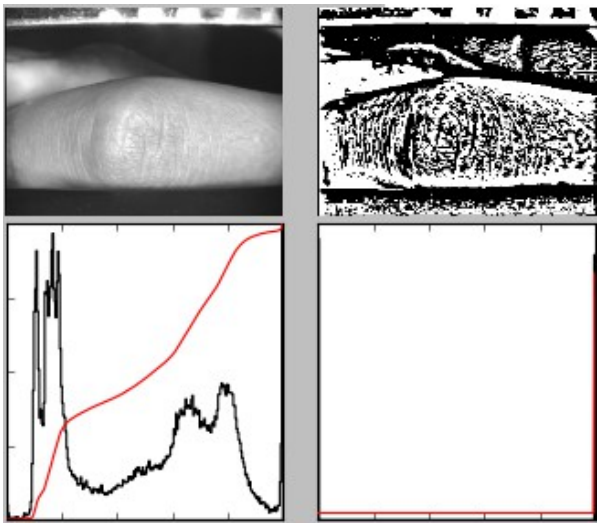

Figura 6. Comparação da imagem original com a imagem limiarizada.

3.5 Ratificação

Todas as aplicações foram essenciais para chegarmos a uma forma final de implementar o algoritmo que identifica essas linhas de expressão e as converte em uma imagem binária. A imagem limiarizada (3.4), por exemplo, apresenta muitos ruídos e não realça efetivamente as linhas de expressão de uma forma muito inteligente, entretanto, a avaliação do sentido deste filtro é interessante. A limiarização "puxa" os tons mais escuros para preto e os tons mais claros para branco. Essa é a ideia que utilizamos como base para identificar as linhas de expressão de uma forma um pouco diferente.

4. Desenvolvimento

Apesar de termos aplicado todos os filtros presentes no tópico anterior, nenhum deles foi efetivamente aplicado durante o processo de encontrar as linhas de expressão dos dedos. Entretanto, os mesmos podem, ainda, em uma modificação futura, serem adaptados conjuntamente para melhorar a taxa de acerto do algoritmo.

O algoritmo final completo e comentado, contendo todos os processos de identificação, análise e síntese das imagens, assim como os de comparação para identificar um certo usuário previamente cadastrado em um banco de dados arbitrário pode ser encontrado no APÊNDICE A ao final deste documento.

4.1 Idenficando as Linhas de Expressão

Avaliamos uma janela contendo uma quantidade arbitrária de pixels para encontrar as variações de tom na imagem. Definimos dois parâmetros de entrada, que são:

- Tamanho da Janela
- Variação de Tom

Empiricamente, chegamos nos valores de 9 pixels para o tamanho da janela e de 3 tons de diferença na variação de tom. Essas medidas apresentaram imagens binárias finais interessantes para nossa aplicação.

85	83	84	76	78	78	80	81	86
0	1	2	3	4	5	6	7	8

Figura 7. Exemplo de uma janela arbitrária com tamanho 9, índices dos pixels variando de 0 a 8.

Dado uma janela da imagem de entrada com tamanho de 9 pixels representada pela Figura 7, com a iluminação ou o valor de cada pixel variando de 0 a 255, podemos inferir que a variação de tom do pixel 8 em relação ao pixel 4 assim como a variação do pixel 0 em relação ao 4 é maior que 3 tons. Logo, marcamos o pixel 4 com um pixel preto. Essa lógica em janela foi aplicada percorrendo toda a imagem de entrada original e avaliando as variações de tons que queremos encontrar. A mesma lógica posteriormente também foi aplicada na vertical, bilinearmente. Após todo esse processamento temos uma imagem binária gerada:

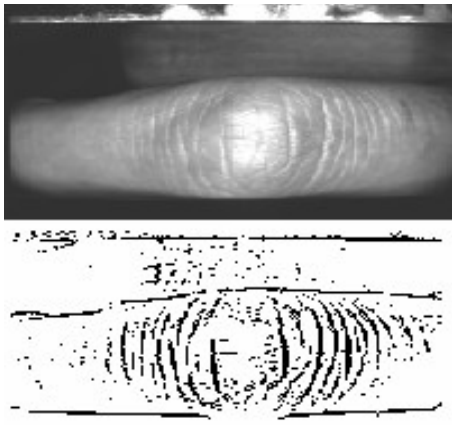

Figura 8. Linhas de expressão identificadas após a aplicação do algoritmo descrito.

