CENTRO UNIVERSITÁRIO DINÂMICA DAS CATARATAS

THIAGO CORREA LIMA DA SILVA

ESTUDO E IMPLEMENTAÇÃO DE UM SISTEMA DE RECONHECIMENTO DE FACES EM VÍDEO COM O ALGORITMO *EIGENFACES* NA LINGUAGEM JAVA

FOZ DO IGUAÇU

2018

THIAGO CORREA LIMA DA SILVA

ESTUDO E IMPLEMENTAÇÃO DE UM SISTEMA DE RECONHECIMENTO DE FACES EM VÍDEO COM O ALGORITMO *EIGENFACES* NA LINGUAGEM JAVA

Trabalho de conclusão de curso apresentado como requisito obrigatório para obtenção do título de Bacharel em Ciência da Computação do Centro Universitário Dinâmica das Cataratas.

Orientador: Prof. Ma. Alessandra Bussador

FOZ DO IGUAÇU

2018

S586e SILVA, Thiago da

Estudo e Implementação de um Sistema de Reconhecimento de Faces em Vídeo com o Algoritmo *EigenFaces* na linguagem JAVA / Thiago Correa Lima Da Silva – Foz do Iguaçu, 2018.

83 f. : il. color.;

Trabalho de Conclusão de Curso – Centro Universitário Dinâmica das Cataratas (UDC). 2018.

Orientador: Prof. Ma. Alessandra Bussador

Bibliografia: p. 79 – 82

1. Reconhecimento de Faces em Vídeo. 2. EigenFaces. 3. ACP - Análise de Componentes Principais. I. Prof. Ma. Alessandra Bussador. II. Centro Universitário Dinâmica das Cataratas. III. Ciência da Computação. IV. Estudo e Implementação de um Sistema de Reconhecimento de Faces em Vídeo com o Algoritmo *EigenFaces* na linguagem JAVA

CDU: 004

TERMO DE APROVAÇÃO

Thiago Correa Lima Da Silva

Estudo e Implementação de um Sistema de Reconhecimento de Faces em Vídeo com o Algoritmo *EigenFaces* na linguagem JAVA

Trabalho de conclusão de curso apresentado como requisito obrigatório para obtenção do título de Bacharel em Ciência da Computação do Centro Universitário Dinâmica das Cataratas - UDC, Foz do Iguaçu, pela seguinte banca examinadora:

Prof. Ma. Alessandra Bussador
Centro Universitário Dinâmica das Cataratas
(Orientador)

Prof. Banca 2
Centro Universitário Dinâmica das Cataratas

Prof. Banca 3
Centro Universitário Dinâmica das Cataratas

Foz do Iguaçu, 2018

Dedico este trabalho aos meus pais,
Sr Carlos Alberto L. da Silva e Sra Tania C. L. da Silva,
que tanto trabalharam e tanto desejaram a conclusão deste processo na minha vida.

AGRADECIMENTOS

Agradeço a tudo e a todos, especialmente ao meu superior que incentivou e possibilitou o desenvolvimendo deste projeto em ambiente de trabalho, e à minha orientadora que, com paciência e compreensão, me ajudou neste processo.

"Emancipate yourself from mental slavery,
none but ourselves can free our mind."
- Robert Nesta Marley (1979)

LISTA DE ILUSTRAÇÕES

FIGURA 1 – Exemplos de Variação 17
FIGURA 2 – Exemplos de EigenFaces 19
FIGURA 3 – Exemplos de EigenFaces 20
FIGURA 4 – Decomposição dos canais do sistema RBG. 28
FIGURA 5 – Conversão em tons de cinza pelo método máximo canal. 29
FIGURA 6 – Conversão em tons de cinza pelo método de conversão clássico . . . 29
FIGURA 7 – Conversão em tons de cinza pelo método de média de canal. 30
FIGURA 8 – Exemplo de escalonamento de imagem. 30
FIGURA 9 – (a) Função ou sinal discreto (b) Função ou sinal interpolado 30
FIGURA 10 – Exemplo de aplicação de equalização de histograma. 31
FIGURA 11 – Exemplo de aplicação de processamento de imagem. **A**: Segmentação **B**: Transformação de cores em escala de cinza. **C**: Escalonamento. **D**: Equalização de histograma 32
FIGURA 12 – Um exemplo de *Haar wavelet* 33
FIGURA 13 – Características Haar representando características relevantes face . . 34
FIGURA 14 – Cascata de classificadores com estágios (C0, C1, ..., Cn) 35
FIGURA 15 – Exemplo de imagens típicas de treinamento. 37
FIGURA 16 – Exemplo de uma imagem de treinamento representada em uma sequência de pesos de engenfaces. 38
FIGURA 17 – Exemplo de eigenfaces representadas em um plano cartesiano (*eigenspace*) de 3 dimensões. 39
FIGURA 18 – Plotagem dos valores de x e y. 39
FIGURA 19 – Plotagem normalizada dos dados de x e y com eigenvetores. 42
FIGURA 20 – Versão rotacionada e refletida da Figura 19 42
FIGURA 21 – Ilustração do processo do sistema. 52
FIGURA 22 – Diagrama de Caso de Uso - Interação entre o usuário e o sistema. . . 53
FIGURA 23 – Diagrama de pacotes UML do sistema proposto. 54
FIGURA 24 – Diagrama de classes UML do pacote ***gui***. 55
FIGURA 25 – Diagrama de classes UML do pacote ***eigenfaces***. 56
FIGURA 26 – Diagrama de classes UML do pacote ***utils***. 59
FIGURA 27 – Diagrama de sequência UML das classes *ACP_Treinamento*, *Matriz2D*, *FileUtils* e *EqualizarHistograma*. 61
FIGURA 28 – Diagrama de sequência UML das classes *Matriz2D*, *AutoVetor_decomp*, *DenseDoubleMatriz* e *EigenValueDecomp*. 62
FIGURA 29 – Imagem da interface do sistema. 69
FIGURA 30 – Imagem do Sistema em Funcionamento. 70

LISTA DE TABELAS

TABELA 1 – Empresas que produzem tecnologias de reconhecimento de faces . . 23
TABELA 2 – Dados dos vetores x e y. 40
TABELA 3 – Resultado dos testes (Fase 1 - Primeiro dia) 73
TABELA 4 – Resultado dos testes (Fase 1 - Segundo dia) 73
TABELA 5 – Resultado dos testes (Fase 2 - Primeiro dia) 74
TABELA 6 – Resultado dos testes (Fase 2 - Segundo dia) 74
TABELA 7 – Resultado dos testes (Fase 3) 75
TABELA 8 – Taxas de sucesso reais dos reconhecimentos nas fases de teste. 76

LISTA DE CÓDIGOS

CÓDIGO 1 – Bloco de código do método *trackFace()* da classe *PanelCadastroDeFace* 62
CÓDIGO 2 – Bloco de código do método *gerar_EigenSpace()* da classe *ACP_Treinamento* 64
CÓDIGO 3 – Bloco de código do método *calcEspaco()* da classe *ACP_Treinamento* - responsável por executar o quinto passo da fase de treinamento. . . 65
CÓDIGO 4 – Bloco de código do método *processaReconhecimento()* da classe *ACP_Reconhecimento* 66
CÓDIGO 5 – Bloco de código do método *calcEspaco()* da classe *ACP_Treinamento* 67
CÓDIGO 6 – Bloco de código do método *negocioReconhecerClip()* da classe *PanelCadastroDeFace* 68

LISTA DE GRÁFICOS

GRÁFICO 1 – Comparação de sistemas de reconhecimento de faces em percentagem (%) de erros 21
GRÁFICO 2 – **Gráfico "A"**: Reconhecimentos de sucesso (eixo Y) por quantidade de imagens usadas para o treino (eixo X). **Gráfico "B"**: Falhas no reconhecimentos (eixo Y) por quantidade de imagens usadas para o treino (eixo X). Referentes as fases de teste "1"e "2". 77
GRÁFICO 3 – **Gráfico "A"**: Reconhecimentos de sucesso (eixo Y) por quantidade de imagens usadas para o treino (eixo X). **Gráfico "B"**: Falhas no reconhecimentos (eixo Y) por quantidade de imagens usadas para o treino (eixo X). Referentes as fases de teste "3". . . 77

LISTA DE ABREVIATURAS

ACP	Análise de Componentes Principais
BMP	*Windows Bitmap*
CDDL	*Commun Development and Distribution License* - Licensa Comum de Desenvolvimento e Distribuição [1]
FBI	*Federal Bureal of Investigation* - Agência Federal de Investigação Americana [1]
FPS	*Frames Per Second* - Quadros por Segundo [1]
GNU/GPL	*General Public License* - Licensa Pública Geral[1]
GUI	*Graphics User Interface* - Interface Gráfica para Usuário [1]
IDE	*Integrated Development Environment* - Ambiente Integrado de Desenvolvimento [1]
JPG	*Joint Photographic Experts Group*
NSA	*National Security Agency* - Agência Nacional de Segurança [1]
PDI	Processamento de Imagens
PCA	*Principal Component Analisys* - Análise de Componentes Principais
PIXELS	*Picture Elements*
PNG	*Portable Network Graphics* - Gráficos Portáteis para Rede [1]
RGB	*Red Blue Green*
UML	*Unified Modeling Language* - Linguagem de Modelagem Unificada [1]
XP	*eXtreme Programming*

SUMÁRIO

1 INTRODUÇÃO 15
1.1 Breve História 15
1.2 Problemática 16
1.2.1 Detecção e Aquisição da Face 16
1.2.2 Treinamento e Reconhecimento da Face 18
1.3 Objetivos 20
1.3.1 Objetivos Específicos 20
1.4 Justificativa 21

2 REVISÃO BIBLIOGRÁFICA 24
2.1 Conceitos de Aquisição e Processamento de Imagens 24
2.1.1 Imagem Digital 24
2.1.2 Formato PNG 25
2.1.3 Vídeo Digital 26
2.1.4 Aquisição da Imagem e de Vídeo 26
2.1.5 Processamento da Imagem 26
2.1.5.1 Conversão em Escala de Cinza 27
2.1.5.2 Escalonamento 28
2.1.5.3 Equalização de Histograma 31
2.1.5.4 Segmentação 31
2.1.5.5 Classificação 32
2.2 Detecção de Faces com Classificadores Haar 32
2.2.1 Os Características Haar (*Haar Features*) 33
2.2.2 Algoritmo Viola-Jones (*Haar Cascades Classifier*) 34
2.3 PROCESSO DA ANÁLISE DE COMPONENTES PRINCIPAIS (ACP) OU *EIGENFACES* 36
2.3.1 O Cálculo da Análise de Componentes Principais (ACP) . . . 38
2.3.1.1 Fase de Treinamento 38
2.3.1.2 O Reconhecimento 43

3 MATERIAIS E MÉTODOS 44
3.1 Materiais 44
3.1.1 Hardwares 44
3.1.2 Softwares 44
3.1.2.1 Linguagem Java 44

3.1.2.2 Biblioteca OpenCV, JavaCV e JavaCPP 45
3.1.2.3 Biblioteca Colt 47
3.1.2.4 NetBeans IDE 48
3.1.2.5 Sistema GIT 48
3.1.2.6 Sistemas Linux e Windows 48
3.2 Métodos 48
3.2.1 Método Experimental 48
3.2.2 Desenvolvimento Ágil 49
3.2.2.1 XP (eXtremeProgramming) 49
3.2.3 Modelagem UML 49

4 IMPLEMENTAÇÃO 51
4.1 Análise de Requisitos 51
4.2 Diagrama de Caso de Uso 52
4.3 Diagrama de Pacotes 52
4.4 Diagrama de Classes - Pacote ***gui*** 53
4.5 Diagrama de Classes - Pacote ***eigenfaces*** 56
4.6 Diagrama de Classes - Pacote ***utils*** 58
4.7 Implementação do Código 61
4.7.1 Constantes Controladas 61
4.7.2 Processo Principal do Sistema 62
4.7.3 Fase de Treinamento 64
4.7.4 Fase de Reconhecimento 66
4.7.5 Definição de Sucesso do Reconhecimento 68
4.7.6 Interface do Sistema 69

5 TESTES E ANÁLISE DOS RESULTADOS 71
5.1 Ambiente de Testes 71
5.2 Testes e Resultados 72
5.2.1 Fase 1 72
5.2.2 Fase 2 73
5.2.3 Fase 3 74
5.3 Análise dos Resultados 75

6 CONCLUSÕES E SUGESTÕES PARA FUTUROS TRABALHOS 78
6.1 CONCLUSÕES 78
6.2 SUGESTÕES PARA FUTUROS TRABALHOS 78

REFERÊNCIAS **80**

1 INTRODUÇÃO

O reconhecimento de face é um dos campos de pesquisa mais interessantes e importantes nas últimas duas décadas (CHAO, 2011). Grandes empresas estão na corrida para ver quem desenvolve o melhor algoritmo de reconhecimento de faces, e apenas à pouco tempo, Google e Baidu conseguiram taxas de erros abaixo das apresentadas por seres humanos (que se acerca aos 8%) (PORTAL, 2017).

As possibilidades de uso de um sistema de reconhecimento de faces são vastas (desde o uso comercial como empresas que querem identificar seus clientes e oferecer um serviço personalizado ao uso da polícia para o combate criminal e forense). Chega a ser um assunto polêmico em alguns círculos pois podem ser considerados invasão de privacidade e privação da liberdade como se conhece.

As pesquisas sobre o assunto envolvem conhecimentos de disciplinas como neurociência, psicologia, computação visual, reconhecimento de padrões, processamento de imagens, matemática avançada (CHAO, 2011), por isso apresenta um grande problema computacional. Além do problema de manipulação de imagens, de identificação e reconhecimento das faces, deve-se considerar problemas de persistência destas, a questão de treinamento das faces, velocidade de verificação, de extração de dados de imagens e vídeos e desempenho em geral.

1.1 BREVE HISTÓRIA

O reconhecimento automático de faces é um conceito relativamente novo. Desenvolvido na década de 60, o primeiro sistema semi-automático requeria que um administrador do sistema localizasse pontos da imagem como olhos, boca, nariz, etc, e depois um algoritmos comparava as distâncias entre estes pontos em diferentes fotos para dar um resultado de comparação (SMITH ARUN ROSS, 2009). Nos anos 80, algoritmos simples foram criados para automatizar este processo. Em 1988, Kirby e Sirovich aplicaram Análise de Componente Principal, uma técnica padrão de estatística, para o problema de reconhecimento de face. Este foi considerado um marco, pois mostrou que menos que 100 valores são requeridos para codificar uma imagem de uma face normalizada e bem posicionada (SMITH ARUN ROSS, 2009). Em 1991, Turk e Pentland descobriram que o erro residual das comparações feitas com a técnica *eigenfaces* poderiam ser usadas para detectar faces em uma imagem - uma descoberta que permitiu detecção em tempo-real.

Apesar desta abordagem ter limitações em considerando a posição da face, qualidade da imagem e fatores de ambiente, ela criou um significante interesse no desenvolvimento de tecnologias automáticas para reconhecimento (SMITH ARUN ROSS, 2009). A tecnologia capturou a atenção da mídia e do público em janeiro de 2001 no evento *Super-Bowl*, que capturou rostos das imagens das câmeras de vigilância e comparou com fotos

digitais "3x4" de uma base de dados apresentando rostos similares (SMITH ARUN ROSS, 2009).

Hoje, o reconhecimento de faces está sendo usado no combate à fraude de passaportes e cédulas de identidade onde a foto é padronizada com ambiente controlado, aplicativos de celulares e redes sociais para entretenimento, enquanto que grandes governos e organizações patrocinam, incentivam e desafiam empresas para conseguir o algoritmo ideal de reconhecimento (INTRONA, 2010).

1.2 PROBLEMÁTICA

A face é um objeto tridimensional (3D) iluminado por fontes de luz e envolvida por um fundo (*background*) com "dados arbitrários" (inclusive outras faces). Portanto, a aparência que uma face possui quando projetada para um plano bidimensional (2D) pode variar tremendamente (JEBARA, 1995). Na verdade, problemas de iluminação e de segmentação tem sido questões pertinentes no campo da computação gráfica e visual como um todo (JEBARA, 1995). Sendo assim o primeiro problema a enfrentar para o reconhecimento de faces seria a aquisição e processamento da imagem para que se possa detectar a face. As seções seguinte irão descrever os procedimentos com mais detalhes.

1.2.1 Detecção e Aquisição da Face

Em geral, em uma imagem retirada de um vídeo digital de nível amador, o módulo de identificação da face deve encontrar condições luminosas descontroladas, alta variação de poses, maquiagem, mudanças nos pelos faciais, adoecimento, envelhecimento, oclusões da face por interferência, roupa ou cabelo, enfim, uma incontável gama de variáveis (JEBARA, 1995). De fato, há diversos desafios e fatores chaves que podem significantemente impactar a performance da identificação e reconhecimento da face e os pontos de verificação (*matching scores*).

Na Figura 1, lista-se exemplos de alguns destes desafios em imagens, respectivamente:

(a) Variação de iluminação;

(b) Variação de poses ou pontos de visão;

(c) Envelhecimento;

(d) Expressões faciais e estilo da face (pelos faciais ou maquiagem);

(e) Oclusão.

A detecção contínua de faces em vídeos (diz-se *tracking* ou rastreamento) segue a mesma lógica e tem o mesmo problema de aquisições de faces estáticas. O rastreamento

FIGURA 1 – Exemplos de Variação

FONTE: *The Reseachgate.net*

nada mais é do que a detecção contínua da face em *frames* advindos de um vídeo, com um forte problema adicional de que deve-se manter a usabilidade do sistema em um computador contemporâneo de baixa a média desempenho. Em outras palavras, o processamento envolvido deve ser eficiente o bastante considerando o tempo de execução do vídeo (*frames* por segundo) e do sistema e tempo de armazenamento e consulta de dados.

