UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO DE JOINVILLE
CURSO DE ENGENHARIA MECATRÔNICA

MATHEUS LUIZ ANDERLE DE SOUZA

UM ESTUDO DE TÉCNICAS DE DETECÇÃO DE OLHOS UTILIZANDO VISÃO COMPUTACIONAL

Joinville
2019

MATHEUS LUIZ ANDERLE DE SOUZA

UM ESTUDO DE TÉCNICAS DE DETECÇÃO DE OLHOS UTILIZANDO VISÃO COMPUTACIONAL

Trabalho de Conclusão de Curso apresentado como requisito parcial para obtenção do título de Bacharel em Engenharia Mecatrônica, no curso Engenharia Mecatrônica da Universidade Federal de Santa Catarina, Centro Tecnológico de Joinville.

Orientador: Prof. Me. Benjamin Grando Moreira

Joinville
2019

RESUMO

Todo ano, um número significativo de motoristas se envolvem em acidentes no trânsito por problemas relacionados à fadiga e sonolência. Neste sentido, é importante investigar técnicas que auxiliem na prevenção desse tipo de acidente. A área de Visão Computacional permite a análise de imagens para a extração de informações com relação a essas imagens podendo, por exemplo, extrair aspectos visuais com relação a um motorista objetivando prevenir possíveis problemas associados. Levando isso em consideração, este trabalho estudou técnicas de detecção de olhos, para identificar se os olhos estão abertos ou fechados, etapa que pode ser utilizada para posterior identificação de sonolência em motoristas. Os algoritmos desenvolvidos utilizam as técnicas de detecção Viola e Jones, Histogramas de Gradientes Orientados e Transformada de Hough. Os algoritmos foram feitos em linguagem de programação Python utilizando as bibliotecas OpenCV e Dlib. Este trabalho faz a análise dos resultados alcançados e compara o desempenho das técnicas entre si levando em consideração custo computacional, uso de memória e acurácia na detecção.

Palavras-chave: Detecção Facial. Detecção de Olhos. Visão Computacional.

ABSTRACT

Every year, a significant number of drivers get involved in traffic accidents due to problems related to fatigue and drowsiness. Due to this, it is important to investigate techniques that help prevent this type of accident. The Computer Vision field enables the analysis of images to extract information regarding these images and, for example, extract visual aspects regarding a driver in order to prevent possible associated problems. Taking this into consideration, this paper has studied eye detection techniques to identify whether the eyes are open or closed, a step that can be used for later identification of drowsiness in drivers. The developed algorithms use Viola and Jones, Hough Transform and Histograms of Oriented Gradients detection techniques. The algorithms were made in Python programming language using the OpenCV and Dlib libraries. This paper analyzes the achieved results and compares the performance of the techniques against each other taking into account computational cost, memory usage and detection accuracy.

Keywords: Facial Detection. Eye Detection. Computer Vision.

AGRADECIMENTOS

Ao professor Benjamin, pela constante ajuda e orientação ao longo desse trabalho, eu não poderia ter pedido por um orientador melhor.

Aos meus pais e ao meu irmão, pelo apoio e paciência ao longo dos anos. Eles estiveram ao meu lado e me deram suporte durante todas as decisões mais importantes da minha vida, então essa conquista também é de vocês.

À minha namorada, Patrícia, pelo apoio, carinho e por ter estado sempre ao meu lado por todos esses anos. Sem você, eu com certeza não teria chegado tão longe.

Aos meus amigos, obrigado por ajudar a balancear a minha vida acadêmica com a minha vida pessoal. Sempre me senti confortável para compartilhar minhas alegrias, tristezas, pedir conselhos ou apenas jogar papo fora.

Também gostaria de agradecer aos professores da banca avaliadora por terem dedicado seu tempo para avaliar este trabalho e terem me dado a oportunidade de apresentá-lo.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Aplicações práticas de técnicas de detecção facial estão cada vez mais presentes na sociedade moderna, como sua utilização em sistemas de câmeras digitais ou então seu uso em conjunto com ferramentas de reconhecimento facial para desbloquear celulares. Por conta disso, são de importância e necessários estudos de métodos e algoritmos que aperfeiçoem soluções já existentes ou então proponham novos cenários que melhorem o desempenho desses sistemas.

Com o constante desenvolvimento da indústria automotiva, as empresas estão buscando soluções cada vez mais autônomas para previnir e remediar problemas causados pelos próprios motoristas. Um desses problemas é relacionado ao número de acidentes decorrentes de motoristas que dormem ao volante. Segundo uma estimativa feita pela National Highway Traffic Safety Administration (NHTSA), sonolência foi a causa de 72 mil acidentes, 44 mil lesões e 800 mortes no ano de 2013 nos Estados Unidos (NHTSA, 2018).

Algumas empresas de automóveis possuem ferramentas para evitar este tipo de problema. A Mercedes, por exemplo, implementa uma Assistência de Atenção (*Attention Assistance*) que detecta os movimentos do motorista ao longo do tempo para observar mudanças repentinas de comportamento ou descuidos recorrentes para avaliar se o motorista está ou não sonolento (MERCEDES-BENZ, 2018). Caso o sistema detecte fadiga, uma mensagem é exibida sugerindo que o motorista descanse. Outras empresas, como a Volkswagen e Audi (JAN et al., 2005; AUDI, 2013), também utilizam técnicas similares de detecção do comportamento do motorista.

Uma forma de detectar o comportamento do motorista envolve a análise dos aspectos visuais por meio de uma câmera. Ferramentas de Visão Computacional se mostram cada vez mais eficientes para detectar aspectos visuais e comportamentais de pessoas. Enquanto empresas ainda não utilizam amplamente esses tipos de parâmetros em seus sistemas, estudos (FERASSINI, 2014; MACIONKI; STUMPF, 2009; LUCION, 2017) na área procuram adaptar algoritmos e técnicas computacionais para realizar uma detecção de fadiga e sonolência com eficiência.

A detecção de fatiga e sonolência pode ser feita de diversas formas, mas uma das principais observações parte de detectar os olhos de uma pessoa. Neste trabalho, busca-se realizar a detecção de olhos abertos e fechados de um motorista utilizando os algoritmos de Viola e Jones, Histogramas de Gradientes Orientados e Transformada de Hough para detecção de círculos. Utilizando um banco de imagens próprio para

a análise, os algoritmos realizam a detecção da região de interesse e as diferentes técnicas foram comparadas em relação ao seu uso de processamento e memória, tempo de execução e também acurácia na detecção.

1.1 Objetivo Geral

O objetivo deste trabalho é estudar técnicas de Visão Computacional para detecção de olhos e otimizar parâmetros associados a essas técnicas.

1.2 Objetivos Específicos

Os objetivos específicos são:

a. Pesquisar técnicas que permitam realizar a detecção de olhos abertos e fechados;
b. Pesquisar ferramentas para auxílio na implementação das técnicas de detecção de olhos;
c. Determinar os parâmetros para otimização dos resultados das técnicas identificadas;
d. Comparar desempenho das técnicas pesquisadas levando em consideração o custo computacional e a acurácia na detecção.

1.3 Estrutura do texto

O próximo capítulo apresenta uma análise técnica de estudos e métodos comumente utilizados no desenvolvimento de sistemas de reconhecimento facial, incluindo as três técnicas aplicadas neste trabalho. No terceiro capítulo, é explicado o processo de desenvolvimento dos códigos, abrangendo desde a delimitação da área de interesse até a detecção dos olhos abertos ou fechados. Em seguida, os resultados adquiridos no capítulo quatro são discutidos e, por fim, serão apresentadas conclusões sobre o desenvolvimento do sistema, incluindo sugestões de alterações no método e possíveis ideias para trabalhos futuros.

2 REVISÃO TEÓRICA

Este capítulo tem como objetivo explorar conceitos relacionados à Visão Computacional e, em especial, conceitos relacionados à detecção de faces e olhos. As seções a seguir apresentarão diversos algoritmos utilizados na área.