4.2 Remoção de Ruídos

É possível notar na Figura 8 uma quantidade grande de ruído na imagem, muitos pontos e pedaços que efetivamente não precisariam ser resaltados. A fim de amenizar esse problema, aplicamos um algoritmo para retirar boa parte dos ruídos. A lógica utilizada se concentra em percorrer todos os pixels da imagem binária, para cada pixel, uma janela que envolve o mesmo precisa conter uma quantidade mínima de pixels pretos, caso não contenha, esse pixel é pintado de branco. Após o processamento desta etapa, obtemos o seguinte resultado:

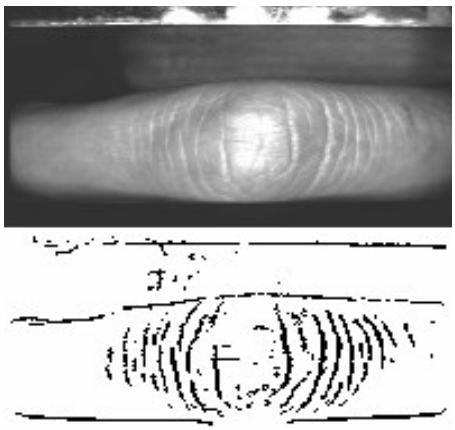

Figura 9. Em cima, imagem original de entrada, em baixo, imagem binária após a identificação das linhas de expressão e do tratamento de ruídos.

4.3 Região de Interesse (ROI)

A partir da imagem binária contida na Figura 9 precisamos extrair a região de interesse. O método de fácil implementação que utilizamos, porém um pouco custoso em termos de tempo de execução; encontra a posição do canto esquerdo superior em que o retângulo (ROI) agrupa uma maior quantidade de pixels pretos. O rentângulo mencionado é uma região retangular de largura e altura constante (alterável no algoritmo anexado). Após a processar a ROI, obtemos o seguinte resultado:

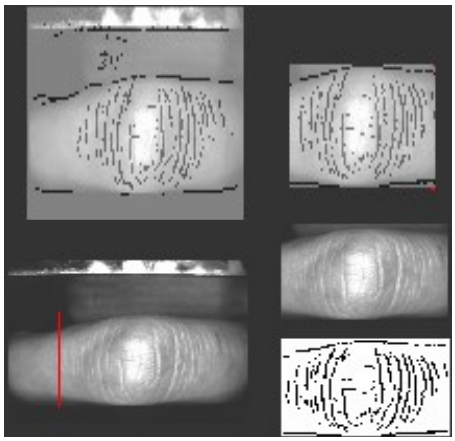

Figura 10. As imagens da direita indicam a extração da ROI da imagem à esquerda.

4.4 Identificação Biométrica

Para comparar uma imagem binária com uma outra arbitrária, construímos a comparação em cima da taxa de erro total absoluta por pixel, que se resume em avaliar a variação de pixels de mesma posição entre duas imagens diferentes de mesmo tamanho. Também utilizamos uma lógica que irá tratar o deslocamento do dedo e algumas possíveis diferenças de análise geradas por pequenas diferenças de iluminação.

Dado duas imagens de entrada que queremos comparar, aplicamos os seguintes passos:

1. Transformar a primeira imagem para binária.
2. Transformar a segunda imagem para binária.
    i. Extrair a ROI da segunda imagem.
3. Varrer ou deslocar a ROI da segunda imagem em cima da binária da primeira imagem, para cada deslocamento, calcular a taxa de erro.
4. A menor taxa de erro até o final do deslocamento será a taxa de erro de comparação entre as duas imagens.

Ao avaliarmos diferentes imagens, calculamos as suas respectivas taxas de erros como descrito acima, e, inferimos que a imagem que estamos procurando é a que contém a menor taxa de erro dentre as avaliadas. Testamos a acurácia de nosso algoritmo e conseguimos obter para 20 dedos diferentes, cada um com pelo menos 2 imagens distintas, capturadas em momentos diferentes, uma acurácia de 90%. Comparamos cada dedo com outros 5 dedos diferentes e em 90% das vezes o algoritmo acertou qual era realmente o dedo correto.

4.5 Aplicação

Além da construção de todos esses algoritmos também implementamos uma interface gráfica que está compilada e demonstra graficamente alguns dos processos descritos neste documento.