O desempenho da detecção da face é uma questão-chave no processo de reconhecimento. A detecção de faces pode ser considerada um caso específico da área de detecção de objetos, alcançando resultados precidos (ELECTRICAL; IEEE, 1997).

Em um trabalho feito em 2003 por Kim Joon Hyung Shim (2003) na Universidade de Stanford - Califórnia, utilizou-se a técnica *EigenFaces* para detecção, acrescido de algoritmo de identificação de gênero, com um *Pentium 3* de 700 Mhz e menos de 500 MB de memória, alcançaram resultados que superam 93% de taxa de sucesso nas condições mais

adversas, considerando iluminação e um número grande de faces contidas nas imagens. Outros *benchmarks* mais recentes (2015) apresentam resultados que superam os 97,2%, como apresentado por (YANG PING LUO, 2015).

O problema com estas técnicas relativamente antigas, é que também reconhecem fotos de fotos, ou desenhos de faces como legítimas faces, por vezes até com pontuação bastante para uma verificação de sucesso com uma face real. Outro problema é que estas técnicas não funcionam com qualquer outro ângulo que não seja o frontal.

Novas técnicas de detecção não-frontais estão sendo implementadas, bem como as de modelagem sub-espacial 3D e comparações com reconhecimento de padrões baseados em aprendizado de máquinas e redes neurais, são fundamentos para os sistemas de reconhecimento de face avançados, capazes de reconhecer não só a face, mas também a estrutura cranial do alvo (BARBU NATHAN LAY, 2015).

Após a face ser identificada e localizada na imagem, para que sirva ao processo de reconhecimento, deve-se recortar esta face da imagem e realizar algumas operações gráficas sobre a mesma. Nesta fase são consideradas atividades de corte da face na região localizada, escalamento e correção de rotação, transformar em escala de cinza, normalizar a imagem para minimizar as condições de ambiente da foto, deixando a face pronta para a próxima fase do processo: o treinamento.

1.2.2 Treinamento e Reconhecimento da Face

Esta fase consiste em manipular a face recortada da imagem, normalizada e tratada de forma a extrair informações e características desta para que se possa salvar de alguma forma e relacionar estas características com a pessoa ou alvo. Nesta fase é importante aquisitar fotos da mesma face mostrando diferentes expressões faciais e situações de lumiosidade e posição.

Existem várias técnicas e algoritmos de reconhecimento, como (WOJCIK KONRAD GROMASZEK, 2016):

- reconhecimento baseado em redes neurais;
- reconhecimento baseado em processamento 3D;
- reconhecimento baseado em descritores de face;
- reconhecimento baseado em reconstrução;
- reconhecimentos tradicionais ou clássicos, dentre outros.

Este trabalho se focará no *EigenFaces*, um algoritmo clássico, famoso por ser pioneiro e objeto de muito estudo, teste e documentação. Os algoritmos convencionais podem ser divididos em duas categorias: caracterização holística ou linear, sendo que *EigenFaces* faz parte da primeira, que por sua vez faz parte de outra subdivisão de métodos

de projeção linear chamada Análise de Componentes Principais - **ACP** (ou no inglês, *Principal Component Analysis* – PCA) (WOJCIK KONRAD GROMASZEK, 2016).

Basicamente, o método ACP consiste em tratar a imagem de uma forma uniforme, coletar características da mesma transformando-as em valores numéricos e disponibilizá-las em espaço, que pode ter duas ou mais dimensões. Este processo visa destacar discrepâncias da face, ou seja, padrões de mudança de contraste, de relevo ou sombreamentos, ou diferença de cores. A transformação destes valores manipulados de volta em imagens, cria os chamados *EigenFaces* (Faces de fantasmas, em Alemão), como mostra a Figura 2 (DAVISON, 2013).

FIGURA 2 – Exemplos de EigenFaces

FONTE: (DAVISON, 2013)

O objetivo é que uma imagem de treinamento possa ser decomposta em uma soma de múltiplas *eigenfaces*, medidas e disponibilizadas em uma sequência especial.

Uma maneira mais simples de representar o relacionamento entre *eigenfaces* e imagens das faces é que estas faces são disponibilizadas em um espaço multidimensional onde os eixos do plano são as *eigenfaces* (DAVISON, 2013), como ilustra a Figura 3.

Seguindo o proceso de reconhecimento, tendo a nova face representada em valores e disponibilizada em um espaço multidimensional, resta a verificação ou reconhecimento desta nova face.

Para o reconhecimento da face propriamente dita, o mesmo processo do treinamento deve ser repetido e a face representada e disponibilizada no mesmo espaço multidimensional das faces treinadas (DAVISON, 2013). A distância entre as duas faces representada no espaço multidimensional é medida, e basicamente, quanto menor a distância, maior é a probabilidade das duas imagens representarem a mesma face, ou seja, uma distância zero entre as duas faces representadas no plano seria uma correspondência perfeita. A Figura 3 ilustra esta descrição:

FIGURA 3 – Exemplos de EigenFaces
FONTE: Adaptado de (DAVISON, 2013)

Este espaço multidimensional, na prática, pode ser um arquivo binário ou texto, comumente chamado tecnicamente de "*bundle*" (em português, agrupamento).

1.3 OBJETIVOS

Este projeto propõe o desenvolvimento de um Sistema de Reconhecimento de Faces em Vídeo utilizando o *EigenFaces*.

O algoritmo *EigenFaces* será implementado seguindo as instruções e exemplos de código de Davison (2013).

1.3.1 Objetivos Específicos

O desenvolvimento do sistema se fará na exploração das seguintes fases:

- Análise das bibliotecas e suas versões para melhor escolha;
- Preparação do ambiente;
- Implementação da interace e manipulação de imagens (e vídeos) utilizando componentes Java e as bibliotecas gráficas (para recorte de *frames* e imagens, aplicação de filtros, mineração de dados da imagem, manipulação de cores, iluminação, normalização, etc);
- Implementação do algoritmo *EigenFaces*;
- Detecção de face em imagens provenientes de câmera;

- Treinamento / reconhecimento de face na imagem;
- Incrementar a implementação feita acima para o treinamento e reconhecimento em vídeo tirado da câmera em tempo real;
- Realizar uma bateria de testes, e comparar resultados.

O objetivo final é criar um sistema de reconhecimento de faces em vídeo, analisando seus resultados e, finalmente, disponibilizando todo o trabalho e o código aberto ao público desenvolvedor e acadêmico.

1.4 JUSTIFICATIVA

O reconhecimento de faces é umas das tecnologias mais estudadas e complexas de hoje em dia. As maiores empresas e governos do planeta estão na corrida para se acercarem aos 100% de acerto em seus reconhecimentos e governos e organizações como FBI (*Federal Bureau of Investigation*) e NSA (*National Security Agency*) investem e desafiam os produtores desta tecnologia (SMITH ARUN ROSS, 2009).

Consta na Tabela 1 algumas das empresas que produzem tecnologias de reconhecimento:

As pioneiras estão avançando exponencialmente no assunto e empresas como Google e Baidu já conseguem taxas de erros menores do que as alcançadas por seres humanos, com rapidez de verificação sem precedentes, como ilustra o Gráfico 1.

GRÁFICO 1 – Comparação de sistemas de reconhecimento de faces em percentagem (%) de erros

FONTE: Adaptado de (PORTAL, 2017)

A Gráfico 1 mostra que a tecnologia da empresa chinesa Baidu está liderando a corrida com apenas 0.23% de taxa de erro, contra 0.37% da empresa americana Google (PORTAL, 2017). Ambas já conseguiram ultrapassar as taxas alcançadas por seres humanos que é de 0.8% de erros. Como mostra Portal (2017), os humanos, em geral, não são uma fonte 100% precisa para reconhecimento de faces. Humanos podem facilmente

confundir pessoas, ou não reconhecê-las pela idade ou por mudança de aparência ou estilo, ou até esquecê-las.

No final de 2016 a Baidu implantou seu sistema em sua histórica "Cidade de Água", substituindo totalmente o sistema de cartões e *tickets* com 99% de sucesso, e já estão no processo de implantação em outros parques temáticos (VERGE.COM, 2016). Policiais chineses estão usando óculos de reconhecimento que verificam, instantaneamente, as faces de turistas e procurados. Com os armazenamentos e reconhecimentos dominando aplicativos de celulares e redes sociais, e a polícia com grande interesse forense no assunto, e outras infinitas possibilidades, obviamente esta tecnologia fará parte do dia a dia do nosso futuro. Sendo assim é necessário estudá-la, entendê-la e desenvolvê-la.

TABELA 1 – Empresas que produzem tecnologias de reconhecimento de faces

Empresas	Local	website
Acsys Biometrics Corp.	Burlington, Canada	http://www.acsysbiometrics.com
Animetrics, Inc.	Conway, NH	http://www.animetrics.com
Asia Software Ltd.	Almaty, Kazakhstan	http://www.asia-soft.com/frs/en/main
Aurora Computer Services Ltd.	Northampton, United Kingdom	http://www.facerec.com
Bioscrypt [Acquired by L-1 Identity Solutions in 2008; Bioscrypt acquired A4 Vision in 2007]	Toronto, Canada	http://www.bioscrypt.com
C-VIS Computer Vision und Automation GmbH [Acquired by Cross Match Technologies]	Palm Beach Gardens, FL	http://www.crossmatch.com
Carnegie Mellon University	Pittsburgh, PA	http://www.ri.cmu.edu/labs/lab_51.html
Cognitec Systems GmbH	Dresden, Germany	http://www.cognitec-systems.de
Cybula Ltd.	York, United Kingdom	http://www.cybula.com
Diamond Information Systems (DIS)	Beijing, China	http://www.disllc.net
FaceKey Corp.	San Antonio, TX	http://www.facekey.com
Neurotechnology [Formerly Neurotechnologija]	Vilnius, Lithuania	http://www.neurotechnology.com
Neven Vision [Acquired by Google in 2006; Formerly Eyematic Interfaces, Inc.]	Mountain View, CA	http://www.google.com/corporate
New Jersey Institute of Technology (NJIT)	Newark, NJ	http://www.cs.njit.edu/liu/facial recognition VPlab/index.html
Nivis, LLC	Atlanta, GA	http://www.nivis.com
Old Dominion University	Norfolk, VA	http://www.lions.odu.edu/org/vlsi/demo/vips.htm
OmniPerception Ltd.	Surrey, United Kingdom	http://www.omniperception.com
Omron	Kyoto, Japan	http://www.omron.com/r_d/coretech/vision
Panvista Limited	Sunderland, United Kingdom	http://www.panvista.co.uk

FONTE: Adaptado de Smith Arun Ross (2009)

2 REVISÃO BIBLIOGRÁFICA

Vários trabalhos serviram como fonte de conhecimento e suporte para o entendimento e desenvolvimento deste tema. Alguns foram considerados mais importantes por abordar detalhadamente o problema de reconhecimento, outros foram buscados para se entender melhor, ou ter outra visão sobre alguns detalhes específicos do problema.

Trabalhos de autores como Silva (2009) ajudaram a estruturar e organizar o tema, por abordar tópicos similares e por possuir riqueza de detalhes no assunto de análise de componentes apresentando uma implementação em *MatLab* (um *software* interativo voltado para o cálculo numérico).

O trabalho de Balan (2009) foi utilizado para se aprofundar no tema de imagens digitais, assim como Gonzalez (2010) que também esclareceu o funcionamento de escalonamento, normalização e monocromatização de imagens. O autor Dr. Andrew Davison, em seu livro, Davison (2013) se destacou pela ótima didática, facilitando o entendimento de Análise de Componentes e sua relação com *EigenFaces*.

Os trabalhos Smith (2002) e Kitani (2018) foram estudados para descrever e implementar algoritmo ACP (ou *Eigenfaces*).

Nas próximas seções serão explanados os conteúdos destes e de outros trabalhos.

2.1 CONCEITOS DE AQUISIÇÃO E PROCESSAMENTO DE IMAGENS

Com a abrangência dos sistemas de comunicação, com difusão de conhecimentos e informações por diversos meios, captar, armazenar e processar imagens se tornou necessidade fundamental. Para o reconhecimento de faces em vídeo, o sucesso deste processo é pré-requisito para o funcionamento dos algoritmos.

As sessões a seguir contemplam fundamentos básicos sobre imagem, vídeo, alguns de seus processamentos necessários para a aplicação dos algoritmos de detecção e reconhecimento de faces.

2.1.1 Imagem Digital

A imagem digital são valores numéricos disponibilizados em uma matriz bidimensional. Basicamente, existem dois tipos de imagens: os chamados *rasters* e vetorizadas. A primeira são representações de cada ponto da imagem com alguma cor, usadas geralmente em fotografias, enquanto que o segundo é materializado por plotadores que recebem os pontos e as distâncias entre eles como parâmetros, considerando retas, curvas, polígonos e etc. Sendo assim, não perdem sua qualidade quando redimensionados (BALAN, 2009).

Converter uma imagem para o formato digital significa transferir os elementos que a compõem para elementos representativos de cada pequeno fragmento original. O menor

elemento da imagem, o *pixel*, é identificado segundo sua intensidade de nível de cinza e as cores correspondentes. Identificados, estes elementos são armazenados por códigos que podem ser reconhecidos pelo dispositivo de visualização e apresentados novamente por um dispositivo de visualização, como um monitor de vídeo ou impressora (BALAN, 2009).

Para que uma imagem analógica (representação real da cena) seja convertida, esta deve sofrer uma separação espacial (amostragem) e em amplitude (quantização) (BALAN, 2009).

A amostragem é a divisão do plano (x, y) em uma grade (ou matriz bi-dimensional) onde x e y serão números inteiros. Os pontos da matriz são denominados *pixels* (*PICTure Elements*) Cada pixel representa uma parte da cena real, desta forma a resolução espacial da imagem é proporcional aos valores de M e N correspondentes na matriz. Em geral, na malha de amostragem, o formato dos pixels (x, y) é retangular, mas pode também ser triangular ou mais complexa. Os valores de cada ponto da matriz, coluna x e linha y, que identificam um único pixel (M, N), devem ser escolhidos de forma a respeitar a relação qualidade da imagem *vs.* espaço de armazenamento, em função da aplicação para a qual a imagem se destina. Para uma imagem digital com 256 níveis de cinza, o número de bytes ocupados para armazenar a imagem é o produto da linha ou coluna da matriz e 8 bits (unidade mínima de armazenamento digital) (BALAN, 2009).

Matematicamente, toda imagem em escala de cinza é um função $f(x, y)$ da intensidade luminosa, em qualquer parte das coordenadas (x, y), proporcional ao brilho (tons de cinza) da imagem em um determinado ponto (GONZALEZ, 2010).

A função $f(x, y)$ é a multiplicação da iluminância $i(x, y)$ que é a quantidade de luz que incide sobre o objeto pela refletância $r(x, y)$ que é fração de luz incidente que o objeto vai refletir ao ponto (x, y).

Sendo assim, podemos dizer que :

$$f(x, y) = i(x, y) * r(x, y), \tag{2.1}$$

onde $0 < i(x, y)$ e $0 < r(x, y) < 1$.

Quando se utiliza uma imagem colorida, no padrão RGB por exemplo, deve se usar uma função $f(x, y)$ para cada banda, R (*Red*), G (*Green*) e B (*Blue*) que são as cores primárias (GONZALEZ, 2010).

2.1.2 Formato PNG

PNG (*Portable Network Graphics* ou "*PNG is Not f*") é um formato de representação de imagens do tipo *raster* e foi desenhado para substituir outros formatos em certa extensão. Uma das vantagens do PNG é o suporte de canal alfa (transparência) de forma eficiente. Além disso, o formato é livre (ROELOFS, 2017).

O trabalho da compressão é justamente retirar essas informações redundantes para diminuir o tamanho do arquivo. Por exemplo, dado um pixel qualquer de uma ima-

gem, possivelmente, a cor ou o valor desse pixel será igual a de vários outros elementos dentro da mesma imagem. Quando a imagem é comprimida, para excluir os pixels de valores iguais, é guardado apenas um valor desse pixel, que é reproduzido para os outros semelhantes, economizando tempo de carregamento (FARIAS, 2017). Existem outras técnicas que são utilizadas para tal processo.

2.1.3 Vídeo Digital

Um vídeo é uma sucessão de imagens apresentadas sequêncialmente em um determinado algoritmo. O olho humano pode distinguir cerca de 20 imagens por segundo. Deste modo, quando se mostram mais de 20 imagens por segundo, é possível enganar o olho e criar a ilusão de uma imagem em movimento.

A fluidez de um vídeo se caracteriza pelo número de imagens por segundo (frequência de quadros), expresso em FPS (*Frames per second*, em português, Quadros por segundo). O vídeo multimídia é geralmente acompanhado de som, ou seja, dados de áudio (SMITH, 2018).

2.1.4 Aquisição da Imagem e de Vídeo

No processo de aquisição de imagens, tradicionalmente, uma "cena" tridimensional (ou imagem analógica) deve ser capturada por um dispositivo eletrônico, que colhe uma amostragem convertendo-a para uma imagem digital de duas dimensões. Por exemplo, uma câmera digital captando uma cena 3D e convertêndo-a em 2D.

Atualmente o dispositivo de conversão mais usado é a câmera do tipo CCD (*charge coupled device*), que é uma matriz de células semi-condutoras fotossensíveis que trabalham com capacitadores, fazendo um armazenamento da carga elétrica proporcional à energia luminosa incidente (GONZALEZ, 2010).

A saída deste processo é a imagem do tipo *raster*, pronta pra ser formatada e compactada por alguma representação (PNG, JPG, BPM, etc).

Na aquisição de vídeos, o mesmo processo descrito acima para imagens é utilizado, com a diferença de que o vídeo é uma fluidez de imagens, sendo assim é necessário aplicar este processo iterativamente. Caso uma única imagem precise ser colhida do vídeo, basta escolher um quadro (descrito na Subseção 2.1.3) para o processo.

2.1.5 Processamento da Imagem

O processamento de imagem é todo o processo que tem uma imagem como entrada e saída.