2.1 Detecção de Faces e Olhos

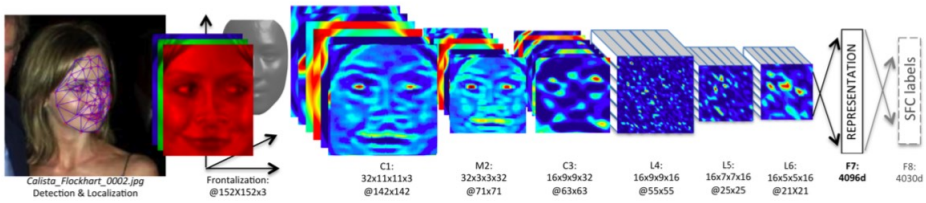
Técnicas de detecção facial são pesquisadas pela área de Visão Computacional há décadas, com os primeiros estudos da área sendo datados da década de 1960 por autores como Woody Bledsoe (BALLANTYNE et al., 1996). Naquela época, os métodos utilizados para realizar o processo de detecção facial envolviam a análise de características simples em uma face, como tamanho de orelhas e comprimento de nariz. No entanto, esses sistemas possuíam diversas limitações, a principal sendo a falta de poder de processamento das máquinas naquela época (BENSON, 2017).

Atualmente, o desenvolvimento de novas técnicas na área é liderado por grandes empresas de tecnologia como Google e Facebook, que utilizam soluções automatizadas para processar grandes bases de dados de seus usuários. O próprio Facebook publicou um artigo em 2014 sobre o DeepFace, um sistema de detecção facial que obteve uma precisão de 97.35% na detecção de rostos em uma base de dados composta por milhões de imagens de seus próprios usuários (TAIGMAN et al., 2014).

Ao invés de analisar as características físicas de uma pessoa, o sistema proposto pelo Facebook utiliza métodos de aprendizado de máquina com base em um grande volume de dados para treinamento. Primeiramente, o sistema delimita o contorno do rosto da pessoa e, em seguida, alinha-o dimensionalmente para que a imagem resultante tenha o formato de um retrato comum. Por fim, realize-se a comparação e identificação final (TAIGMAN et al., 2014), como é demonstrado na Figura 1.

Este tipo de abordagem também é utilizada em determinados modelos de celulares como o iPhone X, da empresa Apple, que detecta o rosto do usuário do aparelho para desbloquear sua tela (APPLE, 2019). Como a limitação de hardware está se tornando um problema cada vez menor, essas aplicações estão cada vez mais frequentes em produtos do dia-a-dia por conta de sua alta eficácia. O DeepFace, por exemplo, obteve um desempenho similar ao de um humano e, quando aplicado à base

Figura 1 – Arquitetura do DeepFace



Fonte: Taigman et al. (2014, p. 4).

de dados do YouTube, reduziu o índice de erro no reconhecimento facial em mais de 50% (TAIGMAN et al., 2014).

O processo para a detecção de olhos é similar. Sistemas como o DeepFace podem realizar seu treinamento utilizando imagens específicas de olhos para aprender a realizar essa detecção, enquanto algoritmos mais antigos como o proposto por Viola e Jones podem utilizar distinções de tonalidades e outras características físicas entre partes do rosto para identificar a região dos olhos (VIOLA; JONES, 2001).

2.2 Algoritmos

Para o desenvolvimento de um sistema de detecção facial e de olhos, é necessário realizar uma análise sobre o funcionamento de diversos algoritmos relacionados à Visão Computacional. Dessa forma, é possível avaliar quais técnicas são mais propícias para o desenvolvimento deste trabalho. Levando isso em consideração, esta seção entrará em detalhes sobre o funcionamento dos algoritmos de Viola e Jones, Histogramas de Gradientes Orientados e detecção de círculos via Transformada de Hough.

2.2.1 Viola e Jones

Um estudo relevante na área de detecção facial foi publicado em 2001 por Paul Viola e Michael J. Jones. Nele, os autores apresentaram um método para a detecção de objetos baseado em aprendizado de máquina utilizando três contribuições principais. A primeira dessas contribuições é a proposição de uma imagem integral que realiza um processamento rápido de determinadas características, a segunda é um algoritmo de aprendizado e a última é a utilização de um método de cascata para descartar com rapidez os resultados indesejados (VIOLA; JONES, 2001).

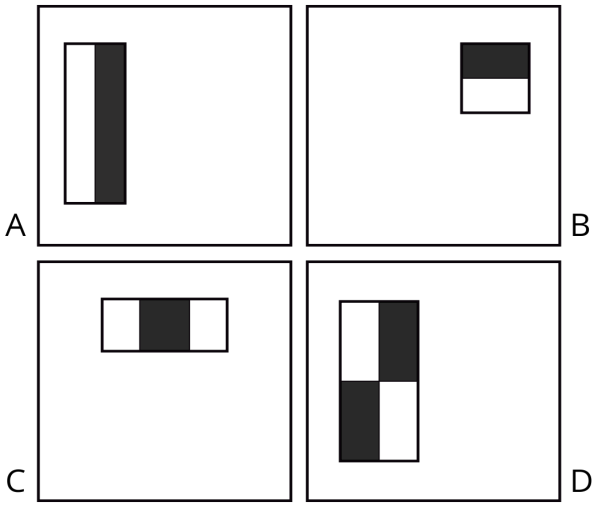
O método de identificação facial proposto por Viola e Jones se destacou por utilizar imagens em preto e branco e não necessitava de imagens adicionais ou cores para realizar a detecção, apenas as características presentes na própria imagem eram necessárias.

2.2.1.1 Imagem Integral

O primeiro passo para o desenvolvimento do algoritmo de Viola e Jones consiste na utilização de imagens integrais, também conhecidas como tabelas de soma de áreas, um algoritmo desenvolvido por Frank Crow em 1984 (CROW, 1984). Imagens integrais conseguem diferenciar características específicas da imagem que está sendo processada baseando-se na soma dos pixels de determinadas regiões da imagem. No caso de um rosto humano, por exemplo, esses classificadores observam diferenças de tonalidades entre objetos da face, distâncias e outras características físicas, conseguindo não apenas detectar rostos, como também diferenciar regiões do rosto, como olhos e boca, caso seja necessário para a aplicação do usuário. A justificativa pela utilização da análise de características ao invés da análise de pixels da imagem se deve ao fato de que existe um número menor de características a serem analisadas em uma imagem se comparado ao número de pixels, resultando em uma velocidade de processamento maior (VIOLA; JONES, 2001).

Dessa forma, Viola e Jones utilizaram três formatos de características em sua técnica, todos baseados em retângulos. O primeiro utiliza dois retângulos e é calculado utilizando a diferença entre a soma dos pixels de duas regiões retangulares que possuem o mesmo tamanho e formato e são adjacentes horizontalmente ou verticalmente. O parâmetro de três retângulos calcula a soma dos dois retângulos exteriores subtraídos da soma do retângulos central. Por fim, o parâmetro de quatro retângulos calcula a diferença entre os pares diagonais de retângulos. Esse conjunto de características está exemplificado na Figura 2.

Figura 2 – Exemplos de imagens integrais.



Fonte: Viola e Jones (2001, p. 3).

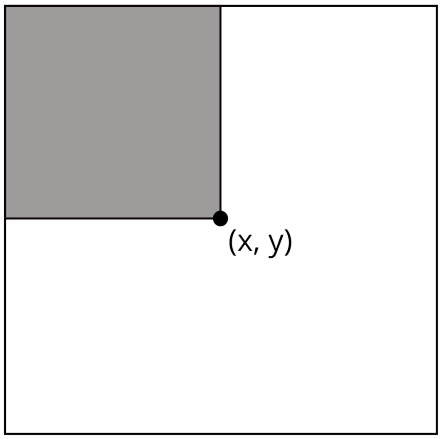
O cálculo das características para a determinação da imagem integral é feito somando os valores dos pixels da região em questão. Esse valor é um número que

representa a intensidade do pixel, que varia de acordo com a região da imagem que está sendo analisada. A área cinza da Figura 3 demonstra a determinação de uma região de interesse, e o valor da imagem integral em questão é representada pela soma dos pixels acima e à esquerda do ponto de coordenada (x, y), como mostrado na equação:

$$ii(x, y) = \sum_{x' \leq x, y' \leq y} i(x', y'), \tag{1}$$

Em que $ii(x, y)$ é a imagem integral e $i(x, y)$ é a imagem original (VIOLA; JONES, 2001).

Figura 3 – Representação de uma Imagem Integral.