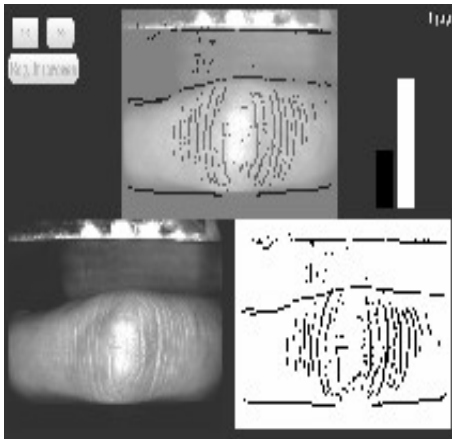

Figura 11. *Screenshot* da aplicação gráfica [5].

A aplicação contém à sua direita o histograma da imagem binária, um controlador de visualização das amostras (fotos das falanges) no canto superior esquerdo, e também o botão "Reg. Interesse" que, por sua vez, executa a função de localização da ROI (no nosso caso, a região ótima) de cada imagem.

5. Comparação

O artigo publicado pela Universidade de Hong Kong [4] calcula e leva em consideração, rudemente falando, apenas o deslocamento do dedo ao longo do

eixo X da imagem. Para avaliar a posição da ROI na imagem, devemos localizar a posição correta em Y e X tal que a maior parte das características para a identificação do indivíduo sejam preservadas dentro da região de interesse.

Partindo deste princípio, os autores aplicaram uma operação Gaussiana de suavização na imagem original, a operação gaussiana irá suprimir o ruído, o que pode beneficiar a extração de características. O algoritmo em questão seta o limite inferior da região de interesse em Y como o ponto inicial do dedo na vertical, de cima para baixo, invariante para basicamente todas as imagens do banco de dados. Uma projeção vertical do histograma é realizada para encontrar a região central do dedo em X (que possui uma menor variação de tons verticalmente, dada a definição de projeção vertical de histogramas). Após obter a região central do dedo em X, a largura da ROI é constante e, portanto, terá o seu ponto central horizontal na região central do dedo em X previamente extraída. A imagem a seguir ilustra os passos para reconhecimento que os autores do artigo seguiram até separar a região de interesse.

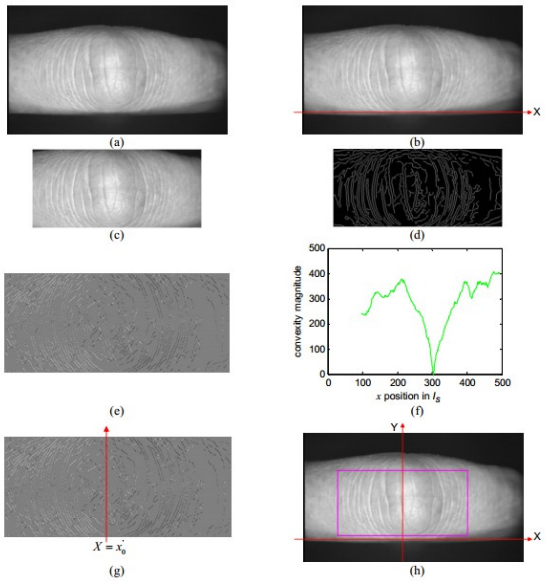

Figura 12. Ilustração para o processo de extração da região de interesse da imagem. (a) imagem obtida por uma operação de subamostragem depois de uma operação gaussiana, (b) o eixo X do sistema de coordenadas da imagem, (c) imagem extraída, (d). imagem obtida através do detector de aresta, (e) imagem obtida através da aplicação do esquema de direção convexa, (h) sistema de coordenadas da região de interesse, onde o retângulo indica a subimagem que será extraída (ROI) [4].

6. Conclusão

Nosso algoritmo possui um overhead durante a avaliação da ROI (diferente do algoritmo do artigo comparado), que, por sua vez, encontra a ROI de uma maneira mais rápida. Entretanto, avaliamos uma ROI muito mais significativa que a encontrada pelo algoritmo comparado, pois, no nosso caso, pegamos realmente a região mais significativa, com uma quantidade maior de dados, diferente da região central do dedo.