Usa-se para melhorar o aspecto visual de certas feições estruturais para o analista humano ou para desempenho e para fornecer outros subsídios para a sua interpretação, inclusive gerando produtos que possam ser posteriormente submetidos a outros processamentos (EYMAR LAERCIO, 2018).

O processamento de imagens pode ser dividido em 3 fases básicas: pré-processamento, realce e classificação. Pré-processamento refere-se ao processamento inicial de dados brutos para calibração radiométrica da imagem, correção de distorções geométricas e remoção de ruído. Realce visa melhorar a qualidade da imagem, permitindo uma melhor discriminação dos objetos presentes na imagem. Na classificação são atribuídas classes aos objetos presentes na imagem (EYMAR LAERCIO, 2018).

Nas próximas seções, algumas técnicas de processamento de imagens serão abordadas pela sua importância e sua necessidade de uso em algoritmos de detecção e reconhecimento de faces. São elas:

- Transformação em Escala de Cinza: considerada uma técnica de pré-processamento, é usada para transformar a imagem para escala de cinza, tornando posteriores operações de processamento de imagens mais rápidas e eficientes;
- Escalonamento: pré-processamento de correção linear geométrica 2D (WANGENHEIM, 2017);
- Equalização: uma técnica de realce de contraste (GONZALEZ, 2010)
- Segmentação: recorte da imagem, onde se extraem os objetos relevantes para a aplicação desejada (EYMAR LAERCIO, 2018).
- Classificação: uso de classificadores para reconhecer padrões em imagens (DAVISON, 2013)

2.1.5.1 Conversão em Escala de Cinza

Em fotografia, computação e colometria, uma imagem em escala de cinza é uma imagem em que cada valor de pixel é uma única amostra da representação de intensidade de luz (JOHNSON, 2006). Na Figura 4 são apresentadas três dos vários métodos utilizados para conversão de imagens coloridas para escala de cinza para imagens do tipo *raster*: método Escolha de Canal, Máximo Canal e Conversão Clássica (NOGUEIRA, 2016).

No Método **Escolha de Canal**, tendo uma imagem do tipo *raster* RGB, uma das formas mais simples para convertê-la em níveis de cinza é escolhendo um dos três canais (*red*, *blue* ou *green*) para ser a imagem resultande, ou seja, a repetiçãao do canal escolhido nos outros dois canais mantendo assim a imagem com três matrizes (em RGB) como mostra a Figura 4.

No Método **Máximos Canal**, considerando uma imagem RBG, cada pixel "p" da matriz da imagem resultante será o de maior valor dentre os pixels da mesma posiçào "p" dos três canais do RGB. O resultado da conversão por este metodo é ilustrado na Figura 5.

FIGURA 4 – Decomposição dos canais do sistema RBG.
FONTE: (NOGUEIRA, 2016)

No Método de **Conversão Clássico** utiliza-se de "pesos"sobre os valores RGB para calcular a resultante "r" pela equação

$$r = 0.29R + 0.59G + 0.11B. \tag{2.2}$$

A Figura 6 ilustra o resultado do uso deste método.

Já o Método **Média de Canal** consiste em calcular a média dos valores RGB de cada pixel e atribuir o resultado ao pixel novamente para cada valor R, G e B, como mostra a Figura 7.

Converter a imagem em escala de cinza torna mais eficientes os subsequente processos, como equalização e aplicação de classificadores (DAVISON, 2013).

2.1.5.2 Escalonamento

No âmbito de Computação Gráfica e Imagens Digitais e referindo-se a imagens do tipo *raster* (matriciais), o escalonamento de imagens pode ser interpretado como uma reamostragem ou reconstrução da imagem (TECH-ALGORITHM, 2007). A imagem reconstruída pode ser maior ou menor dependendo do fator de escala. Quando uma imagem é diminuida de seu tamanho original, os dados da imagem (pixels) são descartados, e

FIGURA 5 – Conversão em tons de cinza pelo método máximo canal.
FONTE: Nogueira (2016)

FIGURA 6 – Conversão em tons de cinza pelo método de conversão clássico
FONTE: Nogueira (2016)

quanto o tamaho da imagem é aumentada, novos dados devem ser inseridos na imagem, como mosta a Figura 8.

A principal função de um algoritmo de escalonamento é de preencher estes espaços "vazios" com pixels de valores coerentes. Para tal utiliza-se técnicas matemáticas de interpolação com os dados da imagem. Interpolação é o processo de estimar valores intermediários de uma função ou sinal discreto amostrado em posições no espaço contínuo (MURTA, 2013).

Existem várias técnicas de interpolação, dentre elas:

- Interpolação do vizinho mais próximo;
- Interpolação Bilinear;
- Interpolação Bicúbica;

FIGURA 7 – Conversão em tons de cinza pelo método de média de canal.

FONTE: Nogueira (2016)

FIGURA 8 – Exemplo de escalonamento de imagem.

FONTE: Adaptado de Tech-Algorithm (2007)

- Interpolação Bicúbica Generalizada.

A interpolação do Vizinho Mais Próximo por exemplo, é a maneira mais simples de abordar a problemática da interpolação e consiste em determinar o pixel vizinho mais próximo e assumir um valor de intensidade à este (GIASSA, 2009), operado pelo arredondamento da coordenada x pelo inteiro mais próximo u_0 e usa a amostra como estimativa de $g(x)$, como ilustra a Figura 9.

FIGURA 9 – (a) Função ou sinal discreto (b) Função ou sinal interpolado

FONTE: Adaptado de (GIASSA, 2009)

2.1.5.3 Equalização de Histograma

O histograma de uma imagem digital é definido segundo Gonzalez (2010) como "uma função discreta $h(r_k) = n_k$ onde r_k é o k-ésimo valor de intensidade e n_k é o número de pixels da imagem com intensidade r_k e cujos níveis de intensidade desta imagem estejam no intervalo [0, L - 1]". Os Histogramas são a base para várias técnicas de processamento no domínio espacial, onde sua manipulação pode ser utilizada para realce de imagens, além de fornecer dados estatísticos a seu respeito (GONZALEZ, 2010).

Histograma pode ser visto como uma função de distribuição de frequência ou como uma função de distribuição de probabilidade e é, essencialmente, um vetor que armazena as ocorrências de cada tom de cinza presentes. Geralmente é mostrado como um gráfico. A equalização de histogramas é "uma operação que melhora o contraste, uniformizando o histograma de forma automática, distribuindo os níveis de cinza existentes e mapeando-os para novos níveis" (MELO, 2016). A Figura 10 ilustra uma imagem que passou por este processo.

FIGURA 10 – Exemplo de aplicação de equalização de histograma.

2.1.5.4 Segmentação

Em visão computacional, segmentação se refere ao processo de recortar uma imagem digital, ou seja, recolher um conjunto de pixels, para simplificar, mudar a representação de uma imagem ou extrair áreas relevantes para facilitar a sua análise.

Como resultado, cada um dos pixels em uma mesma região é similar com referência a alguma característica ou propriedade computacional, tais como cor, intensidade, textura ou continuidade. Este processo é pré requisito para que reconhecimentos de objetos tenham grandes chances de sucesso (GONZALEZ, 2010).

Na Figura 11 ilustra-se a aplicação dos processos acima descritos na problemática de reconhecimento de faces.

FIGURA 11 – Exemplo de aplicação de processamento de imagem. **A**: Segmentação **B**: Transformação de cores em escala de cinza. **C**: Escalonamento. **D**: Equalização de histograma

FONTE: Elaborado pelo autor.

2.1.5.5 Classificação

Segundo Velasquez (2017), classificação de imagem contextual é um tópico de reconhecimento de padrões em Visão Computacional. Chama-se "contextual" pois significa que esta abordagem é focada na proximidade com os pixels e sua relação com o "classificador", (contemplado na seção Subseção 2.2.1). É utilizado efetivamente na detecção de objetos e, também, pode ser aplicado para a detecção de faces em uma imagem.

2.2 DETECÇÃO DE FACES COM CLASSIFICADORES HAAR

Um classificador Haar pode ser treinado para detectar vários tipos de objetos rígidos, definidos, como carros, motos ou partes do corpo humano, como olhos ou boca,

com altas taxas de sucesso. Não é muito eficiente em reconhecer objetos com ramos, tipo arvores, mãos ou objetos camuflados ou contendo pouca textura, contorno e sub-regiões que variam em cor e iluminação (DAVISON, 2013).

Um classificador bem treinado pode envolver milhares de fotos de alta qualidade com imagens positivas. Para a detecção de faces, isto significa imagens de rostos tiradas de perto que devem ter posições similares com muito pouca variação de fundo (*background variation*). Olhos, bocas e narizes devem estar na mesma posição em todas as fotos, e estas devem ser do mesmo tamanho. Também é necessário treinar o classificador com um número similar de imagens negativas (isto é, sem rostos).

Existem diversas bibliotecas com classificadores pré-treinados para diferentes objetos, incluindo faces.

2.2.1 Os Características Haar (*Haar Features*)

Cacarcterísticas de haar, ou filtro haar, recebe esse nome pela sua similaridade com a *Haar wavelet*, como se pode observar na Figura 12 , que consiste em uma sequência de funções quadradas redimensionadas que formam uma familia de ondulações das mais simples possíveis.

FIGURA 12 – Um exemplo de *Haar wavelet*

FONTE: Adaptado de Gonzalez (2010)

Características Haar são, basicamente, imagens treinadas e usadas para achar objetos similares em outras imagens. Viola e Jones, em seu algoritmo *Haar Cascades* (contemplado na seção seguinte), por exemplo, utilizam classificadores retangulares. Cada classificador pode indicar a existência ou a ausência de uma característica em outra imagem. A maior motivação para o uso de características em um objeto ao invés do uso de pixel, é que a velocidade da análise de uma imagem baseada no conjunto de sua principais características é muito maior do que a análise baseada sobre seus pixels, devido ao fato do

número de características ser, substancialmente, inferior em relação ao número de pixels (MOREIRA, 2008).

A Figura 13 mostra um classificador Haar em ação. Estes padrões retangulares podem ser escalonados para que caracterísricas de diferentes tamanhos na imagem possam ser detectados usando a mesma abordagem.

FIGURA 13 – Características Haar representando características relevantes face

FONTE: OpenCV (2018)

2.2.2 Algoritmo Viola-Jones (*Haar Cascades Classifier*)

O *framework* de detecção de objetos Viola-Jones, apelidado em inglês de *Haar Cascades Classifier*, ou em português Cascata de Classificadores Haar (MOREIRA, 2008), foi o primeiro algoritmo de detecção a fornecer detecção de objetos com taxas de sucesso competitivas e em tempo real. Foi proposto em 2001 por Paul Viola e Michael Jones. Apesar de poder detectar uma variedade de classes de objetos, foi desenhada com o objetivo primordial de detectar faces.

Com a utilização das características Haar, o objetivo a ser alcançado é classificar corretamente um dado objeto a partir do conjunto de suas principais características.

De acordo com Davison (2013), o algoritmo funciona com o uso de integrais que são rápidas e eficientes, possibilitando ainda reduzir drasticamente o número de atributos que precisam ser testados para decidir se a imagem contem um objeto (por exemplo uma face). Os testes de atributos da imagem são organizados em "cascatas" também pode ser representado por uma árvore binária, representando o uso das integrais iterativas.

O nó-raiz da árvore binária deve contér um valor representando o teste que se provou ser o melhor em encontrar um objeto durante o treino. Se uma imagem não é rejeitada por este valor teste, então a imagem segue para o nó com o segundo melhor valor de teste, e assim sucessivamente. Apenas se a imagem alcançar o fim de todos os nós

(ou fim da cascata ou da árvore) sem ser rejeitada, a imagem certamente deverá conter o objeto característico.

O grande ponto negativo é que este algoritmo também detecta os objetos que não são realmente os objetos que se tinha a intenção de detectar (DAVISON, 2013). Por exemplo, o desenho de um rosto certamente vai ser detectado com uma face, e este desenho não precisa ser muito bem elaborado. As vezes uma disposição de sombras e regiões luminosas aleatórias podem ser detectadas como objetos característicos .

Segundo Moreira (2008), os estágios dentro de uma cascata são criados através da combinação de funções de classificadores Haar previamente treinados. O principal objetivo do uso de cascatas é fazer com que seus estágios iniciais descartem um grande numero de regiões que **não contém** o objetivo desejado, e estágios mais avançados sejam melhores em evitar um falso positivo na região analisada.

A Figura 14 representa os N estágios de uma cascata de classificadores. Cada um deles deve descartar ao máximo o número de regiões da imagem que não contém o objeto, a fim de diminuir a quantidade de processamento de outras sub-áreas da imagem original (MOREIRA, 2008).

FIGURA 14 – Cascata de classificadores com estágios (C0, C1, ..., Cn)

.

FONTE: Moreira (2008)

O processo para geração da cascata é guiado por um conjunto de metas de detecção e de desempenho. Na prática, uma arquitetura muito simples pode ser usada para produzir uma eficiente cascata de classificadores. O usuário pode informar as taxas mínimas aceitáveis e cada estágio da cascata é treinada junto ao conjunto de características Haar contando com cada vez mais elementos, até que se atinjam as taxas de detecção de sucesso e falsos positivos. As taxas são encontradas testando o detector corrente sobre um conjunto de validação. Se a taxa de falsos positivos ainda não for atendida, então um novo estágio subsequente é ativado, coletando todas as falsas detecções encontradas (MOREIRA, 2008).

Para que o algoritmo possa detectar o objeto de interesse, é necessário que a imagem seja percorrida diversas vezes em várias direções. Assim a cada iteração é analisada uma região de tamanho diferente, chamada janela de busca (MOREIRA, 2008).

O usuário pode definir o tamanho desta janela que deve ser maior ou igual ao tamanho das imagens que foram treinadas na cascata, ou seja, se a cascata foi treinada com imagens positivas de 24x24 pixels, o tamanho mínimo da janela de busca deve seguir a mesma resolução (MOREIRA, 2008). As janelas devem funcionar em *threads*, para que n janelas se desloquem no eixo x da imagem e m janelas no eixo y. A quantidade de janelas deve ser o suficiente para ocupar todo o espaço da imagem.

Outro valor que o usuário pode informar é o número de detecções subsequentes necessários para que a região seja considerada a ter o objeto alvo. Reduzir este valor pode aumentar a velocidade de processamento, mas também aumenta as chances de taxas de falsos positivos (DAVISON, 2013).

O resultado do algoritmo são as posições dos objetos classificados com as características, podendo ser identificados na imagem.

2.3 PROCESSO DA ANÁLISE DE COMPONENTES PRINCIPAIS (ACP) OU *EIGENFACES*

A técnica ACP (Análise de Componentes Principais) foi descrita inicialmente por Karl Pearson em 1901 como solução para alguns problemas biométricos da época, mas ele não propôs nenhuma implementação dessa técnica para um caso com mais de duas variáveis. Em 1933 essa técnica foi descrita por Hotteling, mas os cálculos ainda eram complexos quando se tratava de um problema com dezenas de variáveis. Apenas com o surgimento dos computadores atuais, a técnica ficou mais difundida (SILVA, 2009).

O processo de reconhecimento de faces depende de uma fase de treinamento precedente envolvendo imagens faciais com suas identidades associadas. Imagens típicas de treinamento são mostradas na Figura 15.

É importante que as imagens sejam todas colhidas e orientadas de maneira similar, para que as variações entre as imagens sejam causadas por diferenças entre as faces e não por diferenças em plano de fundo ou posição facial. Também deve haver uniformidade na posição, clareamento, tamanho e resolução da imagem. É recomendado incluir várias imagens da mesma face mostrando diferentes expressões, como sorrindo ou abrindo a boca. A relação da imagem com o indivíduo pode ser feita de diversas maneiras, uma delas é codificar alguma chave única para o indivíduo junto ao nome da imagem (DAVISON, 2013).

O processo de treinamento cria as *eigenfaces*, como mostra a Figura 2 explicada na Subseção 1.2.2, que são composições das imagens de treinamento que têm a propriedade de acentuar elementos que podem ser distinguidos mais facilmente pelo algoritmo. Uma imagem de face pode gerar várias *eigenfaces*. As imagens de treino da face podem ser representadas por uma sequência de "pesos" (p1, p2, p3, p4, p5, p6), como ilustra a Figura 16.

O objetivo é que a imagem de treinamento pode ser decomposta por uma soma

FIGURA 15 – Exemplo de imagens típicas de treinamento.

FONTE: elaborado pelo autor

dos pesos das multiplas *eigenfaces*, sendo todos estes pesos salvos como uma sequência.

Nem todas as *EigenFaces* são igualmente importantes - algumas podem representar elementos faciais mais importantes para distinguir faces. Isto quer dizer que não é necessário utilizar todas as *eigenfaces* para reconhecer uma face, o que permite que uma imagem seja representada por uma pequena quantidade de *eigenfaces*, o que ocupa menos espaço e torna a execução do algoritmo mais rápida. O ponto negativo é que com menos pesos, a probabilidade de se perder importantes elementos faciais também aumenta (DAVISON, 2013).

Outra maneira de se entender a relação entre as *eigenfaces* e as imagens é representa-las em um espaço cartesiano multidimensional (ou um *eigenspace*), como mostra a Figura 17.

Os pesos agora podem ser observados como as coordenadas da imagem de treinamento em um espaço cartesiano multidimensional (*eigenspace*). Na Figura 17, existem apenas 3 eixos representando as *eigenfaces* p1, p2 e p3, porém se houvessem 10 *EigenFaces*, haveriam 10 eixos neste espaço.

Após o processo de treinamento das faces vem o processo de reconhecimento. Neste novo processo, uma nova imagem para reconhecimento deve ser decomposta em *eigenfaces*. A sequência de pesos resultante é comparada com cada sequência de pesos que foram previamente disponibilizadas no espaço multidimensional, e o nome associado com o as características mais próximas dos *eigenfaces* treinados é associado (DAVISON, 2013).

FIGURA 16 – Exemplo de uma imagem de treinamento representada em uma sequência de pesos de engenfaces.