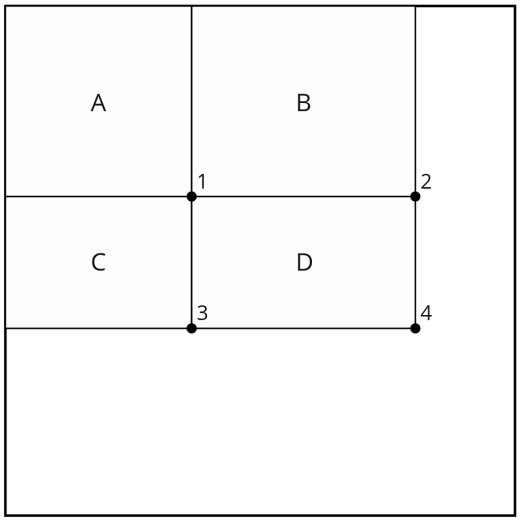
Fonte: Viola e Jones (2001, p. 5).

Por fim, é válido ressaltar que é possível calcular o valor da imagem integral de qualquer área em uma imagem genérica utilizando os quatro vértices da região de interesse. Utilizando como base a Figura 4, o valor do ponto 4 pode ser calculado como a soma dos pixels nas áreas A, B, C e D.

2.2.1.2 AdaBoost

Para selecionar parâmetros importantes para a detecção facial, Viola e Jones se basearam no algoritmo AdaBoost. AdaBoost é um algoritmo de aprendizado de máquina criado por Yoav Freund e Robert Schapire (FREUND; SCHAPIRE, 1996) normalmente utilizado com outros algoritmos para aumentar o desempenho de um sistema realizando uma combinação de classificadores considerados "fracos"para realizar um melhor processo de aprendizado utilizando pesos para cada classificador (FREUND; SCHAPIRE, 1996). Como o número de parâmetros identificados em uma imagem é grande, é necessário diminuir o número de parâmetros que serão analisados e focar em um número pequeno de características essenciais.

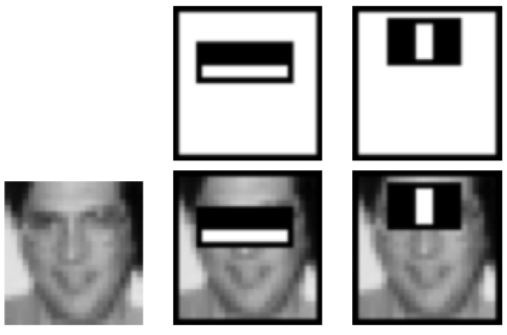
Figura 4 – Divisão de regiões para cálculo de imagens integrais



Fonte: Viola e Jones (2001, p. 11).

Dessa forma, o algoritmo AdaBoost serve como suporte na otimização da performance da técnica de Viola e Jones. A Figura 5 exemplifica a utilização desse processo para a identificação de parâmetros essenciais em um rosto. Ele inicializa a imagem original e começa o processo de detecção de imagens integrais. Na coluna do meio, é calculada a imagem integral entre a região dos olhos e a região das bochechas por meio da diferença de intensidade dos pixels das áreas, enquanto na coluna da direita é calculada uma imagem integral de três retângulos que diferencia o nariz dos olhos.

Figura 5 – Exemplos de classificadores utilizados por Viola e Jones



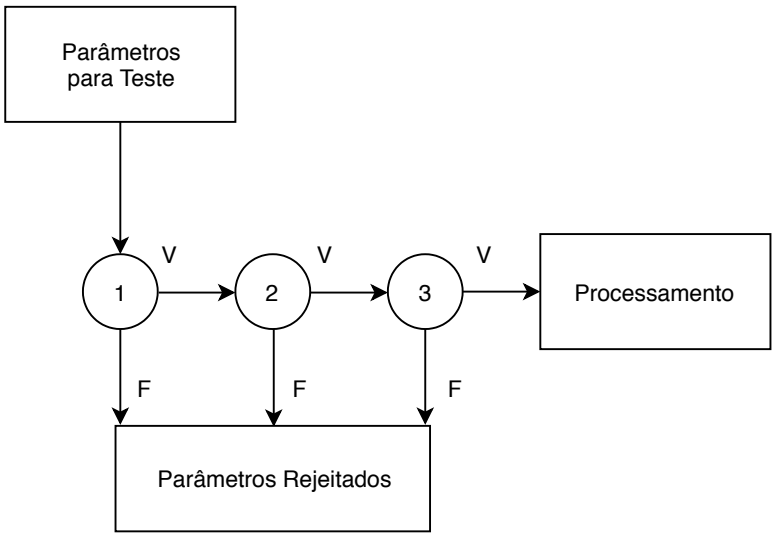
Fonte: Viola e Jones (2001, p. 11).

Além de auxiliar na escolha de características de interesse nas imagens, o AdaBoost também possui um papel no treinamento do classificador ao conseguir encontrar um número reduzido de características relevantes para a detecção (FERASSINI, 2014).

2.2.1.3 Criação dos Classificadores (Cascades)

Para completar o processo de detecção é feita a utilização de uma combinação sucessiva de análises em um formato de cascata, com o sistema processando uma característica crítica de cada vez. Se o primeiro parâmetro for identificado, o sistema segue com análise do próximo parâmetro e assim sucessivamente até um dos parâmetros não ser identificado, o que resulta na rejeição da imagem em questão, ou então até que todos os parâmetros sejam identificados pelo sistema, o que resulta na aprovação da imagem. A Figura 6 exemplifica o fluxo de processamento dos classificadores em uma imagem.

Figura 6 – Fluxo de processamento de um Cascade.



Fonte: Adaptado de Viola e Jones (2001, p. 12).

No caso, uma série de classificadores é aplicado para cada parâmetro. A medida que os teste para a identificação de um determinado parâmetro apresenta um resultado positivo, testes subsequentes são realizados até terminar o processamento da imagem em questão. Por conta disso, o objetivo desse processo é eliminar inicialmente um grande número de exemplos negativos com pouco processamento (VIOLA; JONES, 2001).

2.2.2 Histograma de Gradientes Orientados

Outra abordagem possível para detecção de aspectos faciais se baseia no conceito de Histogramas de Gradientes Orientados (HOGs). As bases para este método foram expostas pela primeira vez por Robert K. McConnell em 1986 durante a aplicação de uma patente (MCCONNELL, 1986), mas o conceito foi popularizado a partir de 2005 com os trabalhos de Navneet Dalal e Bill Triggs (DALAL; TRIGGS, 2005).

2.2.2.1 Gradientes de Imagem

Nesta abordagem, a detecção de objetos e formas é feita pela análise da distribuição da intensidade de gradientes de imagem. O gradiente de uma função de duas dimensões, $f(x, y)$, é definido como o vetor (ACHARJYA et al., 2012):

$$\nabla f = \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} \tag{2}$$

Sendo a magnitude desse vetor representada pela equação:

$$\nabla f = mag(\nabla f) = [g_x^2 + g_y^2]^{1/2} = \left[\left(\frac{\partial f}{\partial x} \right)^2 + \left(\frac{\partial f}{\partial y} \right)^2 \right]^{1/2} \tag{3}$$

Resumidamente, o valor dessa magnitude apresenta diferentes valores dependendo da intensidade da luminosidade do pixel. Em regiões homogêneas, a magnitude do gradiente apresentará valores próximos de zero, e em regiões que apresentam uma diferença de intensidade esse valor deverá ser diferente de zero (LOPES, 2015).

Após o cálculo dos gradientes de imagem, o próximo passo para a implementação desta técnica é a divisão da imagem de teste em pequenas partes, intituladas células, que podem assumir formas retangulares ou radiais. Em seguida, os pixels fazem uma votação baseada nos cálculos de magnitude dos gradientes de imagem para identificação de bordas, e estes votos são então acumulados em cada célula (DALAL; TRIGGS, 2005), resultando na criação de histogramas de gradientes.

2.2.2.2 Blocos de Descritores e Normalização

Diferenças de iluminação e contraste nas imagens podem interferir com o cálculo dos gradientes e, consequentemente, a criação do descritor HOG. Por conta disso, é importante realizar um processo de normalização que envolve a agregação de células em blocos maiores, que podem assumir formas retangulares (R-HOG) e circulares (C-HOG). Os blocos podem se sobrepor, o que significa que determinadas células podem contribuir mais de uma vez para a criação do descritor final (DALAL; TRIGGS, 2005).