Além da flexibilidade de adaptação para diferentes ocasiões e uma variabilidade muito grande por parte da quantidade de parâmetros que estipulamos durante a construção do algoritmo, o mesmo apresentou uma grande acurácia ao comparar e identificar dedos iguais. Apesar de também fazer isso de uma forma lenta.

Os demais algoritmos são bastante eficientes e realizam sua função de forma correta.

Referências

[1] THE HONG KONG POLYTECHNIC UNIVERSITY. *Finger-Knuckle-Print Database* Disponível em: http://www4.comp.polyu.edu.hk/~biometrics/FKP.htm [Acesso em Junho 2013]

[2] PYTHON. Python Brasil. Disponível em: http://www.python.org.br/wiki. [Acesso em Junho 2013]

[3] SKIMAGE. Image Processing in python. Disponível em: http://scikit-image.org/docs/dev/api/skimage.html [Acesso em Junho 2013]

[4] ZHANGA Lin, ZHANGA Lei*, ZHANGA David and ZHUB Hailong. Online Finger-Knuckle-Print Verification for Personal Authentication http://www4.comp.polyu.edu.hk/~biometrics/FKP_files/FKP_PR_R1.pdf

[5] Implementação gráfica e código-fonte dos algoritmos da aplicação descritos ao longo do documento. https://sourceforge.net/projects/biometria/