FONTE: Adapdato de Davison (2013)

Em outras palavras, em termos de espaço multidimensional, a nova imagem é posicionada no plano cartesiano com suas próprias coordenadas (pesos). Então uma técnica de medida de distância (geralmente a distância Euclidiana, que será abordada na Subseção 2.3.1.2) é utilizada para encontrar o vetor de valores correspondente mais próximo, sendo assim, uma distância zero seria uma correspondência (reconhecimento) perfeita (DAVISON, 2013). A Figura 3, na Subseção 1.2.2, ilustra este processo.

2.3.1 O Cálculo da Análise de Componentes Principais (ACP)

Segundo Silva (2009), o algoritmo da ACP pode ser dividido em duas fases: a fase de treinamento e a de reconhecimento. Cada fase consiste na realização de processos, descitos nas próximas seções.

2.3.1.1 Fase de Treinamento

Esta fase consistena execução de 5 passos:

Passo 1 - Preparação dos dados: neste passo deve-se preparar os dados das imagens aquisitadas nas condições descritas na seção Seção 2.3 e disponibilizá-las na forma de vetores de *bytes* para serem manipulados. A Tabela 2 exemplifica 2 conjuntos (vetores) de dados que poderiam ser utilizados: o vetor x e y.

FIGURA 17 – Exemplo de eigenfaces representadas em um plano cartesiano (*eigenspace*) de 3 dimensões.

FONTE: Adapdato de Davison (2013)

Na Figura 18 plota-se os valores correspondentes dos vetores x e y como pontos em um gráfico (DAVISON, 2013).

FIGURA 18 – Plotagem dos valores de x e y.

FONTE: Adapdato de Davison (2013)

Passo 2 - Subtração da Face Média: neste passo deve-se calcular a face média

TABELA 2 – Dados dos vetores x e y.

x	y
2.5	2.4
0.5	0.7
2.2	2.9
1.9	2.2
3.1	3.0
2.3	2.7
2	1.6
1	1.1
1.5	1.6
1.1	0.9

FONTE: Davison (2013)

do conjunto de dados definidos no "Passo 1". A face média (Ψ) é definida por Silva (2009):

$$\Psi = \frac{1}{n}\sum_{1}^{n} X_n \tag{2.3}$$

onde X_n são os conjuntos de dados do "passo 1", sendo n a contagem desses conjuntos. Com a face média calculada (Ψ), deve-se subtraí-la do conjunto de dados (X_n):

$$\Phi_n = X_n - \Psi, \tag{2.4}$$

onde Φ_n é o vetor resultante, tendendo a obter características distintas das imagens.

Passo 3 - Cálculo da matriz de covariância: a covariância é uma medida estatística, generalizada da variância, que compara dois conjuntos de dados. Neste passo deve-se calcular a matriz de covariância para quantificar as diferenças **entre** os conjutos de dados (DAVISON, 2013). Realiza-se o cálculo atravez da formula (SILVA, 2009):

$$C = \Phi^T.\Phi, \tag{2.5}$$

onde Φ é o vetor resultante do "passo 2", Φ^T é seu equivalente transposta e C é a matriz de covariância resultante (SILVA, 2009).

Se a fórmula da covariância for aplicada nos dados dos vetores x e y dados na Tabela 2, teriam como resultado uma matriz de covariância de dimensões 2x2:

$$\begin{pmatrix} 0.6166 & 0.6154 \\ 0.6154 & 0.7166 \end{pmatrix},$$

Passo 4 - Cálculo dos autovetores e autovalores: para que as *eigenfaces* sejam geradas, deve-se decompor a matriz de covariância em outros conjuntos de dados chamados de autovetores (ou *eigenvectors*) e autovalores (ou *eigenvalues*) (SILVA, 2009).

Segundo Davison (2013), o autovetor é um vetor de valores ordinário que quando multiplicado por uma dada matriz, muda de magnitude, enquanto que autovalor é o valor

desta magnitude. Calcula-se os autovalores atraves de

$$|\Phi - \lambda I| = 0, \tag{2.6}$$

onde Φ é o vetor resultante do "passo 2", I é a matriz identidade, e λ é o autovalor (SILVA, 2009). Calcula-se os autovetores atraves de

$$\Phi e = \lambda e, \tag{2.7}$$

onde Φ é o vetor resultante do "passo 2", e é o autovetor e λ é o autovalor (SILVA, 2009). Neste passo também ordena-se os autovetores por ordem de autovalores (ou seja, por ordem de significância) para facilitar possível futuro descarte (SILVA, 2009).

Pode-se observar que o resultado da decomposição da matriz de covariância gereda acima no exempo, como é uma matriz 2x2, possui dois autovetores e dois autovalores:

$$\begin{pmatrix} -0.7352 \\ 0.6779 \end{pmatrix} \text{ e } 0.049,$$
$$\begin{pmatrix} 0.6779 \\ 0.7352 \end{pmatrix} \text{ e } 1.284,$$

Ambos os autovetores possuem tamanho unitário.

Passo 5 - Projeção dos autovetores ao espaço multidimensional: neste último passo de treinamento, cada imagem (ou conjunto de dados) de treinamento é projetado no espaço multidimensional de autovetores. O descritor ACP é calculado por uma combinação linear de autovetores com os vetores originais (SILVA, 2009):

$$W_n = {e_n}^T (X - \Phi), \tag{2.8}$$

onde e são os autovetores, X é o vetor original de faces de treinamento, Φ é a face média e W seria o espaço multidimensional resultante.

Observando novamente a plotagem dos vetores x e y contemplados na Figura 18, se a média destes conjuntos de dados ($\bar{x}$ e $\bar{y}$) for subtraído de seus valores, estes dados então seriam normalizados. Isto também quer dizer que as linhas que representam os vetores x e y são transladados para o centro como mostra a Figura 19.

Os dois autovetores pode ser adicionados ao gráfico como linhas convertendo os vetores em equações. $\begin{pmatrix} -0.7352 \\ 0.6779 \end{pmatrix}$ torna-se $y = \frac{-0.7352}{0.6779}x$ e $\begin{pmatrix} 0.6779 \\ 0.7352 \end{pmatrix}$ torna-se $y = \frac{0.6779}{0.7352}x$.

Os dados normalizados e as duas equações são ilustradas na Figura 19 como linhas azuis.

Observa-se que a Figura 19 demonstra como os autovetores acentuam as relações entre os conjuntos de dados indicando como estes se espalham nas coordenadas do espaço multidimensional. Este espalhamento facilita a distinção dos dados (DAVISON, 2013).

Dos dois autovetores marcados em azul, a linha $\frac{0.6779}{0.7352}x$ é a mais útil, pois os dados estão mais espalhados acerca do trajeto desta linha. Isto pode ser confirmado numericamente observando o autovalor associado com seu autovetor: o autovalor (1.284) é um melhor indicador de espalhamento pois é o maior dentre os dois autovalores.

FIGURA 19 – Plotagem normalizada dos dados de x e y com eigenvetores.

FONTE: Davison (2013)

A outra linha, $\frac{-0.7352}{0.6779}x$, contribui com a informação demonstrando como os dados são espaçados para direita ou esquerda do autovetor principal (a linha $\frac{0.6779}{0.7352}x$). Entretanto, trata-se de um componente menos importante no espalhamento de dados, assim como indica seu autovalor (0.049) (DAVISON, 2013).

O autovetor com o maior autovalor (no caso a linha $\frac{0.6779}{0.7352}x$, ou o vetor $\begin{pmatrix} 0.6779 \\ 0.7352 \end{pmatrix}$) é chamado de componente principal do conjunto de dados. Isto é, os autovetores e autovalores são efetivamente utilizados para performar uma análise de componente principal nos conjuntos de dados (DAVISON, 2013).

Todos os autovetores extraídos da matriz são perpendiculares. Isto significa que é possível rotacionar (talvez refletir) os dados para que os autovetores tornem-se alinhados com os eixos. Se este componente principal $\begin{pmatrix} 0.6779 \\ 0.7352 \end{pmatrix}$ for rotacionado e refletido para que seja alinhado com o eixo y, resulta-se em uma plotagem mostrada na Figura 20:

FIGURA 20 – Versão rotacionada e refletida da Figura 19

FONTE: Davison (2013)

Observando a Figura 20, o ponto mais próximo da origem se situa na coordenada (0.13, 0.27). Pode-se afirmar também que este ponto tem uma sequência de pesos relativos

a cada autovetor, no caso, a sequência 0.13, 0.27.

Tanto a denotação de coordenadas quanto a de pesos pode ser usada para representar as *eigenfaces* - como contemplado anteriormente. Na Figura 16 foi dado o exemplo em que uma imagem é uma sequência de pesos de seis *eigenfaces*, enquanto que, alternativamente, na Figura 17 é representada em coordenadas de um espaço de 3 dimensões, sendo que cada eixo é definido por uma *eigenface* (ou autovetor) (DAVISON, 2013).

2.3.1.2 O Reconhecimento

O processo de reconhecimento pode ser dividido em 3 passos:

Passo 1 - Preparação dos dados: converter a imagem da nova face em um vetor de consulta Q (SILVA, 2009).

Passo 2 - Projeção do novo vetor no espaço: neste passo realiza-se a combinação linear de autovetores com o novo vetor de consulta Q por (SILVA, 2009):

$$W_n = e_n^T(Q - \Phi), \tag{2.9}$$

sendo e os autovetores, Q o vetor de consulta, e Φ a face média.

O terceiro passo é comparar a distância entre o descitor do vetor de consulta com cada um dos descitores armazenados na base de dados (SILVA, 2009).

Existem algumas técnicas de medidas de distância, dentre elas, há uma simples chamada Distância Euclidiana. Esta medida nada mais é do que a distância mínima entre dois pontos no espaço (uma linha reta). A distância euclidiana entre os pontos $P = (p_1, p_2, ..., p_n)$ $Q = (q_1, q_2, ..., q_n)$ é definida como:

$$\sqrt{(p_1 - q_1)^2 + (p_1 - q_1)^2 + ... + (p_n - q_n)^2} = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2} \tag{2.10}$$

O resultado desta medida é utilizada para calcular a taxa de sucesso de uma comparação. Sendo assim, uma distância nula representaria uma equivalência perfeita.

Neste capítulo foram apresentados os conceitos necessários para o entendimento de todo o processo de reconhecimento de faces, constituído de outros sub processos como aquisição de imagem e vídeo, processamento e classificação de imagens, detecção da face, treinamento e, por fim, o reconhecimento efetivo da face.

Os algoritmos de detecção de face Viola-Jones e o de reconhecimento de faces EigenFaces que utiliza o método matemático ACP (Análise de Componentes Principais), foram detalhados na ordem sequêncial de seus processos utilizando exemplos.

O próximo capítulo abordará o plano de materiais e métodos elaborado a partir do conhecimento adquirido no Capítulo 2.

3 MATERIAIS E MÉTODOS

Após os estudos realizados sobre processamento e classificação de imagens, sobre o algoritmo de detecção *Haar-Cascades* e o de reconhecimanto *EigenFaces* contemplado no capítulo 2, é necessário definir os materiais e os meios para desenvolver o sistema aqui proposto.

Este capítulo apresenta os materiais e metodos que serão utilizados para o desenvolvimento do sistema proposto.

3.1 MATERIAIS

As seguintes seções detalham os equipamentos que serão usados para o desenvolvimento do sistema e as tecnologias de software escolhidas baseado na revisão bibliográfica realizada.

3.1.1 Hardwares

Lista-se a seguir, os equipamentos necessários para a realização deste trabalho:

- Câmera digital do tipo *Webcam* de 720p de resolução (ou 1280x720 *pixels*);
  - utilizada para captar o vídeo e disponibilizá-lo no formato digital.
- Microcomputador pessoal com processador Intel i7, 32 gigabytes de memória e placa de vídeo de 32MB (alto rendimento);
  - utilizado como principal instrumento de desenvolvimento.
- Microcomputador pessoal do tipo *notebook*, com processador Intel i3, 2 gigabytes de memória (baixo à médio rendimento);
  - utilizado como instrumento de desenvolvimento secundário, para testar questões de consumo de memória e processamento (desempenho).

3.1.2 Softwares

As subseções seguintes citam e detalham as tecnologias utilizadas para a implementação do trabalho aqui proposto.

3.1.2.1 Linguagem Java

A tecnologia Java é usada para desenvolver aplicativos para uma ampla variedade de ambientes, de dispositivos consumidores a sistemas corporativos heterogêneos (PERRY, 2016).

O Java possui classes prontas para a solução dos mais diversos problemas. A seguinte lista apresenta algumas das classes ou pacotes que este trabalho fará uso:

- Pacote ***java.swing*** e ***java.awt***: possui classes para a construção da interface gráfica para o usuário do sistema. Botões, janelas, manipulação de eventos e outros componentes visuais;
- Pacote ***java.io***: utilizado para manipular entrada e saida de informações, manipulação de arquivos e pastas, etc;
- Pacote ***java.util***: possui classes de estrutura e manipulação de estruturas de dados e classes utilitárias;
- outros pacotes são utilizados casualmente de acordo com a necessidade como o ***java.lang*** que provê classes que são fundamentais para a programação Java, por exemplo as classes de criação e execução de *threads*;

3.1.2.2 Biblioteca OpenCV, JavaCV e JavaCPP

A biblioteca OpenCV (*Open Source Computer Vision Library*) foi originalmente desenvolvida pela Intel em 2000 e hoje é uma biblioteca multiplataforma, totalmente livre ao uso acadêmico e comercial, para o desenvolvimento de aplicativos na área de Visão computacional (OPENCV, 2018).

Seu código foi escrito nas linguagens C e C++, porém possui interface para várias outras linguagens como Python, Visual Basic, Java, dentre outras. Incorpora algoritmos para a resolução de vários problemas na área de visão computacional, inclusive o algoritmo *Haar-Cascades* descrito na Subseção 2.2.2.

A biblioteca JavaCPP, atualmente mantida pelo grupo ByteDeco, é considerada uma *bridge* ("ponte") entre a linguagem C++ e Java, disponibilizando classes de configurações e interfaces para várias bibliotecas escritas em C++, inclusive a OpenCV (BYTEDECO, 2018a).

Por sua vez, a biblioteca JavaCV, inicialmente desenvolvida pela Google e agora mantida pelo grupo ByteDeco, utiliza as interfaces e *wrapers* da biblioteca JavaCPP e provê classes que podem ser usadas para manipular a OpenCV de maneira mais fácil na plataforma Java (BYTEDECO, 2018b). Também oferece drivers de aceleração gráfica que otimizam o funcionamento da OpenCV na plataforma Java, dentre outras utilizades.

Estas três bibliotecas funcionam de maneira conjunta para resolver a problemática de aquisição, processamento de imagens, e classificação de objetos, e será utilizada neste trabalho.

A lista a seguir disponibiliza e detalha os principais pacotes, classes e métodos ou funções destas bibliotecas de desenvolvimento que são utilizadas. As definições de

suas funções foram retiradas das documentações dispostas em OpenCV (2018), ByteDeco (2018b) e ByteDeco (2018a):

- Pacotes/classes:
  - ***javacpp.opencv_core.CvMemStorage***: classe que representa um bloco de memória para armazenamento de vários componentes da OpenCV. Trata problemas de alocação e desalocação de memória;
  - ***javacpp.opencv_core.CvRect***: esta classe é utilizada para representar uma região de uma imagem com pontos x e y, altura e largura do segmento;
  - ***javacpp.opencv_core.CvSeq***: classe que representa uma lista (ou sequência) de classes do tipo *CvRect*;
  - ***javacpp.opencv_core.IplImage***: utilizada para representar uma imagem no fomrato *raster*;
  - ***javacv.VideoInputFrameGrabber***: classe responsável por estabelecer conexão com câmeras conectadas ao computador;
  - ***javacv.FrameGrabber***: esta classe tem a função de monitorar o fluxo de *frames* de uma *webcam* conectada a partir da classe *VideoInputFrameGrabber* e disponibilizar uma imagem advinda deste fluxo quando requisitado;
  - ***javacv.Java2DFrameConverter***: tem a função de converter as classes de representações de imagens *IplImage* e *java.awt.image*;
  - ***javacv.OpenCVFrameConverter***: tem a função de converter as classes de representações de imagens *IplImage* e *javacv.Frame*, disponibilizada pela classe *FrameGrabber* acima listada;
  - ***javacpp.opencv_objdetect.CvHaarClassifierCascade***: classe que contém as configurações e o algoritmo de classificação Haar-Cascades, utilizado para a detecção de objetos (ou faces).
- Métodos ou funções:
  - ***javacpp.opencv_core.cvClearMemStorage***: esta função desaloca o espaço de memória utilizada pela classe *CvMemStorage*;
  - ***javacpp.opencv_core.cvCreateImage***: responsável por criar objetos da classe *IplImage*, aceitando parâmetros como produndidade de cores, tamanho da imagem;
  - ***javacpp.opencv_core.cvReleaseImage***: este método libera o espaço de memória ocupado pelos objeto das classe *IplImage*;
  - ***javacpp.opencv_core.cvGetSeqElem***: manipula objetos da classe *CvSeq*;

- ***javacpp.opencv_core.cvGetSize***: função dispõe o tamanho em *bytes* de um objeto das classes *CvSeq*, *CvRect* e *IplImage*;
- ***javacpp.opencv_imgproc.cvCvtColor***: função para mudar a disposição e profundidade de cores de uma imagem da classe *IplImage*, por exemplo, convertendo-as em escala de cinza como descrito no Subseção 2.1.5.1;
- ***javacpp.opencv_imgproc.cvEqualizeHist***: função com a habilidade de performar equalização de histograma, descrito na Subseção 2.1.5.3;
- ***javacpp.opencv_imgproc.cvResize***: função de escalonamento, como descreve a Subseção 2.1.5.2;
- ***javacpp.helper.opencv_objdetect.cvHaarDetectObjects***: este método recebe um objeto do tipo *CvHaarClassifierCascade*, uma imagem do tipo *IplImage*, dentre outros parâmetros, e efetivamente detecta objetos (no caso, face) na imagem retornando objetos da classe *CvSeq* ou *CvRect*;
- outras funções ou métodos para manipular ou converter classes de objetos irão eventualmente ser utilizadas.