Os blocos do tipo R-HOG possuem três parâmetros: o número de pixels por célula, o número de células por bloco e o número de canais por histograma de célula (LOPES, 2015). Por sua vez, blocos do tipo C-HOG apresentam quatro parâmetros: o número de partições angulares e radiais, o raio da partição central e o fator de expansão para o raio das partições adicionais (DALAL; TRIGGS, 2005). Em seu experimento, Dalal e Triggs descobriram que os melhores parâmetros para descritores R-HOG para a detecção de pessoas são divisões de quatro blocos de 16x16 compostos por quatro

células de 8x8 pixels, com nove canais de histogramas. Já para descritores C-HOG, os melhores parâmetros são duas partições radiais com quatro partições angulares, um raio central de 4 pixels e um fator de expansão de 2. Segundo os autores, ambos os descritores obtiveram desempenhos similares, com C-HOGs apresentando uma pequena vantagem (DALAL; TRIGGS, 2005).

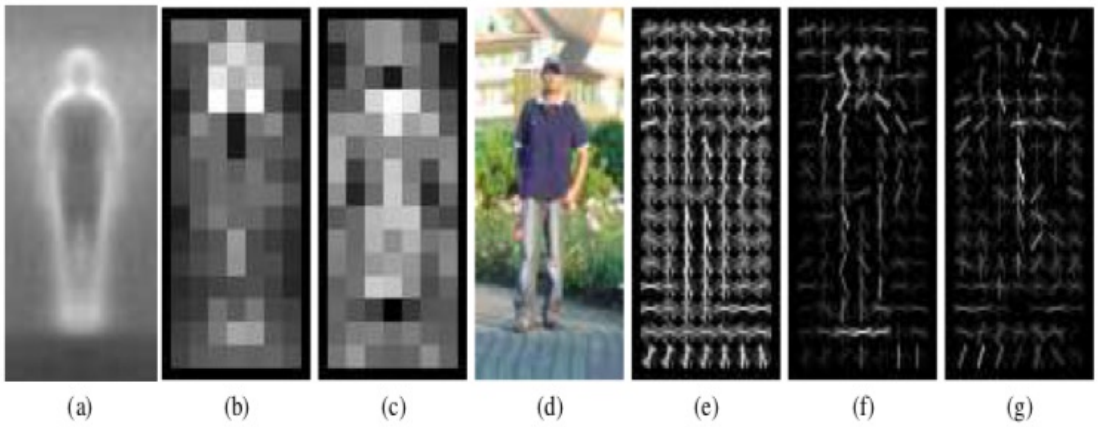
Dalal e Triggs apresentaram quatro métodos diferentes para a normalização dos blocos ao todo, e todos apresentaram melhorias de desempenho se comparados aos blocos não-normalizados. No entanto, o método L1-sqrt apresenta um menor custo computacional (LOPES, 2015). Considerando H como um vetor não-normalizado contendo todos os histogramas de um bloco, $\|H\|$ o somatório desses vetores, e ϵ uma constante muito pequena, a normalização resultante é:

$$H_{normalizado} = \sqrt{\frac{H}{\|H\| + \epsilon}} \tag{4}$$

2.2.2.3 Implementação das Técnicas

O processo de funcionamento da técnica de HOG é exemplificado na Figura 7 (DALAL; TRIGGS, 2005). Para sua implementação, as imagens de testes (d) são divididas em células menores (b, c), e em cada célula são determinadas a intensidade e a direção dos gradientes de orientação (f, g). Essa direção é determinada ao analisar as células adjacentes à célula em questão. O gradiente funciona como uma espécie de vetor que aponta na direção da mudança de intensidade da célula, criando uma forma simples de representação do objeto em questão.

Figura 7 – Geração de histogramas de gradientes orientados

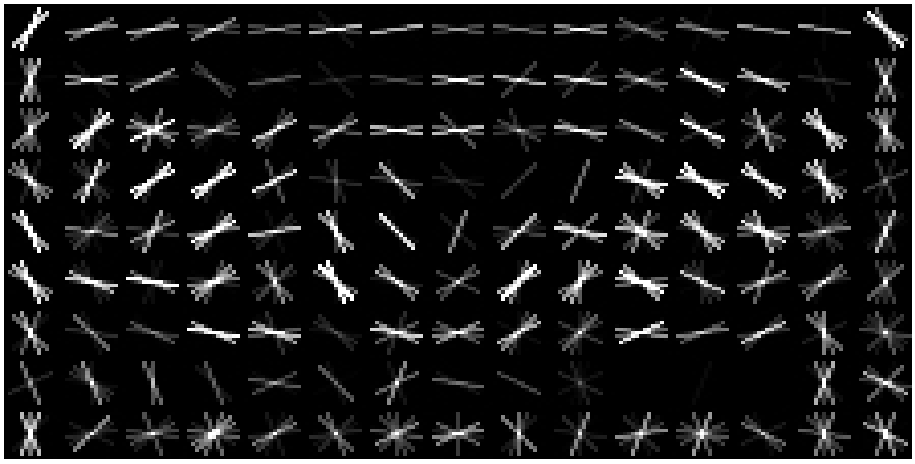


Fonte: Dalal e Triggs (2005, p. 8).

A partir disso, são extraídas da imagem as regiões de interesse para o identificador de acordo com as necessidades do usuário. Por exemplo, os HOGs conseguem indicar contornos da região dos olhos e diferenciá-los de outras regiões do

rosto (SAN; AYE, 2013). A Figura 8 representa um descritor HOG (intensidade e a direção dos gradientes de orientação) calculado a partir de um conjunto de imagens para realizar a detecção de um olho aberto.

Figura 8 – Exemplo de um descritor HOG para detecção de olhos abertos.



Fonte: Autor (2019).

Com base em uma área de interesse de uma imagem com um descritor HOG calculado, é possível analisar várias imagens de um banco de dados à procura de áreas de interesse compatíveis com o descritor.

2.2.3 Detecção de Círculos utilizando a Transformada de Hough

A Transformada de Hough é uma técnica de extração de instâncias imperfeitas de objetos em imagens criada por Paul Hough durante a década de 1960 (HOUGH, 1962). Anos depois, Dana H. Ballard ajudou a popularizar o método no ramo da visão computacional ao discutir usos genéricos de identificação de formatos, como parábolas e linhas (BALLARD, 1981).

De forma geral, a Transformada de Hough faz um mapeamento de todos os pixels em uma imagem, representando-os na forma de retas ou senóides em um espaço de parâmetros (MARRONI, 2002). O espaço de parâmetros é definido pela representação paramétrica utilizada para descrever linhas no plano da imagem (BALLARD, 1981). Dessa forma, é possível identificar formatos por meio do acúmulo de pontos coincidentes da reta.

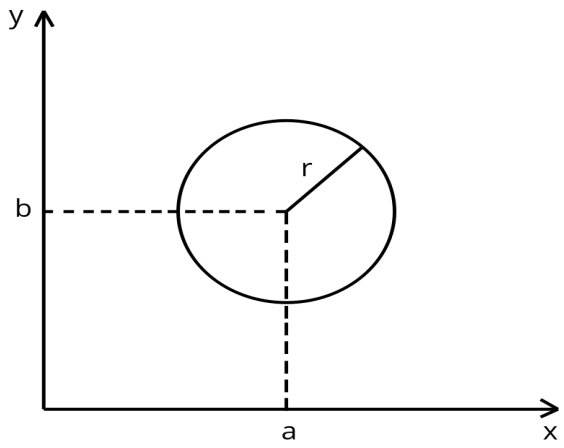
Uma das especializações da Transformada de Hough envolve a detecção de círculos em imagens de acordo com parâmetros definidos. Em um espaço bidimensional, um círculo pode ser representado pela seguinte equação:

$$(x - a)^2 + (y - b)^2 = r^2 \tag{5}$$

Nela, (a, b) são as coordenadas do centro do cículo, r representa seu raio e

(x, y) representa um ponto da sua superfície, como é observado na Figura 9.

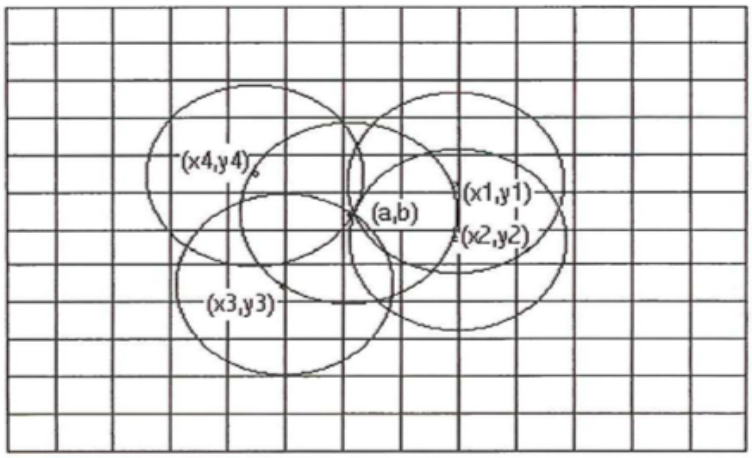
Figura 9 – Representação de um círculo.