Apêndices

```
//CLASSE PROCESSAMENTO
package com.me.biometria;

import java.util.ArrayList;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.graphics.Color;
import com.badlogic.gdx.graphics.Pixmap;
import com.badlogic.gdx.graphics.Pixmap.Format;
import com.badlogic.gdx.math.Vector2;
import com.badlogic.gdx.math.Vector3;

public abstract class Processamento {
    //classe que encapsula todos os processamentos modularizados
    protected static Pixmap getBinario(String texturepath, boolean imageminterna){
        //imagem de recebimento
        Pixmap px;
        if (imageminterna) px = new Pixmap(Gdx.files.internal(texturepath));
        else px = new Pixmap(Gdx.files.absolute(texturepath));
        //imagem binária a ser gerada e retornada
        Pixmap pxretorno = new Pixmap(px.getWidth(), px.getHeight(), Format.RGBA8888);
        //pintando a imagem de retorno de branco
        pxretorno.setColor(Color.WHITE);
        pxretorno.fill();
        //pré-processamento colocando as cores altas e baixas totalmente brancas para
        separar os contornos
        short limiteinferior = 20, limitesuperior = 210; //variando de 0 à 255
        for (int i=0; i<px.getHeight(); i++){
            for (int j=0; j<px.getWidth(); j++){
                if (Math.abs(px.getPixel(j, i)) < limiteinferior || Math.abs(px.getPixel(j, i
                )) > limitesuperior){
                    px.drawPixel(j, i, 0xFFFFFFFF);
                }
            }
        }
        //valor ímpar\/
        byte tamanhojanela = 9; //tamanho da janela de análise na horizontal, as linhas
        ficam mais grossas ao aumentar
        byte vtom = 3; //sensibilidade da mudança de tom, quanto maior precisa variar mais tom
        para que haja uma representação binária na janela que percorre a imagem original
        //excluir pixels
        byte excluiry = 20, excluirx = 10;
        //criar a imagem binária percorrendo a imagem original
        for (int i=excluiry; i<px.getHeight() - excluiry; i++){
            for (int j=excluirx; j<px.getWidth() - tamanhojanela - excluirx; j++){

                int pxmeio = px.getPixel(j + (int)(tamanhojanela/2), i);
                int variacao1 = px.getPixel(j, i) - pxmeio;
                int variacao2 = px.getPixel(j + tamanhojanela - 1, i) - pxmeio;
                if ((Math.abs(variacao1) > vtom) &&
                        ((Math.abs(variacao2)) > vtom) &&
                        (!isPositivo(variacao1) && !isPositivo(variacao2))){ //aqui pode ser
                        == também
                    pxretorno.drawPixel(j + (int)(tamanhojanela/2), i, 0x000000FF);
                }
            }
        }
        vtom = 4;
        tamanhojanela = 9;
```

```
    //aplicar a janela na vertical também, apenas repetindo o código acima for
    (int i=excluiry; i<px.getHeight() - excluiry - tamanhojanela; i++){
        for (int j=excluirx; j<px.getWidth() - excluirx; j++){

            int pxmeio = px.getPixel(j, i + (int)(tamanhojanela/2));
            int variacao1 = px.getPixel(j, i) - pxmeio;
            int variacao2 = px.getPixel(j, i + tamanhojanela - 1) - pxmeio;
            if ((Math.abs(variacao1) > vtom) &&
                    ((Math.abs(variacao2)) > vtom) &&
                    (isPositivo(variacao1) && isPositivo(variacao2))){ //aqui pode ser
                    == também
                pxretorno.drawPixel(j , i + (int)(tamanhojanela/2), 0x000000FF);
            }
        }
    }

    //limpar alguns ruídos
    byte c = 4, caux = c; //tamanho da janela para limpeza dos ruídos
    short mincontagem = 7; //ter pelo menos 'mincontagem' pixels pretos na janela
    (c*2)*(c*2)
    while (caux >= 1){
        if (caux == 1) {c = 3; mincontagem = 7;}else c = caux;
        byte contar = 0;
        for (int i=excluiry; i<pxretorno.getHeight(); i++){ for
            (int j=excluirx; j<pxretorno.getWidth(); j++){
                //for das janelas
                for (int h=c*-1; h<c; h++){
                    for (int f=c*-1; f<c; f++){
                        if (Math.abs(pxretorno.getPixel(j + f, i + h)) > 200){
                            contar ++;
                        }
                    }
                }
                if (contar < mincontagem) pxretorno.drawPixel(j, i, 0xFFFFFFFF);
                contar = 0;
            }
        }
        caux--;
        mincontagem --;
    }
    px.dispose();
    return pxretorno;
}
private static boolean isPositivo(int n){
    if (n > 0) return true;
```

```
    return
    false;
}

protected static int quantosPixelsPretos(Pixmap px){
    int cont =
    0;
    for (int i = 0; i < px.getHeight(); i++){
        for (int j = 0; j < px.getWidth(); j++){
            if (px.getPixel(i, j) != -1) cont ++;
        }
    }

    return cont;
}
```

```
protected static int quantosPixelsBrancos(Pixmap px){
    int cont =
    0;
    for (int i = 0; i < px.getHeight(); i++){
        for (int j = 0; j < px.getWidth(); j++){
            if (px.getPixel(i, j) == -1) cont ++;
        }
    }

    return cont;
}

protected static Pixmap clonarPixmap(Pixmap px){
    Pixmap px2 = new Pixmap(px.getWidth(), px.getHeight(), px.getFormat());
    for (int j=0; j<px.getWidth(); j++){
        for(int i=0; i<px.getHeight(); i++){
            px2.drawPixel(j, i, px.getPixel(j, i));
        }
    }
    return px2;
}
//variaveis para desenhar as linhas da região de interesse
protected static Vector2 comeco;
protected static int janelax = 265, janelay = 165;
protected static Pixmap getAreaInteresse(Pixmap pxr){
    ArrayList<Vector3> custos = new ArrayList<Vector3>();
    Pixmap px = clonarPixmap(pxr);