Estas três bibliotecas também possuem um conjunto de parâmetros e constantes pré-estabelecidas para facilitar o desenvolvimento, e irão ser utilizadas.

3.1.2.3 Biblioteca Colt

A biblioteca Colt desenvovida pela CERN (sigla francesa que corresponde à *Conseil Européen Pour la Recherche Nucléaire*, em portugês Conselho Europeu de Pesquisas Nucleares) provê uma gama de bibliotecas para Computação Científica e Técnica de Alta Performance em Java (COLT, 2018). Será utilizada na implementação do algoritmo *EigenFaces* para performar as operações necessárias descritas na Subseção 2.3.1. Lista-se a seguir as classes que serão utilizadas desta biblioteca (COLT, 2018):

- ***cern.colt.matrix.DoubleMatrix2D***: uma classe abstrata para representrar matrizes de duas dimensões, contendo elementos do tipo primitivo *double*;
- ***cern.colt.matrix.impl.DenseDoubleMatrix2D***: uma classe que herda e implementa os métodos abstratos da classe *DoubleMatrix2D* acima descrita. Possui métodos para clonar a matriz, reaver algum elemento, buscar algum elemento, conversão em vetores de *double* por exemplo, dentre outras utilidades;
- ***cern.jet.math.Functions***: contém classes métodos que definem funções matemáticas a serem aplicadas em matrizes respresentadas pela classe *DenseDoubleMatrix2D*, como soma, multiplicação, subtração de matrizes, etc;

- ***cern.colt.matrix.linalg.EigenvalueDecomposition***: classe que contém métodos para a extração dos auto-vetores (*EigenValues*) e auto-valores (*EigenVectors*) de uma matriz recebida, necessários para certas etapas do algorítio *EigenFaces* ou ACP descrito na Subseção 2.3.1.

3.1.2.4 NetBeans IDE

O NetBeans é um IDE (*Integrated Development Environment*, ou em português, Ambiente Integrado de Dezemvolvimento) livre licensiado pela *Common Development and Distribution License* (CDDL) e *GNU General Public License*.

Este editor de código tráz facilidades como auto-complemento de código, desenho visual de interface, atalhos de teclados para maior produtividade na escrita, *debugger*, dentre outras vantagens, e foi escolhido para a implementação neste trabalho.

3.1.2.5 Sistema GIT

Git é um sistema de gerenciamento e controle de código livre criado pelos desenvolvedores Linus Tornalds e Junio Hamano sob a licensa GNU (GPLv2) (GIT, 2018). Este sistema é extremamente útil para se ter cada etapa da escrita do código salva, com possibilidade de reversão de ações, criação de ramos de desenvolvimento dentre outras utilidades, e é utilizado tanto na escrita desta monografia quanto na escrita do código do sistema de reconhecimento aqui proposto.

3.1.2.6 Sistemas Linux e Windows

O ambiente de desenvolvimento deste trabalho foi configurado sob o sistema operacional Linux, executando na máquina de alto rendimento descrita na Subseção 3.1.1, item "2". Já o sistema operacional Windows executa-se na maquina de baixo rendimento descrito também na Subseção 3.1.1, item "3", para testes e apresentação.

3.2 MÉTODOS

Este trabalho adota uma abordagem experimental, descrita na seção a seguir.

3.2.1 Método Experimental

No método experimental, variáveis são tratadas de maneira pré-estabelecida e seus efeitos suficientemente controlados e conhecidos pelos pesquisador. O princípio central da aplicação do método experimental é que deve-se aceitar os resultados como eles são apresentados, com imprevistos e acidentes que possam haver diante dos resultados obtidos, ignorando as próprias opiniões e também as alheias (FACHIN, 2017).

3.2.2 Desenvolvimento Ágil

O Desenvolvimento Ágil foi padronizado pelo Manifesto Ágil, como um método independente que reúne características de outras metodologias. Este manifesto foi criado para valorizar os seguintes pontos (ARAUJO, 2016):

- Indivíduos e suas interações são mais importantes que processos e ferramentas;
- Software que funciona importa mais que uma documentação abrangente;
- Colaboração com o cliente vale mais do que negociação de contratos;
- Responder às mudanças é melhor do que seguir planos.

No desenvolvimento ágil, os projetos adotam o modelo iterativo e em espiral. Neste processo, as fases do projeto são executadas diversas vezes, produzindo ciclos curtos que se repetem ao longo de todo o desenvolvimento, sendo que, ao final de cada ciclo, sempre se tem um software funcional. Os ciclos são chamados de iterações e crescem em número de funcionalidades a cada repetição, sendo que, no último ciclo, todas as funcionalidades desejadas estariam implementadas, testadas e aprovadas (CASTRO, 2018).

3.2.2.1 XP (eXtremeProgramming)

Faz parte dos métodos ágeis, já que se encaixa no que define o que é um método ágil, conforme o Manifesto Ágil. Ela é baseada em três pilares: agilidade no desenvolvimento, economia de recursos e qualidade do produto final (CASTRO, 2018).

A **XP** será utilizada neste projeto por ser incremental, focar-se no desenvolvimento ao invés da documentação e gerar, a cada iteração, um protótipo, essencial para testes e refinamento até que se chegue à um resultado satisfatório.

3.2.3 Modelagem UML

A modelagem UML (*Unified Modeling Language*, em português Linguagem de Modelagem Unificada), por vezes chamada de linguagem UML, procura fornecer meios para auxiliar no levantamento dos requisitos que irão constituir um sistema, além de recursos para a modelagem de estruturas que farão parte do mesmo. O fato da UML ser um padrão de grande aceitação no mercado também se deve, em grande parte, à forte integração desta com conceitos da Orientação a Objetos (OO) (GROFFE, 2013). Como este trabalho propõe a implementação na liguagem Java que é orientada a objetos, este padrão de modelagem será utilizado na elaboração de documentos modelando os componentes esperados para o funcionamento do sistema.

Dois dos diagramas que fazem parte da linguagem UML foram utilizados (GROFFE, 2013):

- Diagrama de Pacotes: Demonstra as interações entre diferentes pacotes os quais contém conjunto classes que compartilham finalidades semelhantes;
- Diagrama de Classes: Permite a visualização do conjunto de classes, detalhando atributos e operações (métodos) presentes, assim como prováveis relacionamentos entre essas estruturas;
- Diagrama de Sequência: Permite a visualização do conjunto de classes e seus fluxos de informações, respeitando a ordem cronológica das chamadas de seus métodos.

Estes dois diagramas foram utilizados para ilustrar as classes Java que serão implementadas, as interações de seus objetos e os processos do sistema que consiste na aquisição do vídeo, deteção e reconhecimento da face, ou seja, toda a entrada, processamento e saída de dados do sistema detalhando seus componentes e interações.

Neste capítulo foram abordados os materiais e métodos necessários para o desenvolvimento do sistema de reconhecimento de faces em vídeo, utilizando o algorítimo *EigenFaces*. Pode-se destacar a modelagem que trouxe esclarecimento quanto à estrutura do sistema e ao processo que deve ser percorrido, o desenvolvimento ágil considerado essencial para se ter resultados rápidos para que sejam analisados e iterados num ciclo de desenvolvimento espiral com melhoras a cada iteração. Outro destaque são as bibliotecas OpenCV, JavaCV e Colt, que juntamente com suas documentações facilitaram o processo de implementação do algorítimo *EigenFaces* e o desenvolvimento do sistema como um todo e por fim, o sistema GIT que grava e assegura cada etapa do processo de escrita da monografia e do cógido.

4 IMPLEMENTAÇÃO

Neste capítulo serão abordadas questões do planejamento e da implementação do sistema de reconhecimento de faces em vídeo. Primeiramente, os requisitos do sistemas serão analisados e descritos, depois a modelagem destes requisitos em forma de fluxograma, e a modelagem das classes que foram criadas por meio dos diagramas descritos na Subseção 3.2.3, e por fim, as questõs de configuração do ambiente de desenvolvimento e a implementação efetiva do código. Todo este processo é feito seguindo conceitos da metodologia ágil XP, descrita na Subseção 3.2.2.

4.1 ANÁLISE DE REQUISITOS

Para que se possa iniciar o planejamento da implementação do sistema, aqui proposto, por meio de diagramação, deve-se colher os requisitos necessários para o funcionamento do sistema, ou seja, analisar o problema descrevendo entradas e saídas de dados, todas as situações e seus fluxos de informações. Sendo assim lista-se por ordem de execução os requisitos que atendem à proposta do sistema, discriminados por **entrada**, **processamento** e **saída** de dados ou informações. Para melhor entendimento, ilustra-se o processo desta seção na Figura 21.

1. **entrada:** uma câmera do tipo *webcam*, descrita na Seção 3.1, deve estar filmando uma área com iluminação controlada, produzindo o fluxo de *frames* e disponibilizando em formado digital para o sistema;

2. **processamento/saída:** o sistema adquire controle do fluxo de *frames* da *webcam*, manipulando-o para que se possa ser exibido na tela do microcomputador e separa um quadro para a detecção, em tempo real, utilizando o algoritmo, descrito na Subseção 2.2.2, com os materiais contemplados na Subseção 3.1.2.2;

3. **processamento:** o sistema processo a detecção da face na imagem, e caso seja detectada uma face, o sistema deverá desenhar um retângulo em volta da face detectada neste fluxo de *frames*, e a partir daí se inicia o processo de reconhecimento;

4. **processamento/saída:** o processo de reconhecimento é acionado e caso a face não seja reconhecida, o sistema disponibiliza campos para que o usuário entre com uma identificação. Caso haja alguma equivalência, o sistema identifica o usuário escrevendo seu nome logo abaixo do retângulo desenhado pelo processo de detecção;

5. **entrada/processamento:** se o usuário identifica a face a partir do item acima descrito, o sistema inicia o processo de treinamento da face definido na Seção 2.3, criando um *eigenspace*, e no momento seguinte à criação deste "espaço", o sistema começa a utiliza-lo para os novos processos de reconhecimento.

FIGURA 21 – Ilustração do processo do sistema.

FONTE: Elaborado pelo autor.

4.2 DIAGRAMA DE CASO DE USO

O diagrama de Caso de Uso da Figura 22 ilustra a interação entre o usuário (ator) e o sistema. Observa-se que as únicas atividades do usuário é de se posicionar para que a câmera capture as imagens, e preencher o campo disponibilizado para identificação, caso o usuário deseje.

4.3 DIAGRAMA DE PACOTES

Nesta seção, os pacotes criados para implementação do sistema aqui proposto serão apresentados em forma de diagrama de pacotes UML. A Figura 23 ilustra tal diagrama:

O pacote *"gui"* possui classes responsáveis pela interface com o usuário e regras do sistema como controle de fluxo de quadros e execução de tarefas. O pacote *"eigenfaces"* possui as classes responsáveis pela execução do algoritmo ACP e a geração do espaço multidimensional de *eigenfaces*. Por fim, o pacote de nome *"utils"*, possui classes utilitárias

FIGURA 22 – Diagrama de Caso de Uso - Interação entre o usuário e o sistema.
FONTE: Elaborado pelo autor.

resposáveis por manipular arquivos (carregar, salvar e deletar) e conversões de formatos de imagens.

Na seção seguinte, cada pacote será descrito com seus respectivos diagramas de classe.

4.4 DIAGRAMA DE CLASSES - PACOTE ***GUI***

As classes do pacote ***gui*** são representadas pelo diagrama UML na Figura 24.

A classe ***FrameCadastroDeFace*** é apenas uma janela com alguns componentes visuais, como um botão, um campo para entrada de informação e a tela para a visualizaçào do fluxo de quadros. A tela para visualização do fluxo é representada pela classe ***PanelCadastroDeFace***.

A classe ***PanelCadastroDeFace*** também é responsávem pelo controle de frames e de execução de tarefas como a de deteção, treinamento e reconhecimento das faces. A seguir suas principais objetos e métodos são detalhados:

- Objetos:
  - **CvHaarClassifierCascade classifier**: classificadores pré treinados, assim como descritos na Subseção 2.2.2, para o processo de detecção da face;
  - **ExecutorService executor:** este objeto é uma *thread*, responsável por executar e controlar o processo de aquisiçao de quadros, deteção treinamento e reconecimento;

FIGURA 23 – Diagrama de pacotes UML do sistema proposto.
FONTE: Elaborado pelo autor.

- **Rectangle faceRect:** objeto contendo a posição da face recem detectada em relação ao quadro retirado do vídeo;
- **BufferedImage imagem_rosto:** objeto que salva um a imagem da face, caso detectada;
- **String nomeDaFace:** objeto que salva o nome da face detectada, informação esta que é entrada pelo usuário do sistema;
- **FrameCadastroDeFace pai:** objeto que referencia o seu parente de nível superior da classe ***PanelCadastroDeFace***;
- **ACP_Reconhecimento faceRecog:** objeto que instancia a classe responsável por executar o algoritmo de reconhecimento ACP, contido no pacote ***eigenfaces***;
- **ACP_Treinamento:** esta classe, contida no pacote **eigenfaces**, não é instanciada como objeto, porém é chamada de forma estática para produzir o espaço multidimensional a partir das imagens de treinamnto salvas em uma pasta especifica;

• Métodos:

- **BufferedImage loadImage():** método que carrega imagem do sistema de arquivos para memória;
- **FrameGrabber initGrabber():** estabelece fluxo de quadros com a câmera;

FIGURA 24 – Diagrama de classes UML do pacote ***gui***.

FONTE: Elaborado pelo autor.

- **IpiImage picGrab():** aquisita um quadro proveniente do fluxo de quadros;
- **void drawRect():** desenha um quadrado em volta do rosto aquisitado de acordo com as coordenadas do objeto **faceRect**;
- **void salvaface_traininImages()**: este método salva a face continda no objeto **imagem_rosto** para futuro processo de treinamento;
- **void trackFace():** método chamado dentro da *thread* ExecutorService executor que contém os processos e os controles de estado de detecção, treinamento e reconhecmento da face;
- **boolefan recogFace():** método chamado quando o sistema se encontra no estado de reconhecimento, onde a imagem passa por processamento, redimensionando-a e transformando-a em escala de cinza e passada para o método o objeto **ACP_Reconhecimento.processaReconhecimento()** é acionado;
- **boolean negocioReconhecerClip():** sub-método do método *recogFace()* definido acima, chamado quando o sistema se encontra no estado de reconhecimento, onde a imagem passa por conversões de tipo, aciona efetivamente **ACP_Reconhecimento.processaReconhecimento()** e determina se o reconhecimento foi um sucesso ou uma falha, detalhado posteriormente na autorefsec:defsucregoc, passando os dados de reconhecimento para a interface;

4.5 DIAGRAMA DE CLASSES - PACOTE ***EIGENFACES***

As classes do pacote ***eigenfaces*** são representadas pelo diagrama UML da Figura 25 e descritas em seguida.

FIGURA 25 – Diagrama de classes UML do pacote ***eigenfaces***.

FONTE: Elaborado pelo autor.

A classe ***EigenSpace*** representa o espaço multidimensional citado e descrito na Seção 2.3. Possui métodos de extração de informações contidas neste espaço. Seus objetos são matrizes que representam os autovalores e autovetores contemplados na Subseção 2.3.1.1. També possui objetos que relacionam os autovetores gerados com as suas respectivas imagens de treinamento.

- Métodos:
  - **double[][] getImages():** método que retorna as imagens de treinamento em forma de matriz de bytes de duas dimensões. Cada linha representa uma imagem, e o número de colunas representa os pixels de cada imagem;
  - **double[][] getAutoVetores():** método que retorna os autovetores gerados a partir das imagens de treinamento;
  - **int getNumEigenFaces():** este método retorna o número de *eigenfaces* presentes no espaço multidimensional *EigenSpace*.
  - **double[] getImagem_media():** este método retorna o número de *eigenfaces* presentes no espaço multidimensional *EigenSpace*.
  - **double[] getAuto_valores():** retorna os autovalores correspondentes com os autovetores existentes no espaço;
  - **ArrayList<String> getListaPath_images():** returna uma lista com o caminho completo dos arquivos que representam as imagens de treinamento;
  - **double[][] calculaCoordenadas(int numEFs):** este método recebe o número de *eigenfaces* a se priorizar para analise, como explica a Subseção 2.3.1.1, e retorna as coordenadas (ou pesos) respectivos a cada imagem de treinamento;

A classe ***ACP_ Treinamento*** contém a implementação responsável pelo processo de treinamento do ACP descrito na Subseção 2.3.1.1. Portanto esta classe gera um objeto da classe **EigenSpace** descrita acima. Seus principais métodos são contemplados abaixo:

- Métodos:
  - **EigenSpace gerar_EigenSpace():** este método engloba os cálculos do algoritmo ACP, especificamente os passos da fase de treinamento da Subseção 2.3.1.1, retornando um objeto da classe *EigenSpace* detalhada anteriormente.
  - **Matriz2D calcEspaco():** este método engloba especificamente o passos '5' da fase de treinamento referente ao cálculo do espaço, contempladas na Subseção 2.3.1.1.
  - outros métodos para conversão de objetos da classe *Double* para o tipo primitivo *double*, conversão de imagens representadas pela classe *BufferedImage* para *DoubleMatrix2D*, contemplada na Subseção 3.1.2.3, etc.