Fonte: Autor (2019).

Utilizando isso como base, é possível detectar um círculo de raio r em uma imagem por meio da análise de seus pontos de borda. Interpretando esses pontos de borda como possíveis centros de outros círculos, é possível criar um campo acumulador no formato de grade que armazena o número total de curvas que atravessam cada célula, como exemplificado na Figura 10 (DUDA; HART, 1972).

Figura 10 – Identificação de círculos adicionais nas bordas do círculo original.



Fonte: Marroni (2002, p. 40).

Após todos os possíveis candidatos a círculos terem sido computados, um processo de votação é feito para avaliar quais células possuem o maior número de curvas acumuladas. Por fim, o círculo mais votado do acumulador é o círculo identificado da imagem.

Este capítulo apresentou três técnicas de Visão Computacional que podem ser utilizadas para realizar a detecção de olhos e, no caso dos algoritmos de Viola e Jones e HOG, também é possível realizar a detecção de face. O próximo capítulo entra em

detalhes sobre a metodologia adotada para a aplicação de cada um dos métodos de detecção bem como a forma de estabelecimento dos parâmetros para cada método.

3 MÉTODO

Este capítulo entrará em detalhes sobre a metodologia escolhida para realizar a aplicação do sistema proposto, incluindo informações sobre as ferramentas empregadas, técnicas escolhidas e decisões que precisaram ser tomadas ao longo da realização do trabalho.

3.1 Escopo do Trabalho

Esse trabalho considerou um escopo bem definido para a detecção de olhos, que envolve um motorista de um carro. Nesse sentido, as otimizações consideraram ajustar os parâmetros considerando um motorista a uma distância entre 30 e 80 centímetros da câmera. Para simular esse cenário, foram obtidas trinta imagens de um mesmo usuário utilizando diferentes ângulos para o rosto, com todas as imagens listadas no Apêndice B. Além dos parâmetros em relação à distância da câmera e ângulo de rosto, é importante destacar que em metade das fotos adquiridas o usuário apresenta os olhos fechados para que se possa realizar uma análise mais específica da técnica dos Histogramas de Gradiente Orientados. A Figura 11 exemplifica duas imagens utilizadas para a análise do sistema.

Figura 11 – Exemplos de imagens utilizadas para implementação das técnicas

Fonte: Autor (2019).

Embora tenham sido utilizadas apenas imagens de um mesmo usuário para o desenvolvimento dos códigos, com a metodologia proposta por este trabalho, as etapas executadas para descobrir os valores otimizados de parâmetros também podem ser

generalizadas para um número maior de usuários.

3.2 Ferramentas Utilizadas

Nesta seção, estão listadas as ferramentas de hardware e software utilizadas para o desenvolvimento dos algoritmos deste trabalho, sendo as bibliotecas OpenCV e Dlib utilizadas para a utilização de funções relacionadas às técnicas mencionadas no capítulo anterior, e as bibliotecas Psutil e Time utilizadas para realizar a análise do desempenho de cada código.

3.2.1 Sistema Operacional e Características de Software e Hardware

Para o desenvolvimento dos códigos e a subsequente análise dos resultados, os códigos foram criados utilizando um Macbook Air (2013) com as seguintes características de hardware:

a. Processador: Intel i7 dual-core 1.7GHz;
b. Memória RAM: 8GB;
c. GPU: Intel HD Graphics 5000;
d. Disco Rígido: 256GB SSD;
e. Câmera: FaceTime HD 720p.

A linguagem de programação escolhida para codificação é a Python (versão 3.7) e foram utilizadas como bibliotecas de suporte para as aplicações das técnicas a OpenCV (versão 4.1) e Dlib (versão 19.18). Para a extração do custo computacional, foi utilizada a biblioteca Psutil (versão 5.3).

3.2.2 OpenCV

OpenCV é uma biblioteca de programação open source focada no desenvolvimento de ferramentas voltadas para a área de Visão Computacional em tempo real e aprendizado de máquina. Introduzida pela Intel em 1999, esta biblioteca é multi plataforma, sendo implementada para C++ e possuindo interfaces para Python, Java e outras linguagens de programação (OPENCV, 1999d).

Dentre as ferramentas proporcionadas por esta ferramenta estão funções para processamento digital de imagens e vídeos, álgebra linear, identificação de objetos. A biblioteca possui uma vasta gama de técnicas implementadas, incluindo aplicações da Transformada de Hough para círculos e do algoritmo de Viola e Jones, incluindo filtros em cascata para a detecção de várias partes do rosto de uma pessoa.

Pelo fato de possuir uma licença do tipo *Berkeley Software Distribution* (BSD) que permite amplas modificações em seu código-fonte, a biblioteca OpenCV é amplamente utilizada por empresas como Google, Yahoo, Microsoft, IBM, Sony, Honda,

Toyota, dentre outras, e já registrou mais de 18 milhões de downloads mundialmente, segundo seus criadores (OPENCV, 1999d).

3.2.3 Dlib

Para a aplicação de um algoritmo para Histogramas de Gradientes Orientados, optou-se pela utilização da biblioteca open source Dlib. Lançada oficialmente por Davis King em 2002, a Dlib é uma biblioteca especializada na aplicação de algoritmos para aprendizado de máquina, mas também possui funções e módulos para processamento de imagens, data mining, threads, interfaces para usuários, entre outros (DLIB, 2002).

Segundo King, a filosofia de desenvolvimento da Dlib é focada na portabilidade e facilidade de uso. Embora a biblioteca tenha sido escrita na linguagem de programação C++, ela possui uma interface para Python, possibilitando seu uso para os sistemas propostos por este trabalho (DLIB, 2002).

3.2.4 Psutil

Psutil é uma biblioteca especializada no monitoramento do sistema operacional e seus processos (RODOLA, 2009). Criada em 2009 por Giampaolo Rodola, essa biblioteca conta com funções para determinar valores instantâneos de uso de CPU e memória por um processo ou pelo próprio sistema operacional e informações gerais sobre o sistema.

No escopo deste trabalho, algumas funções desta biblioteca foram utilizadas para monitorar o custo computacional dos códigos desenvolvidos. Para que essa análise pudesse ser feita de forma a encontrar os pontos críticos de processamento, instâncias dessas funções foram colocadas em posições específicas de cada código para que pudesse ser adquirida uma média e também observar os valores máximos de custo computacional.

3.3 Metodologia de Otimização dos Parâmetros

Para a utilização das técnicas apresentadas neste trabalho, é necessário definir uma série de parâmetros que influenciam diretamente no desempenho computacional e qualidade de detecção de objetos. Por conta disso, foi elaborada uma metodologia para identificar os melhores parâmetros para cada técnica considerando o escopo do trabalho. Em determinadas implementações, também foram incluídas etapas de pré-processamento para a redução de ruídos na imagem.

Três técnicas foram utilizadas neste trabalho. O algoritmo de Viola e Jones foi escolhido por já possuir diversas aplicações para detecção de faces e olhos (FERASSINI, 2014), com classificadores já treinados utilizando bibliotecas como OpenCV (1999d). A técnica de Histogramas de Gradientes Orientados (HOGs) de

Dalal e Triggs, apesar de ter sido desenvolvida com o objetivo principal de detecção de pessoas, também possui usos documentados para a detecção de olhos (CHEN; LIU, 2011). Por fim, a detecção de círculos por meio de uma forma especializada da Transformada de Hough foi utilizada por conta do formato da íris do olho. O objetivo final deste trabalho é analisar possíveis diferenças de desempenhos entre as técnicas. Por conta disso, cada técnica foi implementada utilizando diferentes parâmetros e suposições.

De forma geral, todos os códigos seguem as mesmas etapas de execução. Primeiramente, um diretório com diversas imagens capturadas, conforme o escopo do trabalho, é acessado e a primeira imagem é lida. Dependendo da técnica a ser utilizada, essa imagem poderá passar por um pré-processamento antes de ser realizada a detecção de objetos. Por fim, os resultados obtidos são armazenados em outro diretório e o código irá abrir a próxima imagem do banco de dados, repetindo sucessivamente estas etapas até todas as imagens do diretório serem processadas. Esse processo é representado pelo fluxograma representado na Figura 12.