    for (int j=0; j<px.getWidth() - janelax; j++){
        for (int i=0; i<px.getHeight() - janelay; i++){
            //for da janela, pixel por pixel
            int contagem = 0; //quantos pixels pretos
            for (int j2=j; j2 < j+janelax; j2++){
                for (int i2=i; i2 < i+janelay; i2++){
                    if (px.getPixel(j2, i2) != -1) contagem++;
                }
            }
            custos.add(new Vector3(j, i, contagem));
        }
        System.out.println("Processando: " + (int)(j*100/(px.getWidth()-janelax)) + "%");
    }

    //pegar a janela ótima
    Vector3 otima = custos.get(0);
    for(int i=0; i<custos.size(); i++){
        if (otima.z < custos.get(i).z) otima = custos.get(i);
    }

    //deletar se quiser, passar pra uma variável externa os pontos de começo da área de
    interesse
    comeco = new Vector2(otima.x, otima.y);
    //deletar se quiser \
    //deletar qualquer área diferente da de interesse
    for(int j=0; j<px.getWidth();j++){
        for(int i=0; i<px.getHeight(); i++){
            if (!((j > otima.x) && (j < (otima.x + janelax))
                    && (i > otima.y) && (i < (otima.y + janelay)))){
                px.drawPixel(j, i, 0xFFFFFFFF);
            }
        }
    }
```

```
        return px;
    }
    public static Pixmap getAreaInteresseCortada(Pixmap px){
        Pixmap processado = getAreaInteresse(px);
        Pixmap saida = new Pixmap(janelax, janelay, Format.RGBA8888);
        int primeirox = 9999, primeiroy = 9999;
        for (int i=0; i<processado.getHeight(); i++){ for
        (int j=0; j<processado.getWidth(); j++){
            if (processado.getPixel(j, i) != -1)
                { if (primeirox > j) primeirox =
                j; if (primeiroy > i) primeiroy =
                i;
            }
        }
        for (int i=0; i<saida.getHeight(); i++){ for
        (int j=0; j<saida.getWidth(); j++){
            if (processado.getPixel(primeirox + j, primeiroy + i) != -1)
                saida.drawPixel(j, i, 0x00000000);
            else saida.drawPixel(j, i, 0xFFFFFFFF);
        }
        }
        return saida;
    }
    public static Pixmap getAreaInteresseCortadaBinaria(String texturepath, boolean
    imageminterna){
        return getAreaInteresseCortada(getBinario(texturepath, imageminterna));
    }
}
```

```
//CLASSE COMPARAÇÃO
package com.me.biometria;

import com.badlogic.gdx.ApplicationListener;
import com.badlogic.gdx.graphics.Pixmap;

public class Comparacao implements
    ApplicationListener{ private static byte qntdimagens
    = 5;
    private static Pixmap[] pixmap = new Pixmap[qntdimagens];
    private static int[] acerto = new int[qntdimagens]; private
    static final String diretorioimagens =
    "C:\\Users\\Érick\\Documents\\ibiometria\\";

    public void main (){
        //imagem no banco de dados
        pixmap[0] = Processamento.getBinario(diretorioimagens + "0.jpg", false);
        //imagens a serem comparadas
        for (int i=1; i<pixmap.length; i++){
            System.out.println("Avaliando a imagem " + i + ".jpg ->");
            pixmap[i] = Processamento.getAreaInteresseCortadaBinaria(diretorioimagens + i +
            ".jpg", false);
        }

        //separar o menor acerto entre as comparações com as imagens acima
        int menoracerto = 999999, indice = 0;
        for (int i=1; i<pixmap.length; i++){
            System.out.println("Comparando com a imagem " + i + ".jpg ...");
            acerto[i] = comparar(pixmap[0], pixmap[i]);
            if (acerto[i] < menoracerto) {
                menoracerto = acerto[i];
                indice = i;
            }
        }
        System.out.println("0.jpg <-> " + indice + ".jpg, Taxa de erro: " + menoracerto);
    }

    public int comparar(Pixmap px1, Pixmap px2){
        final byte tamanhojanela = 3;
        boolean incrementar;
        int taxaerro = 0;
        int menorsubtaxa = 99999999;
        for (int i=50; i<px1.getHeight() - px2.getHeight(); i++){//percorrer imagem original
        a ser comparada
            for (int j=40; j<px1.getWidth() - px2.getWidth() - 40; j++){
                taxaerro = 0;
                for (int i2=0; i2<px2.getHeight(); i2 ++){//andar com a ROI em cima da
                imagem original e verificar compatibilidade
                    for (int j2=0; j2<px2.getWidth(); j2++){
                        //taxa de erro absoluta entre cada pixel
                        if (px1.getPixel(j + j2, i + i2) != px2.getPixel(j2, i2)){
                            taxaerro ++;
                            //taxa de erro por janela
                            incrementar = true;
                            for (int h=(-1*tamanhojanela); h<tamanhojanela; h++){
                                for (int f=(-1*tamanhojanela); f<tamanhojanela; f++){
                                    if ((px1.getPixel(j + j2, i+ i2)!= -1) && (px2.getPixel(
                                    j2 + h, i2 + f) != -1))
                                        incrementar = false;
                                }
                            }
                            if (incrementar) taxaerro +=2;
                        }
                    }
```

```
            }
            //retornar a menor taxa de erro dentre todas as áreas avaliadas pela
            comparação da ROI com a imagem original
            if (menorsubtaxa > taxaerro) {
                menorsubtaxa = taxaerro;
            }
        }
        System.out.println("Porcentagem da comparação: " + i*100/(px1.getHeight() - px2.
        getHeight()) + "%");
    }
    return menorsubtaxa;
}
```