A classe ***ACP_ Reconhecimento*** contém a implementação responsável pelo processo de reconhecimento da ACP descrito na Subseção 2.3.1.2. Portanto esta classe gera um objeto da classe **ResultadoReconhecimento** posteriormente nesta seção. Seus principais métodos são contemplados abaixo:

- Objetos:
  - EigenSpace eigenSpace: objeto da Classe ***EigenSpace***, descrita acima nesta seção, criada pela Classe *ACP_Treinamento* e utilizada para a fase de reconhecimento.
- Métodos:
  - **void processa_Reconhecimento():** este método engloba os cálculos do algoritmo ACP, especificamente os passos da fase de reconhecimento da Subseção 2.3.1.2.
  - **Matriz2D calcEspaco(int nEF, Matriz2D imMat)**:
  - **double[] getEucDists(Matriz2D im)**: método que calcula a distância euclidiana de todas as faces treinadas (autovalores) contidos no espaço com a nova face;
  - **ResultadoReconhecimento getMenorDist(double[] dists)**: método que retorna qual das distâncias calculadas acima é a menor, construindo um objeto da classe ***ResultadoReconhecimento***;
  - esta classe também contém métodos para conversões de tipos, dentre outros.

A classe ***ResultadoReconhecimento*** representa o resultado do reconhecimento procesada pela classe ACP_Reconhecimento contemplada anteriormente. Seus principais métodos são detalhados abaixo:

- Métodos:
  - **String getName():** método que retorna a identificação da face previsamente informada pelo usuário, correspondente com o resultado;
  - **String getMatchFileName():** método que retorna o caminho do arquivo que corresponde ao resultante.
  - **BufferedImage getMatchDist():** método que retorna distância corresponde com o resultado;

4.6 DIAGRAMA DE CLASSES - PACOTE ***UTILS***

As classes do pacote ***utils*** são representadas pelo diagrama UML da Figura 26.

A classe ***FileUtils*** contém métodos que carregam arquivos do disco rígido para a memória, conversão entre classes de imagem, e também que salvam as imagens contidas na memória para o disco rígido, dentro outros como remoção de arquivos e manipulação de pastas.

A classe ***ImageUtils*** contém métodos que responsáveis por algumas rotinas de processamento de imagens. Seus principais métodos são descritos abaixo:

FIGURA 26 – Diagrama de classes UML do pacote ***utils***.

FONTE: Elaborado pelo autor.

- Métodos:
  - **String clipToRectangle(BufferedImage, int, int, int, int):** este método recebe uma imagem e quatro pontos, e segmenta a imagem de acordo com estes parâmetros;
  - **BufferedImage escalaImagemCinza(BufferedImage, double):** recebe uma imagem, a escalona para proporcionalmente de acordo com um parâmetro constante de escala e retorna outra imagem representada pela classe BufferedImage, já transformada para escala de cores cinza;
  - **BufferedImage escalaImagemCinza(IpiImage)**: apresenta a mesma finalizade do método acima porém recebendo um objeto da classe IpiImage da biblioteca OpenCV descrita na Subseção 3.1.2.2, retornando outra imagem representada pela classe BufferedImage, já transformada para escala de cores

cinza;

- esta classe também possui métodos de conversão de vetores para o tipo *BufferedImage* e vice-versa.

A classe ***AutoVetor_decomp*** herda as propriedades da classe *EigenvalueDecomposition* da biblioteca *COLT* contemplada na Subseção 3.1.2.3, responsável por decompor uma matriz de covariância em autovalores e autovetores, como detalha a Subseção 2.3.1.1.

A classe ***EqualizarHistograma*** é responsável por processar a equalização de histogramas em todas as imagens de treinamento, as quais são salvas pela classe *FileUtils* em uma pasta definida no disco rígido.

Por fim, a classe ***Matriz2D*** herda a propriedades da classe ***DoubleMatriz2D*** da biblioteca *COLT*, responsável por representar matrizes com funções para calculá-las. Contém métodos de conversões de tipos e cálculo de matrizes. Seus principais métodos são detalhados abaixo, os quais herdam as funcionalidades contempladas na Subseção 3.1.2.3:

- Métodos:
  - **void subtract(Matriz2D):** este método recebe uma matriz a ser subtraída e realizar o cálculo;
  - **String max(double[]):** recebe um vetor e retorna seu maior valor;
  - **void normalise():** normaliza a objeto matriz representada por esta classe;
  - **void transpose():** realiza o processo de transposição da matriz representada por esta classe;
  - **double[] calcMedia_cols():** calcula a média das colunas da matriz representada por esta classe, retornando um vetor com os estes valores médios.
  - ***void subtrairMedia()*:** este método faz a chamada do método acima descrito *calcMedia_cols* e a subtrai com a matriz representada por esta classe, tendo como resultado um vetor médio subtraído, ou face média subtraída, como descreve a Subseção 2.3.1.1.

Para melhor visualização e entendimento da relação entre estas classes e seus métodos, apresenta-se na Figura 27 o diagrama de sequência das classes *ACP_Treinamento*, *Matriz2D*, *FileUtils* e *EqualizarHistograma*. Também apresenta-se o diagrama de sequência das classes *Matriz2D*, *AutoVetor_decomp*, *DenseDoubleMatriz* e *EigenValueDecomp*, sendo estas duas últimas, classes pertencentes à biblioteca Colt, contemplada na Subseção 3.1.2.3. Os detalhes das operações executadas por estes métodos serão contemplados na Seção 4.7.

FIGURA 27 – Diagrama de sequência UML das classes *ACP_Treinamento*, *Matriz2D*, *FileUtils* e *EqualizarHistograma*.

FONTE: Elaborado pelo autor.

4.7 IMPLEMENTAÇÃO DO CÓDIGO

Nesta seção será abordado a implementação do código, definindo quais foram as variáveis (ou constantes em termos de programação) pré-estabelecidas e constroladas para o obter diversos efeitos ou resultados, como define o método experimentao descrito na Subseção 3.2.1.

Também serão contemplados os *snippets* (ou blocos de código) referentes às principais partes implementadas do sistema e descritas neste trabalho: o fluxo do processo do sistema como ilustra na Figura 21 da Seção 4.1, a fase de treinamento e a de reconhecimendo do algoritmo EigenFaces (ACP), descritos na Subseção 2.3.1.

4.7.1 Constantes Controladas

De acordo com o método experimental, utilizou-se três constantes controladas como variáveis do experimento, para que sejam manipulados e seus efeitos obervados. Existem outras constantes no sitema para constrole de quadros analisados ou do tamanho padrão da imagem que tem efeito no desempenho ou em outros aspectos do sistema, porém os três principais listados abaixo influem na taxa de sucesso/falha do reconhecimento, os quais serão contemplados no Capítulo 5, referente aos testes e resultados.

- **NUM_FACES_TREINO:** esta constante é resposável por controlar o número de faces a ser usado no treinamento, acionado quando o usuário informa sua iden-

FIGURA 28 – Diagrama de sequência UML das classes *Matriz2D*, *AutoVetor_decomp*, *DenseDoubleMatriz* e *EigenValueDecomp*.

FONTE: Elaborado pelo autor.

tificação. Apresenta-se sua lógica no código na Subseção 4.7.2;

- **NUM_EF_recog:** esta constante é responsável por definir quantos autovetores serão utilizados para o reconhecimento, pois seu descarte é possível devido a ordenação por autovalores, como contempla a Subseção 2.3.1.1, passo quatro, e a Subseção 4.7.4. Sua lógica é apresentada na Subseção 4.7.4;

- **MIN_DIST:** esta constante define se a distância encontrada na fase de reconhecimento pode ser considerada um resultado de sucesso ou falha. Sua lógica apresentada na Subseção 4.7.5.

4.7.2 Processo Principal do Sistema

O bloco do CÓDIGO 1 corresponde a *Thread* (classe para processamento paralelo) que controla o fluxo do sistema que se refere aos processos de detecção, treinamento e reconhecimento da imagem capturada do fluxo de quadros, disponibilizadas pela *webcam*.

CÓDIGO 1 – Bloco de código do método *trackFace()* da classe *PanelCadastroDeFace*

```
447 CvRect faceDetectada = detectarFace(grayIm);
448 if (faceDetectada != null) //detectou a face:
449 {   //salva posicao da face detectada na variavel global
450     setRectangle(faceDetectada);
451     //abre campos para identificacao
452     pai.setSalvarFaceVisible(true);
```

```
453
454     //se esta em estado de treinamento
455     if (isSalvaTreinaFace()) {
456         String nomeFace = pai.getNomeFaceField();
457         if (!pegouOIndexDoNome) {
458             ultimoIndexNomeFace = getUltimoIndexNomeFace(nomeFace);
459             pegouOIndexDoNome = true;
460         }
461         if (salvaFace_trainingImages(img, nomeFace, ultimoIndexNomeFace++))
462             countFacesParaTreino++;
463         if (countFacesParaTreino > NUM_FACES_TREINO) {//TREINA ATE TER ↩
    NUM_FACES_TREINO
464             setSalvaFace(false);//desliga o treino
465
466             //reseta vars de controle
467             pegouOIndexDoNome = false;
468             countFacesParaTreino = 0;
469             pai.setSalvarFaceVisible(false);
470
471             //thread para criar novo bundle
472             ExecutorService criaBunbdle = Executors.↩
    newSingleThreadScheduledExecutor();
473             criaBunbdle.execute(new Runnable() {
474                 @Override
475                 public void run() {
476                     criandoBase = true;
477                     long startTime = System.currentTimeMillis();
478                     ACP_Treinamento.construirEspaco(NUM_EF_treino);//cria novo↩
     bundle
479                     System.out.println("BUNDLE CRIADO EM " + (System.↩
    currentTimeMillis() - startTime) + "ms");
480                     faceRecog = new ACP_Reconhecimento(NUM_EF_recog);//usa ↩
    novo bundle
481                     criandoBase = false;
482                 }
483             });
484         }
485     } else {
486         //reconhece face
487         recogFace(img);
488     }
```

Este bloco está contido do método *trackFace()* da classe *PanelCadastroDeFace*, descritas na Seção 4.4, que é chamado a cada captura de quadros, atribuindo como parâmetro a image já redimensionada e transformada para cores da escala cinza, como mostra o CÓDIGO 1.

A linha de número *447*, o método *detectarFace(grayIm)* recebe uma imagem já redimensionada e transformada em escala da cinza pelo método *escalaImagemCinza()*, descrito na Seção 4.6, e processa a detecção da face utilizando o algoritmo de Viola-Jones (descritos na Subseção 2.2.2). Se a face não for detectada, nada acontece. Caso a face seja detectada na imagem, as coordenadas desta são salvas em uma variável global e os campos

de identificação contidos na classe *FrameCadastroFace* para identificação do usuário são ativados (linha *450* e *452*, respectivamente).

Caso o usuário se identifique atravéz dos campos de entrada, o sistema se encontrará em "estado de treinamento", controlado pelo método *isSalvaTreinaFace()* da linha 455. Neste bloco salva-se a imagem da face para treino quantas vezes for definida pela constante *NUM_FACES_TREINO*, definidas na Subseção 4.7.1. Quando atingir este valor, o "estado de treinamento" do sistema, bem como os campos de entrada são desativados e começa a rotina de construção do espaço (*eingenspace*), executada pelo método *ACP_Treinamento.construirEspaco()*, que será detalhado em seguida nesta seção.

Se o sistema não se encontra em "estado de treinamento", controlado pelo método *isSalvaTreinaFace()* da linha *445*, o método *recogFace()* (linha *487*) é executado, o qual será detalhado na Subseção 4.7.4.

4.7.3 Fase de Treinamento

O método *gerar_EigenSpace()* do CÓDIGO 2 é chamado pelo método *ACP_Treinamento.construirEspaco()*, contemplado no bloco de código anterior, que por sua vez resgata todas as imagens de treino, salvar em disco rígido e retorna os caminhos destes arquivos.

CÓDIGO 2 – Bloco de código do método *gerar_EigenSpace()* da classe *ACP_Treinamento*

```
64 private static EigenSpace gerar_EigenSpace(ArrayList<String> fnms){
65         //PASSO 1 :: DADOS!
66         //carrega imagens da pasta de treinamento
67         BufferedImage[]ims=FileUtils.carregaImagensDeTreino(fnms);
68         //converte bufferedImage para matriz2D
69         Matriz2D matriz_imgs=buffImage2Matriz2D(ims);
70 
71         //PASSO 2:: CALCULAR A MÉDIA DE CADA IMAGEM E APLICAR SUBTRAÇÃO COM AS↩
    MESMAS
72         double[]avgImage=matriz_imgs.calcMedia_cols();
73         matriz_imgs.subtrairMedia();//imagens de treino com a face média ↩
    subtraída
74 
75         //PASSO 3:: CALCULAR A MATRIZ DE COVARIÂNCIA
76         Matriz2D imsDataTr=matriz_imgs.transpose();
77         Matriz2D covarMat=matriz_imgs.multiply(imsDataTr);
78 
79         //PASSO 4:: CALCULAR OS AUTOVETORES E AUTOVALORES DA MATRIZ DE ↩
    COVARIANCIA
80         AutoVetor_decomp egValDecomp=covarMat.getEigenvalueDecomp();
81         double[]egVals=egValDecomp.getEigenValues();
82         double[][]egVecs=egValDecomp.getEigenVectors();
83 
84         //PASSO 4.1:: ordenar o vetor de autovetores por ordem de autovalores ↩
    (para futuro possivel descarte)
85         ordenaEgVecs(egVals,egVecs);
```

```
86
87          //PASSO 5:: No último passo cada imagem de treinamento é projetada no ↩
    espaço face.
88          // "O descritor ACP (ou engeifaces, normalizados) é calculado por uma ↩
    combinação linear
89          // de Auto-vetores com os vetores originais."
90          Matriz2D egFaces=calcEspaco(matriz_imgs,new Matriz2D(egVecs));
91
92          System.out.println("::: Salvando EigenFaces como imagens...");
93          FileUtils.salvarEigenfaces_imgs(egFaces,ims[0].getWidth());
94          System.out.println("::: EIGENFACES GERADOS :::");
95
96          //Para cada face, apenas os coeficientes  são armazenados para futura ↩
    comparação.
97          return new EigenSpace(fnms,matriz_imgs.toArray(),avgImage,egFaces.↩
    toArray(),
98          egVals,ims[0].getWidth(),ims[0].getHeight()
99          );
100 }
```

Este bloco é o responsável por executar a fase de treinamento do algoritimo ACP (ou *EigenFaces*), contemplada na seção Subseção 2.3.1.1, e seus cinco passos:

No primeiro passo, carrega-se as imagens do disco rígido para a memória e as converte para as classes que as representam em matrizes (linha *67* e *69*).

Para o segundo passo, o cálculo da face média é executado e subtrai-se esta com a matriz criada no primeiro passo, como está nas linhas *72* e *73* do CÓDIGO 2.

No terceiro passo, o vetor de covariância é calculado multiplicando a matriz gerada no passo anterior com sua equivalente transposta, como contempla a Subseção 2.3.1.1 (linhas *76* e *77*).

No quarto passo deve-se decompor a matriz de covariância em autovalores (*eigenvalues*) e autovetores (*eigenvectors*). Isto é feito atravéz dos métodos *getEigenValues()* e *getEigenVectors()* (linhas *81* e *82*, respectivamente) da classe *AutoValor_decomp*, descritas na Seção 4.6, e posteriormenter ordenados por ordem de autovalores (linha *85*).

Para o quinto e último passo desta fase de treinamento, o cálculo do espaço (*eigenspace*) é executado atravéz do método *calcEspaco()*, descritos no bloco do CÓDIGO 3.

CÓDIGO 3 – Bloco de código do método *calcEspaco()* da classe *ACP_Treinamento* - responsável por executar o quinto passo da fase de treinamento.

```
171 private static Matriz2D calcEspaco(Matriz2D imsMat,Matriz2D egVecs){
172         Matriz2D egVecsTr=egVecs.transpose();
173         Matriz2D egFaces=egVecsTr.multiply(imsMat);
174         double[][]egFacesData=egFaces.toArray();
175
176         //normalizacao
177         for(int row=0;row<egFacesData.length;row++){
178             double norm=Matriz2D.norm(egFacesData[row]);   // valor normal
179             for(int col=0;col<egFacesData[row].length;col++)
180                 egFacesData[row][col]=egFacesData[row][col]/norm;//normaliza
```

```
181         }
182         return new Matriz2D(egFacesData);
183 }
```

4.7.4 Fase de Reconhecimento

O método *processaReconhecimento()* contemplado no bloco do CÓDIGO 4, é responsável por executar os três passos da fase de reconhecimento do algoritmo *Eigenfaces* (ACP), descritos na Subseção 2.3.1.2.

CÓDIGO 4 – Bloco de código do método *processaReconhecimento()* da classe *ACP_Reconhecimento*

```
113 private ResultadoReconhecimento processaReconhecimento(BufferedImage im) {
114         //PASSO 1:: CONVERTE IMAGEM PARA VETOR E NORMALIZA
115         //converte imagem para vetor
116         double[] imArr = ImageUtils.createArrFromIm(im);
117         Matriz2D imMat = new Matriz2D(imArr, 1);
118         // normaliza o vator
119         imMat.normalise();
120 
121         //PASSO 2:: PROJETAR O VETOR DE CONSULTA NO ESPAÇO
122         /// multiplicando de autovetores com o vetor DE CONSULTA com a face mé↩
    dia já subtraida
123         // subtracao da face media
124         imMat.subtract( new Matriz2D( this.getEigenSpace().getImagem_media(), ↩
    1) );
125         // projetar o vetor de consulta no espaço face, retornando suas ↩
    coordenadas do espa'co
126         // limitar o uso das eigenfaces "autovetores" por NUM_EF_recog ↩
    previamente fornecida
127         Matriz2D espaco = calcEspaco(num_eigenfaces, imMat);
128 
129         //PASSO 3:: calcula a distancia euclidiana entre a nova imagem e as ↩
    imagens pre treinadas (eigenfaces)
130         double[] dists = this.getEucDists(espaco);
131         InfoDistancia distInfo = getMenorDist(dists);
132 
133         //consulta o nome da imagem
134         ArrayList<String> imageFNms = this.getEigenSpace().↩
    getListaPath_imagens();
135         String matchingFNm = imageFNms.get(distInfo.getIndex());
136 
137         //extrai raiz quadrada
138         double minDist = Math.sqrt(distInfo.getValue());
139 
140         //salva no objeto
141         return new ResultadoReconhecimento(matchingFNm, minDist);
142 }
```

O método *regocFace()* da classe *PanelCadastroDeFace*, como descrito na da Seção 4.4, invoca este método *processaReconhecimento()* para fazer o reconhecimento, pas-

sando imagem da classe *BufferedImage* já processada, que representa a nova face a ser reconhecida.