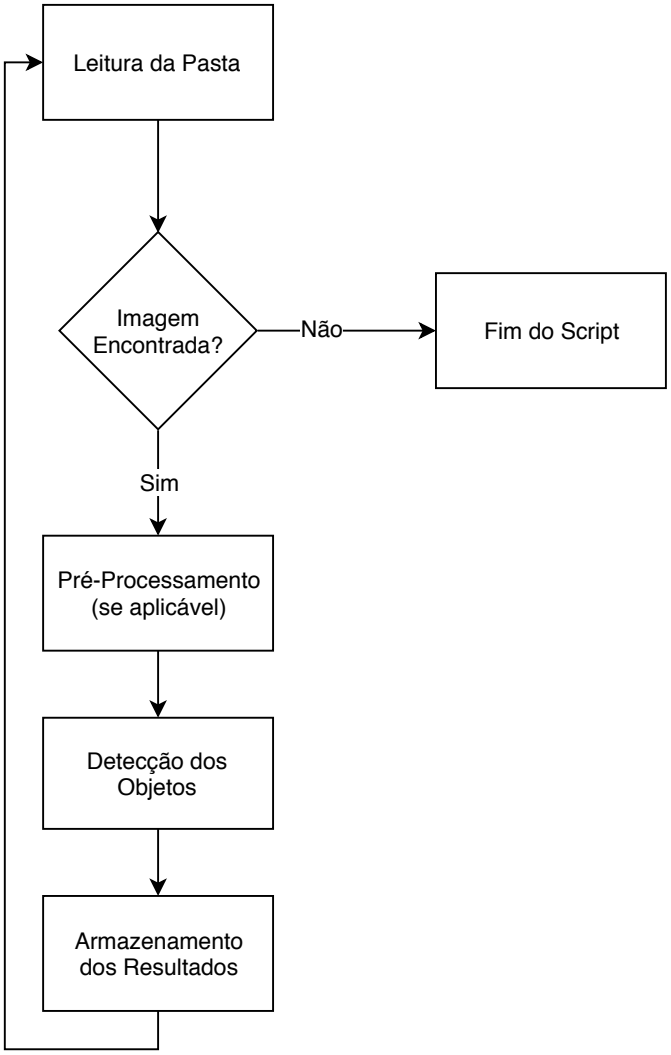
Esta seção entrará em detalhes sobre as etapas de cada código desenvolvido e os parâmetros escolhidos para as técnicas implementadas.

3.3.1 Aplicação e Escolha de Parâmetros para o Algoritmo de Viola e Jones

Para realizar a detecção do olho pelo método de Viola e Jones utilizando a biblioteca OpenCV optou-se por, primeiramente, fazer a detecção do rosto da pessoa com o intuito de reduzir a área de computação necessária para o olho. Utilizando a função DetectMultiScale (OPENCV, 1999a) juntamente com o classificador (*cascade*) de uma face (*haarcascade_frontalface_default.xml*) disponibilizado no código fonte da biblioteca (OPENCV, 1999c). Os arquivos de *cascade* são disponobo os parâmetros necessários para realizar a detecção do rosto são:

- **Image:** imagem a ser analisada;
- **ScaleFactor:** parâmetro de especificação do quanto o tamanho da imagem é reduzido para realização da detecção. Quando a imagem é analisada, ela é comparada com o modelo de treinamento que, por sua vez, possui um tamanho fixo. O ScaleFactor determina qual a razão de redução da imagem de teste que possibilite uma melhor comparação com o modelo de treinamento;
- **MinNeighbors:** por conta da forma como a função é aplicada, a detecção de objetos ocorre em forma de "deslizamento de janela", o que ocasiona em várias detecções para um mesmo objeto. O parâmetro MinNeighbors determina o número mínimo de detecções que um objeto precisa ter para que ele seja propriamente identificado, buscando evitar falsos positivos. Quando maior é este parâmetro, maior é a qualidade da detecção do objeto;

Figura 12 – Fluxograma das etapas de execução dos códigos.



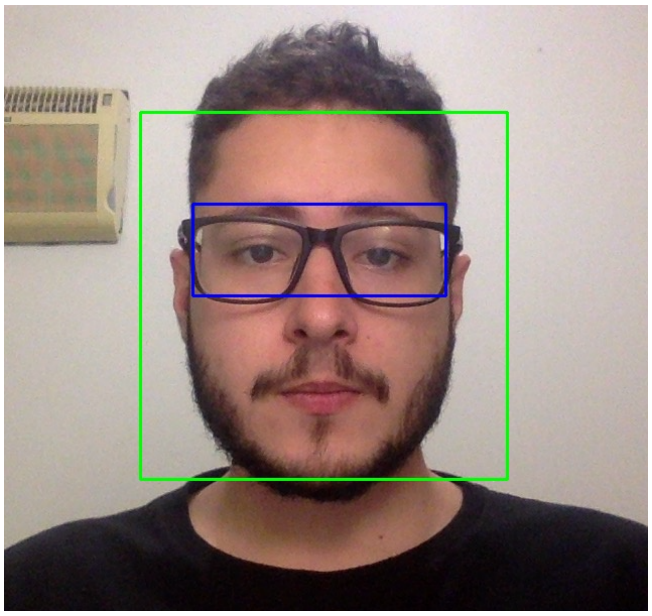
Fonte: Autor (2019).

- **MinSize:** considera um tamanho mínimo para a detecção do rosto buscando evitar falsos positivos;
- **MaxSize:** considera um tamanho máximo para a detecção do rosto.

Além de reduzir a área de interesse na imagem para otimizar a computação do script, também optou-se por utilizar o posicionamento proporcional médio dos olhos com relação ao rosto para delimitar a região de interesse. Este processo é ilustrado na Figura 13, na qual o retângulo em cor verde representa o tamanho total do rosto identificado pelo *cascade* da biblioteca OpenCV enquanto o retângulo azul representa a região total dos olhos.

Para a detecção de olhos é utilizada a mesma função da detecção de face, mas com um diferente *cascade* (*haarcascade_eye.xml*) também disponibilizado juntamente com o código-fonte do OpenCV (OPENCV, 1999c). Ela utiliza os mesmos parâmetros

Figura 13 – Delimitação do rosto e delimitação da região do olho

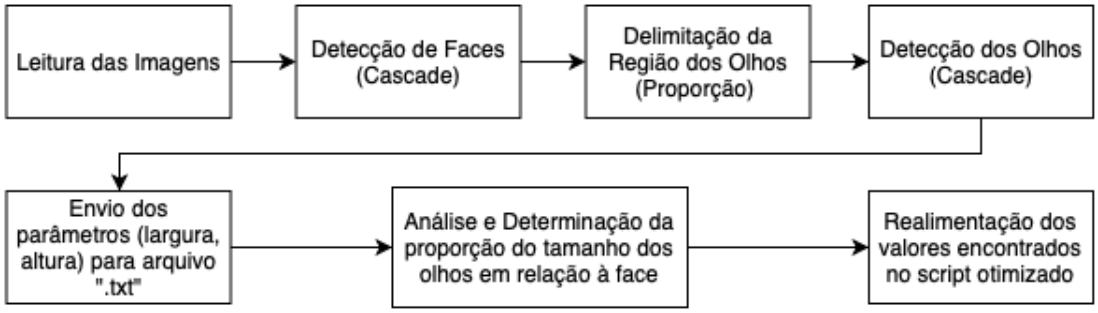


Fonte: Autor (2019).

mencionados anteriormente, porém com valores diferentes, otimizados para o banco de imagens utilizado. Este trabalho irá se referir a este primeiro script como VJ1.

Com o objetivo de obter parâmetros mais próximos do ideal, o script desenvolvido primeiramente enviava os valores dos tamanhos das regiões detectadas do rosto e dos olhos para um arquivo de texto. Em seguida, analisou-se os valores obtidos para que pudessem ser delimitados tamanhos mínimos e máximos para a detecção mais precisa dos objetos com base na proporção da largura da face detectada. A Figura 14 descreve as etapas para a determinação dos valores mínimos para a detecção dos olhos.

Figura 14 – Etapas para determinação de parâmetros para detecção de olhos.

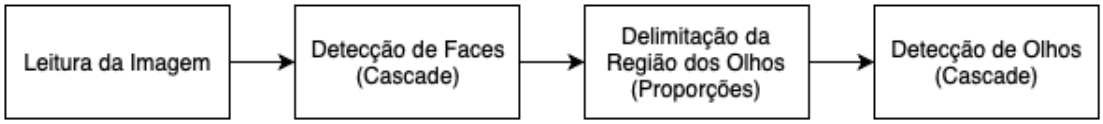


Fonte: Autor (2019).

Para fins de comparação, também optou-se por escrever um script (VJ2) que realizasse a detecção de olhos pelo algoritmo de Viola e Jones utilizando parâmetros padrão da função DetectMultiScale. Este teste visou realizar uma comparação de

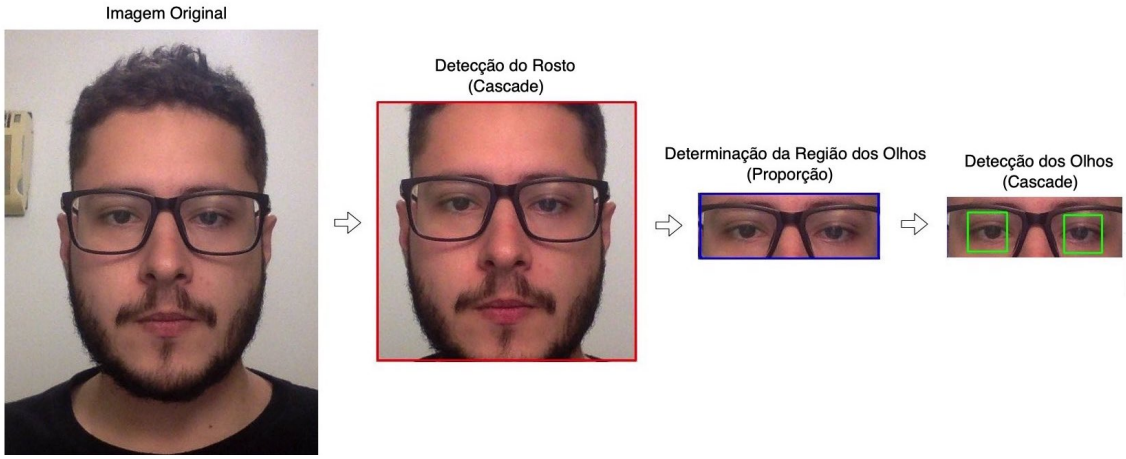
tempo de execução e nível de detecção em função dos parâmetros utilizados. De forma geral, a Figura 29 exibe um diagrama do funcionamento do script para a implementação da técnica de Viola e Jones enquanto a Figura 16 apresenta as etapas do processo de delimitação das regiões de interesse a partir de uma imagem capturada.

Figura 15 – Etapas de execução para implementação dos algoritmos VJ1 e VJ2.



Fonte: Autor (2019).

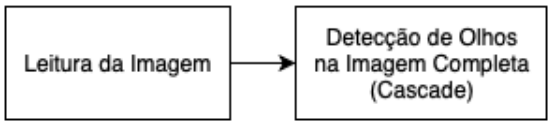
Figura 16 – Etapas de execução para implementação de Viola e Jones



Fonte: Autor (2019).

Por fim, foi criado um último código (VJ3) que realiza a detecção de olhos utilizando o classificador de Viola e Jones percorrendo a imagem inteira, como ilustrado na Figura 30.

Figura 17 – Etapas de execução de VJ3.



Fonte: Autor (2019).

Neste código, não são utilizadas as técnicas para realizar a delimitação da região de interesse. Esta abordagem foi adotada para verificar o impacto no uso computacional, tempo de execução e qualidade das detecções quando não é realizada a detecção inicial da região da face.

3.3.2 Aplicação Envolvendo o Uso de Histogramas de Gradientes Orientados

Para a aplicação dos HOGs foi utilizada a biblioteca Dlib e foram necessários dois passos principais. O primeiro envolve a criação de classificadores para identificação de olhos abertos e fechados para as imagens da base de dados. Essas identificações foram feitas utilizando uma ferramenta chamada *ImgLab* que é integrada com a Dlib. Por meio dela, o usuário identifica, utilizando retângulos, as áreas de interesse nas imagens de treinamento do algoritmo. A Figura 18 exibe uma das imagens utilizadas para o treinamento do classificador, com os retângulos em vermelho destacando a região de interesse a ser utilizada pelo programa.

Figura 18 – Delimitação da região de interesse para criação do classificador HOG

Fonte: Autor (2019).

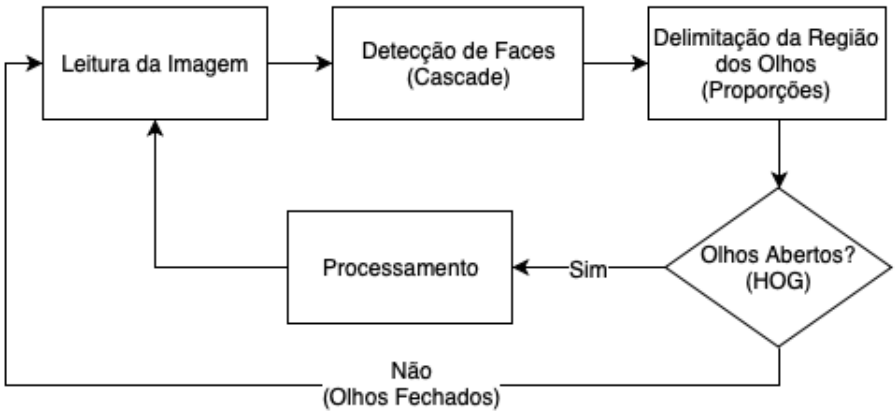
Com a identificação das regiões de interesse realizada, o *ImgLab* salva as dimensões e posições dos retângulos anotados em um arquivo do tipo *Extensible Markup Language* (XML). O segundo passo envolve utilizar esses arquivos em um script que utiliza um modelo de treinamento supervisionado do tipo *Support Vector Machines* (SVM) para criar os descritores HOG.

SVM é uma técnica de aprendizado de máquina especializada em realizar classificações entre dois grupos (CORTES; VAPNIK, 1995). Ao receber um grupo de imagens devidamente marcadas como pertencentes a uma classe, um algoritmo de treinamento consegue criar um modelo que classifica novos exemplos como pertencentes a uma das duas categorias. Para este trabalho, foram treinados três descritores HOGs: um detecta olhos abertos, outro detecta olhos fechados e um descritor genérico que detecta tanto olhos abertos como olhos fechados.

Um dos códigos desenvolvidos para a utilização desta técnica (denominada

HOG1) utiliza o *cascade* de detecção de face de Viola e Jones seguido da delimitação da região proporcional dos olhos buscando reduzir a área onde o descritor HOG irá atuar. A lógica implementada para a execução do HOG faz com que, primeiramente, apenas o descritor HOG para olhos abertos percorra a região de interesse. Se ele encontrar os olhos, a imagem é processada e o sistema parte para a próxima imagem da fila. Caso ele não encontre um olho aberto, considera-se que os olhos estão fechados. A Figura 31 descreve as etapas da execução do código desta implementação.

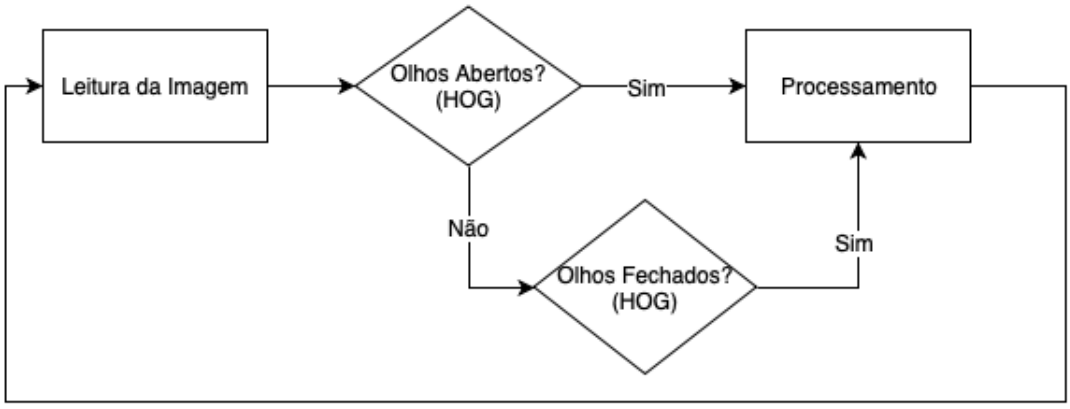
Figura 19 – Etapas de execução para o código de implementação de HOG1.