Para o primeiro passo da fase de reconhecimento, prepara-se os dados para o cálculo fazendo conversão entre classes (linha *116* e *117*), e normaliza a matriz que representa a nova imagem com a média dos próprios valores (linha *119*).

No segundo passo projeta-se o novo vetor de consulta ao espaço previamente criado na fase de treinamento. Para isto, é necessário extrair a face média do espaço, previamente salva no objeto da classe *eigenspace*, e subtraí-la do vetor de consulta (linha *125* do CÓDIGO 4) para que seja atendida a equação de definição do espaço com o novo vetor de consulta, detallhada na Subseção 2.3.1.2.

O método *calcEspaco()* (linha *128* do CÓDIGO 4), semelhante ao método *calcEspaco()* da classe *ACP_ Treiamento*, é responsável pelo cálculo do novo espaço com o novo vetor de consulta, detalhado e ilustrado no bloco de código do CÓDIGO 5. Nele recupara-se os autovalores contidos no espaço previamente criado pela fase de treinamento (linha *150* - CÓDIGO 5) e se extrai a submatriz determinada de autovalores pela constante *NUM_ EF_ recog*, especificada na Subseção 4.7.1 (linha *151* - CÓDIGO 5), seguidas das operações de transposição e multiplicação (linhas *152* e *154* do CÓDIGO 5), para que se atenda a equação de criação do espaço (*eigenspace*) com o novo vetor de consulta (Subseção 2.3.1.2).

CÓDIGO 5 – Bloco de código do método *calcEspaco()* da classe *ACP_ Treinamento*

```
148 private static Matriz2D calcEspaco(Matriz2D imsMat,Matriz2D egVecs){
149         Matriz2D egVecsTr=egVecs.transpose();
150         Matriz2D egFaces=egVecsTr.multiply(imsMat);
151         double[][]egFacesData=egFaces.toArray();
152 
153         //normalizacao
154         for(int row=0;row<egFacesData.length;row++){
155             double norm=Matriz2D.norm(egFacesData[row]);   // valor normal
156             for(int col=0;col<egFacesData[row].length;col++)
157                  egFacesData[row][col]=egFacesData[row][col]/norm;//normaliza
158         }
159         return new Matriz2D(egFacesData);
160 }
```

Para o terceiro e último passo da implementação desta fase de reconhecimento, a Distância Euclidiana entre as coordanadas de cada autovetor (*eigenface*) geradas na faze de treinamento e as coordenadas dos mesmos no novo espaço gerados nesta faze são calculados (linha *131* do CÓDIGO 4), subtraíndo-as e elevando-as ao quadrado. A menor delas é elegida (linha *132* do CÓDIGO 4) e é calculada sua raiz quadrada para completar a equação da Distância Euclidiana, contemplada na Subseção 2.3.1.2, criando um objeto da classe *ResultadoReconhecimento*, detalhado na Seção 4.5, que salva o resultado da melhor distância obtida e a identificação da imagem de treinamento correspondente (a face).

4.7.5 Definição de Sucesso do Reconhecimento

Como visto na definição do algoritmo *EigenFaces*, disposta na Subseção 2.3.1, este apenas apresenta como resultado a menor distância entre os autovetores e o novo vetor de consulta, sendo responsabilidade de outro processo a análise deste resultado para definir se efetivamente este valor corresponte a uma reconhecimento de sucesso ou uma falha.

A constante *MIN_DIST*, definida na Subseção 4.7.1, é a responsável pelo resultado final de sucesso ou falha, a qual é controlada e nos testes da Capítulo 5 posterior. Como mostra o bloco de código do CÓDIGO 6, a checagem desta constante contra a distância encontrada na fase de reconhecimento pode ser vista na linha *614*.

CÓDIGO 6 – Bloco de código do método *negocioReconhecerClip()* da classe *PanelCadastroDeFace*

```
607 if (result == null) {
608     System.out.println("RECOG SEM RESULTADO! =/");
609     return false;
610 } else {
611     double distancia = result.getMatchDistance();
612     String distStr = String.format("%.4f", distancia);
613     String nomeDaFaceBanco = result.getName();
614     if (distancia < MIN_DIST) { //se for menor q MIN_DIST = sucesso
615         nomeDaFace = nomeDaFaceBanco;
616
617         System.out.println("  RECONHECIDO: " + nomeDaFace + " (" + distStr + "↩
    )");
618         System.out.println("  foto: " + result.getMatchFileName());
619         pai.setRecogName(nomeDaFace, distStr);
620         return true;
621     } else {
622         nomeDaFace = null;
623         pai.setRecogName("", "");
624         System.out.println("RESULTADO DUVIDOSO: " + nomeDaFaceBanco + " (" + ↩
    distStr + ")");
625         System.out.println("  foto: " + result.getMatchFileName());
626
627         return false;
628     }
629 }
```

Caso a menor ditância calculada seja menor do que a constante *MIN_DIST*, o resultado é considerado sucesso e os dados correspondentes a este são passados como parâmetro para a interface (linha *619*) para que seja exibido. Tando em caso de sucesso ou de erro, um registro do resultado juntamente com o caminho da imagem correspondente e a distância resultante é realizado (linhas *617*, *618*, *624* e *625*) para futura análise dos resultados, que são apresentados no Capítulo 5.

4.7.6 Interface do Sistema

A Figura 29 ilustra a interface produzida pela implementação das classes do pacote "gui", e a Figura 30 representa o sistema em funcionamento, mostrando o fluxo de quadros da câmera na janela do sistema e detectando uma face.

FIGURA 29 – Imagem da interface do sistema.

FONTE: Elaborado pelo autor.

Neste capítulo foi apresentado a implementação do código, focando no fluxo de processo do sistema ilustrado e a implementação do algoritmo *EigenFaces*, detalhando suas fases de treinamento e reconhecimento, e a definição do resultado como sucesso ou falha.

No próximo capítulo será apresentado a maneria de como foram controladas as constantes (definidas na Subseção 4.7.1) nas sessões de testes e seus efeitos como resultados deste experimento.

FIGURA 30 – Imagem do Sistema em Funcionamento.

FONTE: Elaborado pelo autor.

5 TESTES E ANÁLISE DOS RESULTADOS

Este capítulo tem como finalidade apresentar os testes realizados através da manipulação das constantes definidas na Subseção 4.7.1, seus resultados e posteriormente, a análise dos mesmos.

5.1 AMBIENTE DE TESTES

Os testes foram realizados no local de trabalho do autor e os objetos de testes foram as faces dos funcionários do escritório do mesmo, dentro os quais a participação variava diariamente entre 6 a 10 participantes, incluindo o autor. Tomou-se cuidado para que as faces relacionadas fossem sempre as mesmas. Ou seja, a "face 1" seria a mesma face em todos os dias de treino.

Os equipamentos (*hardwares*), descritos na Subseção 3.1.1, foram postos em um local de saída dos funcionários do escritóio, onde as condições luminosas eram sempre as mesmas.

A interface do sistema, como se pode ver na Figura 30, dispõe de uma máscara que indica o local em que a face deve ser posicionada corretamente em frente a câmera, e a câmera posicionada no mesmo local em todos os testes. A cada dia, caso o resultado de correspondência fosse considerada ruim (muito acima do valor mínimo *MIN_DIST*), ou caso o objeto de teste tenha se movido no momento do treino interrompendo o processo, a face poderia ser treinada novamente, o que acumularia as faces de treino do objeto de teste.

Os usuários foram sujeitos ao teste sempre na mesma ordem. As faces dos objetos de testes foram expostas durante o tempo necessário para o sistema coletar a quantidade estabelecida pela constante *NUM_FACES_TREINO*. O sistema foi configurado para faser cerca de 3 a 4 deteções por segundo.

Os testes documentados foram realizados em 5 (cinco) dias e divididos em três fases: duas fases de dois e uma fase de um dias. Para cada fase uma configuração distinta das constantes de controle contempladas na Subseção 4.7.1 foram definidas, as quais são detalhadas na seção seguinte.

Apresentam-se os resultados dos testes do sistema nas condições de cada fase em tabelas: uma para cada dia de teste. Todas apresentam as contagens das quantidades de reconhecimento realizadas por pessoa, bem como a nota média das distâncias correspodentes aos resultados de cada reconhecimento, tanto na falha quanto no sucesso, e a contagem de falsos positivos e falsos negativos.

A contagem dos resultados falsos positivos e falsos negativos foram feitos a olho nú e devidamente registrados pelo analista, comparando os resultados registrados pelo sistema no momento dos testes, como descreve a Subseção 4.7.5.

5.2 TESTES E RESULTADOS

Esta sessão de testes, foi dividida em três fases: uma para cada configuração das constantes de controle. A finalidade de cada constante é contemplada na Subseção 4.7.1 e suas interações com o código descritas durante Seção 4.7.

5.2.1 Fase 1

Para a primeira fase da *sprint* testes, feita em dois dias, os valores das constantes de controle foram configuradas como define a lista abaixo.

- **NUM_FACES_TREINO = 2**
  - esta constante é resposável por controlar o número de faces a ser usado no treinamento, foi configurado para o valor 2 (dois) esperando-se manter um número reduzido de faces para treinamento, e consequentemente do tamanho do espaço multidimensional a ser gerado;
- **NUM_EF_recog = 5**
  - esta constante é resposável por controlar o número de *eigenfaces* ou autovetores a serem usadas no reconhecimento, foi configurado para o valor 5 (cinco), para se observar os resultados iniciais do experimento;
- **MIN_DIST = 0.433**
  - esta constante define se a distância encontrada na fase de reconhecimento pode ser considerada um resultado de sucesso ou falha, foi definida para o valor 0.433, para se observer os resultados iniciais.

Estes valores iniciais foram definidos pelo desenvolvedor a partir de testes unitários não documentados em *sprints* anteriores durante o desenvolvimento do sistema, onde os objetos de teste eram a face do próprio desenvolvedor, voluntários esporádicos ou imagens de faces.

Os resultados do primeiro dia de teste desta fase é descrito na Tabela 3. Neste dia houveram 18 imagens de treinamento coletadas, com destaque para a "face 5", que por alguma razão, talvez sua pele muito branca e oleosa, refletia as luzes do ambiente dificultando o processo e apresentando maus resultados, e assim foi efetuada a coleta de mais imagens de treino deste usuário.

Os resultados do segundo dia de teste são apresentados na Tabela 4. O esperado era que os usuários fossem reconhecidos com resultados parecidos. Observa-se que a "face 2 e 3" obtiveram maus resultados e tiveram que ser treinadas, possivelmente por faserem uso de maquiagem. A "face 5" continuou obtendo os maus resultados de anteriormente. Mais 4 faces foram adicionadas, acumulando um total de 22 imagens usadas para o treinamento.

TABELA 3 – Resultado dos testes (Fase 1 - Primeiro dia)

		1	2	3	4	5
Objetos de teste	qt. imagens treino geradas	qt. reconhecimentos	distância média (sucesso)	qt. sucesso / Falso positivo	distância média (falha)	qt. falha / Falso negativo
Face 1	2	20	0.392	14 / 0	0.895	6 / 6
Face 2	2	30	0.403	19 / 3	0.65	11 / 8
Face 3	4	20	0.429	16 / 4	0.459	4 / 3
Face 4	2	20	0.431	13 / 4	0.527	7 / 4
Face 5	6	40	0.428	12 / 8	0.951	28 / 15
Face 6	2	20	0.427	10 / 2	0.559	10 / 8
Totais	18	150				

FONTE: Elaborado pelo autor.

TABELA 4 – Resultado dos testes (Fase 1 - Segundo dia)

		1	2	3	4	5
Objetos de teste	qt. imagens treino geradas	qt. reconhecimentos	distância média (sucesso)	qt. sucesso / Falso positivo	distância média (falha)	qt. falha / Falso negativo
Face 1	0	20	0.422	12 / 6	0.895	8 / 5
Face 2	2	40	0.419	13 / 5	0.508	27 / 15
Face 3	2	40	0.42	16 / 4	0.459	24 / 8
Face 4	0	20	0.399	11 / 4	0.445	9 / 4
Face 5	0	20	0.418	2 / 2	0.746	18 / 12
Face 6	0	20	0.431	11 / 4	0.479	9 / 4
Totais	22	160				

FONTE: Elaborado pelo autor.

5.2.2 Fase 2

Para a segunda fase da *sprint* testes, feita em dois dias, os valores das constantes de controle foram configuradas como define a lista abaixo.

- **NUM_FACES_TREINO = 1**
  - esta constante é resposável por controlar o número de faces a ser usado no treinamento, foi configurado para o valor 1 (um) esperando-se reduzir ainda mais o espeço (em 50%) para observar se diminui o numero de falsos positivos;
- **NUM_EF_recog = 2**
  - esta constante foi configurado para o valor 2 (dois), para priozirar as *eigenfaces* com mais probabilidade de ter atributos distintos entre as faces de treinamento, esperando-se observar um numero menor de falsos positivos
- **MIN_DIST = 0.55**
  - esta constante foi definida para o valor "0.55", esperando-se obervar uma melhor taxa de sucesso no reconhecimento.

Os resultados do primeiro dia de teste desta fase é descrito na Tabela 5. Neste dia coletou-se imagens de treino apenas de novos usuários, acumulando um total de 28 imagens de treino.

TABELA 5 – Resultado dos testes (Fase 2 - Primeiro dia)

		1	2	3	4	5
Objetos de teste	qt. imagens treino geradas	qt. reconhecimentos	distância média (sucesso)	qt. sucesso / Falso positivo	distância média (falha)	qt. falha / Falso negativo
Face 1	0	20	0.459	13 / 4	0.562	7 / 3
Face 2	0	20	0.52	11 / 4	0.651	9 / 7
Face 3	0	20	0.539	9 / 1	0.629	11 / 5
Face 4	0	20	0.435	13 / 4	0.58	7 / 4
Face 5	0	20	0.418	5 / 4	0.89	15 / 13
Face 6	-	-	-	-	-	-
Face 7	2	40	0.52	22 / 13	0.779	18 / 10
Face 8	1	20	0.479	12 / 4	0.6	8 / 4
Face 9	1	20	0.549	15 / 6	0.646	5 / 3
Face 10	2	40	0.543	10 / 4	0.72	30 / 20
Totais	28	180				

FONTE: Elaborado pelo autor.

Os resultados do segundo dia de teste desta fase é descrito na Tabela 6. Neste dia coletou-se uma imagem de cada usuário, acumulando um total de 35 imagens de treino. Foram feitos mais iterações de reconhecimento no usuário "face 5" pois este continuara a apresentar resultados maus (valores muito altos).

TABELA 6 – Resultado dos testes (Fase 2 - Segundo dia)

		1	2	3	4	5
Objetos de teste	qt. imagens treino geradas	qt. reconhecimentos	distância média (sucesso)	qt. sucesso / Falso positivo	distância média (falha)	qt. falha / Falso negativo
Face 1	1	20	0.501	12 / 5	0.582	8 / 5
Face 2	-	-	-	-	-	-
Face 3	1	20	0.49	12 / 8	0.629	8 / 3
Face 4	-	-	-	-	-	-
Face 5	1	40	0.548	9 / 7	0.759	31 / 25
Face 6	-	-	-	-	-	-
Face 7	1	20	0.51	10 / 4	0.779	10 / 5
Face 8	1	20	0.541	9 / 7	0.631	11 / 5
Face 9	1	20	0.516	12 / 8	0.646	8 / 6
Face 10	1	20	0.49	11 / 6	0.622	9 / 6
Totais	35	160				

FONTE: Elaborado pelo autor.

5.2.3 Fase 3

Para a terceira e últimas fase com apenas uma oportunidade para teste, as faces de treino foram deletadas para que se comece o teste com o espaço *eigenspace* limpo/ Os valores das constantes de controle foram configuradas como define a lista abaixo.

- **NUM_FACES_TREINO = 7**
  - esta constante é resposável por controlar o número de faces a ser usado no treinamento, foi configurado para o valor 7 (sete) nesta fase, obter um valor maior de *eigenfaces* geradas;
- **NUM_EF_recog = 1**

- esta constante foi configurado para o valor 1 (um), para descartar o máximo de *eigenfaces* possível, afim de diminuir o número de falsos positivos;

- **MIN_DIST = 0.69**
  - esta constante foi definida para o valor "0.69", pois ao observar o teste anterior, analisou-se que em geral os resultados estavam piorando a medida que o número de imagens de treno aumentavam, e em muitos casos haviam muitos falsos negativos que poderiam ser confimação de sucesso.

Os resultados desta fase de teste são apresentos na Tabela 7. Neste dia coletou-se 14 imagens de cada usuário, acumulando um total de 98 imagens de treino. Foram feitos mais iterações de reconhecimento para se ter uma melhor amostragem.

TABELA 7 – Resultado dos testes (Fase 3)

		1	2	3	4	5
Objetos de teste	qt. imagens treino geradas	qt. reconhecimentos	distância média (sucesso)	qt. sucesso / Falso positivo	distância média (falha)	qt. falha / Falso negativo
Face 1	14	40	0.552	34 / 0	0.72	6 / 6
Face 2	-	-	-	-	-	-
Face 3	14	40	0.61	28 / 3	0.76	12 / 8
Face 4	-	-	-	-	-	-
Face 5	14	40	0.68	13 / 4	1.11	27 / 18
Face 6	-	-	-	-	-	-
Face 7	14	40	0.6	20 / 9	0.757	20 / 12
Face 8	14	40	0.566	12 / 9	0.703	28 / 22
Face 9	14	40	0.516	18 / 10	0.754	22 / 10
Face 10	14	40	0.658	11 / 7	0.695	29 / 23
Totais	98	280				

FONTE: Elaborado pelo autor.

5.3 ANÁLISE DOS RESULTADOS

Ao decorrer dos testes e se analisar os resultados iniciais, pode-se perceber logo de início que a técnica é extremamente sensível a luzes/sombras, poses e expressões. Algo que pode variar muito a reflexão da luz na face do usuário é a cor da pele, oleosidade, maquiagem e acessórios como boné ou chapéu. Um mínimo movimento no momento da coleta das imagens de treino pode causar uma grande diferença no resultado.

Como se pode verificar na Tabela 8, na primeira fase foram feitos 310 reconhecimentos sendo que 149 foram considerados pelo sistema como sucesso. Destes, apenas 103 foram de fato equivalentes, apresentando a taxa real de reconhecimento com *33.22%* de sucesso. Dos reconhecimentos considerados como falha somaram o total de 161, dos quais 92 eram falsos negativos, pois se tratava de uma real equivalência. Sendo assim, 55% das falhas deveriam ser reconhecimento de sucesso.

Observou-se, que dependendo da face e suas condições, as médias das distâncias euclidianas poderiam variar forma imprevisível. Algumas faces tinham bons resultados

TABELA 8 – Taxas de sucesso reais dos reconhecimentos nas fases de teste.

	Total Reconhecimentos	sucesso real (sucesso – falsos positivos)	Taxa sucesso (%)	falhas	Taxa falha (%)
1a Fase	310	103	33.22	161	51.93
2a Fase	340	96	28.23	195	57.35
3a Fase	280	94	33.57	144	51.42

FONTE: Elaborado pelo autor.

de reconhecimento enquanto outras os apresentavam péssimos, em relação à constante *MIN_DIST*. O grande número de falsos negativos apresentados incentivaram a reconfiguração da constante *MIN_DIST* para um valor maior, *NUM_FACES_TREINO* foi reduzido para que o espaço não sofresse muita adição de tamanho, *NUM_EF* foi reduzido para priorizar *eigenfaces*, e apenas treinos de novas faces foram realizadas, esperando-se que os resultados de reconhecimento melhorassem ou se mantivessem.

Na segunda fase foram feitos 340 reconhecimentos sendo que 185 foram considerados pelo sistema como sucesso. Destes, 96 foram de fato equivalentes. Sendo assim, apresentando a taxa real de reconhecimento de *28.3%* de sucesso. Dos reconhecimentos considerados como falha, que somaram o total de 195, 124 se tratavam de uma real equivalência. Sendo assim 63.5% das falhas deveriam ser reconhecimento de sucesso.

Observa-se que,os resultados da segunda fase foram piores que o da anterior, porém as contagens de falsos positivos e negativos ainda se apresentavam elevadas. Reconfigurou-se então as variáveis como conta na Subseção 5.2.3 levando em conta lógica de que se um maior número de possível da coleta de imagens de uma face para o treino, combinado com o descarte de auto-vetores por auto-valores, "filtraria" melhor as caracteristicas da face e suas variáveis (sombras e reflexoes, movimentos, feiçoes ou enquadramento da face) melhor registradas. A constante *MIN_DIST* também sofreu adição de valores para que se diminuíse ainda mais os falsos negativos que poderiam ser reconhecimentos de sucesso, com cautela para não provocar resultados falsos positivos.

Segundo os resultados da terceira fase, foram feitos 280 reconhecimentos sendo que 136 foram considerados pelo sistema como sucesso. Destes, apenas 94 foram de fato equivalentes, apresentando a taxa real de reconhecimento com *33.57%* de sucesso. Dos reconhecimentos considerados como falha (total de 144), 99 se tratava de uma real equivalência. Sendo assim cerca de 50% das falhas deveriam ser reconhecimento de sucesso.

Observou-se então uma melhora considerável nos resultados, levando em conta que o espaço de eigenfaces era muito maior relacionada às duas primeiras fases, porém ainda é um valor que não apresenta precisão e confiabilidade.

Não obstante, desconsiderando a constante *MIN_DIST* e analisando apenas a contagem de resultados de sucesso (com falsos positivos já subtraídos) somandos com a contagem dos falsos negativos, verificamos que na maioria das vezes o algoritmo *Eigenfaces* conseguiu equivalência de sucesso, como mostram os resultados dos testes apresentados.

Os Gráfico 2 e Gráfico 3 apresentam dois sub-gráficos "A" e "B": os gráficos "A" são às contagens de reconhecimentos de sucesso (eixo Y) a medida em que a quantidade

GRÁFICO 2 – **Gráfico "A"**: Reconhecimentos de sucesso (eixo Y) por quantidade de imagens usadas para o treino (eixo X). **Gráfico "B"**: Falhas no reconhecimentos (eixo Y) por quantidade de imagens usadas para o treino (eixo X). Referentes as fases de teste "1"e "2".

FONTE: Elaborado pelo autor.

GRÁFICO 3 – **Gráfico "A"**: Reconhecimentos de sucesso (eixo Y) por quantidade de imagens usadas para o treino (eixo X). **Gráfico "B"**: Falhas no reconhecimentos (eixo Y) por quantidade de imagens usadas para o treino (eixo X). Referentes as fases de teste "3".

FONTE: Elaborado pelo autor.

de imagens de treino cresce (eixo X), e os gráficos "B", são as contagens de falhas de reconhecimento a medida em que as imagens de treinamento crescem. O Gráfico 2 se refere aos resultados da fase "1" e "2", e o Gráfico 3 é referente à fase "3". Foram divididos desta forma pelo fato de quem o espaço foi reiniciado no terceiro dia, como descreve a Subseção 5.2.3.

6 CONCLUSÕES E SUGESTÕES PARA FUTUROS TRABALHOS

6.1 CONCLUSÕES

De acordo com a análise executada e apresentada no Seção 5.3, conclue-se que nos dois gráficos "A", as linhas de quantidade de sucesso e falsos positivos tendem a se encontram enquanto que nos gráficos "B" o contrário acontece: as linhas de quantidade de falhas e de falsos negativos tendem a se afastar, apesar de não estar tão evidenciado Gráfico 2 "B" podendo-se concluir que à medida em que o espaço multidimensional de *eigenfaces* (*eigenspace*) cresce, o método ACP apresenta distâncias falsas e verdadeiras similares, imprecisando seu veridito final.

Também pode-se concluir que o sistema desenvolvido não pode ser considerado eficaz como um sistema de reconhecimento biométrico de digitais, por exemplo, sendo incapaz, por si só, de identificar o usuário com precisão. Todavia, poderia ser utilizado como uma ferramenta de auxílio, já que de 3 reconhecimentos, pelo menos 1, aproximadamente, apresentou ser verdadeiro.

Todos os conceitos abordados no trabalho aqui proposto e a experiência proporcionada pela pesquisa e pelo desenvolvimento do sistema em suas *sprints* foram considerados essenciais para conclusão de seus objetivos tal como para evolução técnica e acadêmica do autor.

De acordo com os resultados observados na execução deste trabalho e de outros, tal como novas técnicas que estão sendo criadas atualmente, pode-se especular que o problema de reconhecimento de faces não poderá ser resolvido com apenas um algoritimo ou outro, mas sim com a cooperação e integração de vários algoritimos e métodos.

6.2 SUGESTÕES PARA FUTUROS TRABALHOS

Este trabalho e todo o código de sua implementação, como consta nos objetivos, será aberto para o público desenvolvedor e acadêmico que tiver interesse em abordar o assunto.

Lista-se abaixo algumas sugestões de trabalhos futuros que podem ser feitos a partir deste:

- **Apicação de outras métricas de distância na Análise de Componentes Principais**
  - como contempla a Subseção 2.3.1, é possível se usar outras medidas de distância no lugar da Distância Euclidina, como a distância de Manhatan ou de Mahalanobis;
- **Definição de Sucesso do Reconhecimento**

- como pode-se constatar na Subseção 4.7.5 e no Capítulo 5, a maniulação de constantes pode ter alta influência sobre o veredito final do reconhecimento. Em especial, a manipulaçào da constante *MIN_DIST*, que limita a distancia máxima para que uma equivalência seja considerada verdadeira, pode ser feita de diversas formas. O sistema pode ser evoluido com uma sub-rotina que itera vários reconhecimentos de uma face analisa os resultados de n distâncias, ao contrário de com foi implementado;

- **Melhorias no Treinamento**
  - o treinamento sendo parte essencial do processo, pode ser evoluido atravéz melhores técnicas processamentos de imagens, ou por manipulação de constantes, por exemplo, descartando imagens treinadas que tem maus resultados subsequentes de reconhecimento, forçando, automaticamente, um retreinamento.

REFERÊNCIAS

ARAUJO, A. D. Manifesto Ágil e as top 3 metodologias Ágeis de desenvolvimento! BeCode, 2016. Disponível em: <https://becode.com.br/manifesto-agil-e-top-metodologias-ageis/>. Acesso em: 06.2018.

BALAN, W. C. A imagem digital. **Espaço para apoio didático do Curso de Comunicação social - Radialismo**, 2009. Disponível em: <http://willians.pro.br/textos/>. Acesso em: 05.2018.

BARBU NATHAN LAY, G. G. A. Face detection with a 3d model. 2015. Disponível em: <https://pdfs.semanticscholar.org/2bbf/047993f08a608ce4ca22720d374d4cf76f98.pdf>.

BYTEDECO. Javacpp - the missing bridge between java and native c++ libraries. 2018. Disponível em: <https://github.com/bytedeco/javacpp-presets>. Acesso em: 05.2018.

BYTEDECO. Javacv - java interface to opencv, ffmpeg, and more. 2018. Disponível em: <https://github.com/bytedeco/javacv>. Acesso em: 05.2018.

CASTRO, V. A. Introdução ao desenvolvimento Ágil. DevMedia, 2018. Disponível em: <https://www.devmedia.com.br/introducao-ao-desenvolvimento-agil/5916>. Acesso em: 06.2018.

CHAO, W.-L. **Face Recognition**. Dissertação (Mestrado) — National Taiwan University, GICE, Taiwan, Maio 2011. Disponível em: <http://disp.ee.ntu.edu.tw/~pujols/Face%20Recognition-survey.pdf>.

COLT. 2018. Disponível em: <http://dst.lbl.gov/ACSSoftware/colt/>. Acesso em: 05.2018.

DAVISON, D. A. Java prog. techniques for games. Universidade Prince of Songkla, Thailândia, Vienna, 2013. Disponível em: <hhttp://fivedots.coe.psu.ac.th/~ad/index.html>.

ELECTRICAL, I. of; IEEE, E. E. Face recognition: eigenface, elastic matching, and neural nets. **Proceedings of the IEEE (Volume: 85, Issue: 9, Sep 1997)**, 1997. Disponível em: <http://ieeexplore.ieee.org/document/628712/>.

EYMAR LAERCIO, R. Teoria : Processamento de imagens. **Instituto Nacional de Pesquisas Espaciais - INPE**, 2018. Disponível em: <http://www.dpi.inpe.br/spring/teoria/realce/realce.htm>. Acesso em: 05.2018.

FACHIN, O. Fundamentos de metodologia - 6a ed. 2017. Saraiva, 2017.

FARIAS, F. Como compactar imagens sem perder qualidade e aumentar a velocidade de seu site. **Resultados Digitais**, 2017. Disponível em: <https://resultadosdigitais.com.br/blog/compactar-imagens-sem-perder-qualidade/>. Acesso em: 05.2018.

GIASSA, M. Upsampling and interpolation. GIASSA.NET, 2009. Disponível em: <http://www.giassa.net/?page_id=200>. Acesso em: 06.2018.

GIT. 2018. Disponível em: <https://git-scm.com/>. Acesso em: 05.2018.

GONZALEZ, R. C. W. R. C. In: _____. **Processamento Digital de Imagens 3ªed**. São Paulo: Pearson Prentice Hall, 2010.

GROFFE, R. J. Artigo modelagem de sistemas através de uml: uma visão geral. Devmedia, 2013. Disponível em: <https://www.devmedia.com.br/modelagem-de-sistemas-atraves-de-uml-uma-visao-geral/27913>. Acesso em: 06.2018.

INTRONA, H. N. L. D. Facial recognition technology - a survey of polycy and implementation issues. 2010. Disponível em: <https://www.nyu.edu/projects/nissenbaum/papers/facial_recognition_report.pdf>.

JEBARA, T. S. **3D Pode Estimation and Normalization for Face Recognition**. Dissertação (Mestrado) — McGill University, Columbia University, Montréal, Québec - Canada, 1995. Disponível em: <http://www.cs.columbia.edu/~jebara/htmlpapers/UTHESIS/node7.html>.

JOHNSON, S. In: _____. **Stephen Johnson on Digital Photography**. [S.l.]: O'Reilly Media, Incorporated, 2006.

KIM JOON HYUNG SHIM, J. Y. I. **Final Project - Face Detection Project**. Dissertação (Mestrado) — Stanford University, Stanford, California, 2003. Disponível em: <https://web.stanford.edu/class/ee368/Project_03/Project/reports/ee368group02.pdf>.

KITANI, C. E. T. E. C. Um tutorial sobre análise de componentes principais para o reconhecimento automático de faces. **Departamente de Engenharia Elétrica**, Centro Universitário da FEI, 2018. Disponível em: <http://fei.edu.br/~cet/Tutorial_ReconhecimentoFaces.pdf>. Acesso em: 05.2018.

MELO, G. Processamento digital de imagens. UFRN, 2016. Disponível em: <https://melosgabriel.github.io/>. Acesso em: 06.2018.

MOREIRA, G. C. G. **Reconhecedor de Objetos em Vídeos Digitais para Aplicações Interativas**. Dissertação (Mestrado) — Pontifícia Universidade Católica do Rio de Janeiro - PUCRJ, Rio de Janeiro, 2008. Disponível em: <https://www.maxwell.vrac.puc-rio.br/13069/13069_1.PDF>.

MURTA, L. O. Interpolação em imagens. FFCLRP/USP, 2013. Disponível em: <http://dcm.ffclrp.usp.br/~murta/PIM/PIM_10.pdf>. Acesso em: 06.2018.

NOGUEIRA, R. C. de A. **Análise de Conversão de Imagem Colorida para Tons de Cinza Via Contraste Percebido**. Dissertação (Mestrado) — Universidade Federal de Pernambuco - Centro de Engenharia da Computação, Recife - Pernambuco, 2016. Disponível em: <http://www.cin.ufpe.br/~tg/2016-1/rcan.pdf>.

OPENCV. Opencv - open source computer vision. 2018. Disponível em: <https://opencv.org/about.html>. Acesso em: 05.2018.

PERRY, J. S. Aprenda: Tecnologia java. IBM - Developer Works, 2016. Disponível em: <https://www.ibm.com/developerworks/br/java/tutorials/j-introtojava1/index.html>. Acesso em: 06.2018.

PORTAL, S. T. S. Face recognition software comparison as recognition. **Digital Economy Compass 2017**, 2017. Disponível em: <https://www.slideshare.net/statista_com/statista-digital-economy-compass-2017>.

ROELOFS, G. A basic introduction to png features. **PNG LIB**, 2017. Disponível em: <http://www.libpng.org/pub/png/pngintro.html>. Acesso em: 05.2018.

SILVA, G. N. da. **Estudo das Técnicas PCA (Análise de Componentes Principais) e Auto-faces Aplicadas ao Reconhecimento de Faces Humanas**. Dissertação (Mestrado) — Centro Universitário Eurípides Soares - UNIVEM, Marília, 2009. Disponível em: <http://aberto.univem.edu.br/bitstream/handle/11077/284/Estudo+da+t%E9cnica+PCA+(An%E1lise+de+Componentes+Principais)+e+Auto-faces+aplicadas+ao+reconhecimento+de+faces+humanas.pdf;jsessionid=2D900F25381260DCCB85EC832736A216?sequence=1>.

SMITH ARUN ROSS, D. C. K. Face recognition. **National Science and Technology Consil - NSTC**, NSTC, p. 92–99, 2009. Disponível em: <https://www.hsdl.org/?view&did=463904>. Acesso em: 04.2018.

SMITH, L. I. A tutorial on principal components analysis. 2002. Disponível em: <http://www.cs.otago.ac.nz/cosc453/student_tutorials/principal_components.pdf>. Acesso em: 05.2018.

SMITH, L. I. Vídeo digital. 2018. Disponível em: <https://br.ccm.net/contents/739-video-digital>. Acesso em: 05.2018.

TECH-ALGORITHM. Nearest neighbor image scaling. 2007. Disponível em: <http://tech-algorithm.com/articles/nearest-neighbor-image-scaling/>. Acesso em: 06.2018.

VELASQUEZ, H. C. Advanced vision module. **The University of Edinburgh**, 2017. Disponível em: <http://www.inf.ed.ac.uk/teaching/courses/av/index.html>. Acesso em: 05.2018.

VERGE.COM, T. • baidu swaps tickets for facial recognition in historic chinese 'water town'. novembro 2016. Disponível em: <https://www.theverge.com/2016/11/17/13663084/baidu-facial-recognition-park-wuzhen>.

WANGENHEIM, D. A. von. Transformações geométricas 2d e coordenadas homogêneas. **Image Processing and Computer Graphcs Lab - LAPiX**, Universidade Federal de Santa Catarina – UFSC, 2017. Disponível em: <http://www.lapix.ufsc.br/ensino/computacao-grafica/transformacoes-geometricas-2d-e-coordenadas-homogeneas>. Acesso em: 05.2018.

WOJCIK KONRAD GROMASZEK, M. J. W. Face recognition: Issues, methods and alternative applications. 2016. Disponível em: <https://www.intechopen.com/books/face-recognition-semisupervised-classification-subspace-projection-and-evaluation-methods/face-recognition-issues-methods-and-alternative-applications>.

YANG PING LUO, C. C. L. e. X. T. S. Wider face: A face recognition benchmark. 2015. Disponível em: <https://arxiv.org/pdf/1511.06523.pdf>.