Fonte: Autor (2019).

Por fim, uma implementação alternativa (HOG2) foi feita com os descritores HOGs analisando a imagem inteira, um descritor de cada vez, para realizar a detecção dos olhos sem utilizar a delimitação da região de interesse, como é descrito na Figura 32.

Figura 20 – Etapas de execução para implementação do detector de olhos na imagem completa.



Fonte: Autor (2019).

Primeiramente, o descritor HOG para olhos abertos percorre a imagem procurando resultados positivos. Caso ele não encontre um olho aberto, o programa

então utiliza o descritor HOG para a detecção de olhos fechados. Por fim, os resultados do processamento são salvos em um outro diretório, com a diferença entre olhos abertos e fechados sendo indicada por meio de cores diferentes para a representação gráfica nas imagens.

3.3.3 Transformadas de Hough para Detecção de Círculos

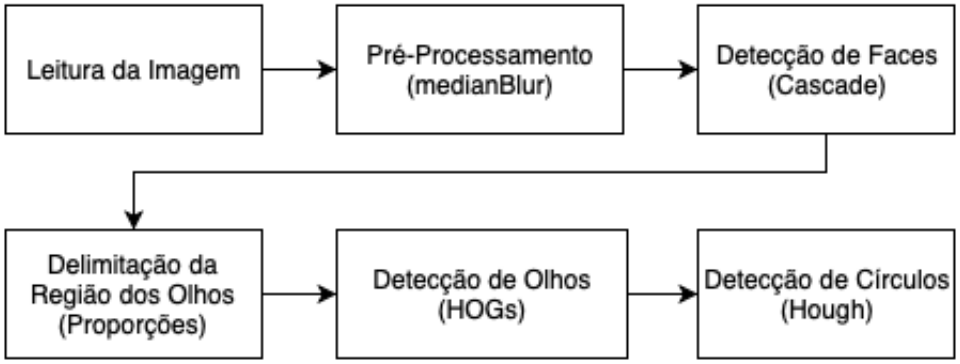
A aplicação da Transformada de Hough engloba a utilização da detecção de círculo na região do olho de uma pessoa para identificar a pupila, para então deduzir se a pessoa está com o olho aberto ou não. Essa abordagem foi feita tendo como base a identificação da região do olho utilizando outras técnicas, como HOG ou Viola e Jones, para então executar a detecção de círculos. No caso, se círculos fossem detectados na região de interesse, interpreta-se que os olhos estão abertos – caso contrário, assume-se que os olhos estão fechados. Essa detecção é feita por meio da função HoughCircles, da biblioteca OpenCV. Ela necessita dos seguintes parâmetros (OPENCV, 1999b):

- **Image:** imagem a ser analisada;
- **Method:** método de detecção a ser utilizado. Somente o método de Gradiente de Hough está implementado na biblioteca;
- **DP:** razão inversa da resolução do acumulador em relação à resolução da imagem;
- **MinDist:** distância mínima entre os centros dos círculos detectados. Se este parâmetro for muito pequeno, múltiplos círculos vizinhos podem ser falsamente detectados. Se for muito alto, alguns círculos podem não ser detectados;
- **Param1:** parâmetro específico do método implementado. Neste caso, é o limite máximo dentre os dois passados para o detector de bordas da função Canny() (OPENCV, 1999b);
- **Param2:** segundo parâmetro específico do método. Neste caso, é o limite do acumulador para os centros de círculos na fase de detecção. Quanto menor for, mais círculos falsos serão detectados;
- **MinRadius:** raio mínimo do círculo;
- **MaxRadius:** raio máximo do círculo.

Assim como na implementação da técnica de Viola e Jones, códigos utilizando tanto valores de parâmetros otimizados (CIRC1) como parâmetros de valor padrão (CIRC2) para a função HoughCircle foram desenvolvidos. No entanto, para otimizar a detecção de círculos, optou-se por realizar uma etapa de pré-processamento utilizando uma função intitulada medianBlur que atua como um filtro de suavização na imagem, diminuindo ruídos que poderiam ser erroneamente detectados como círculos. Em seguida, a região de interesse foi delimitada utilizando o *cascade* para

faces seguido da relação proporcional da região dos olhos. Por fim, um descritor HOG genérico que detecta tanto olhos abertos como olhos fechados delimita a região onde a função HoughCircles atua. A Figura 33 descreve as etapas da implementação desses algoritmos.

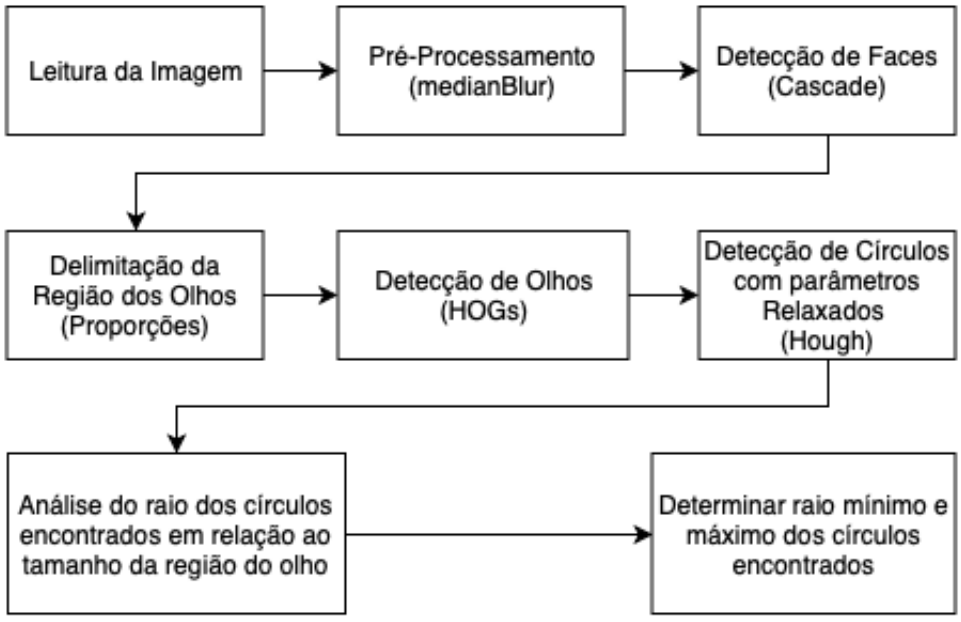
Figura 21 – Etapas de implementação da detecção de Círculos com HOG genérico.



Fonte: Autor (2019).

Para realizar a otimização dos parâmetros, observou-se os tamanhos máximo e mínimo de raio para a detecção do olho no script que utiliza os valores padrão para a função. Esses valores foram então aplicados em um novo código com o objetivo de otimizar o desempenho e o tempo de execução do código, processo demonstrado pela Figura 22. A seção de resultados entrará em detalhes sobre as diferenças entre as aplicações.

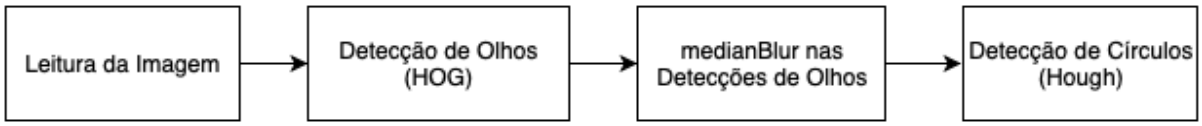
Figura 22 – Etapas para verificação dos parâmetros ideais para detecção de círculo.



Fonte: Autor (2019).

Além das implementações mencionadas, outro script (CIRC3) foi desenvolvido com o objetivo de analisar a diferença no desempenho e tempo de execução da técnica de detecção de círculos se o descritor HOG for utilizado na imagem completa, sem a delimitação da região da face. No entanto, para evitar erros de detecção do HOG, executou-se a etapa de "pré-processamento"apenas após a descreve região dos olhos, aplicando a função medianBlur diretamente nas áreas delimitdas pelos HOGs. A Figura 34 exemplifica o processo.

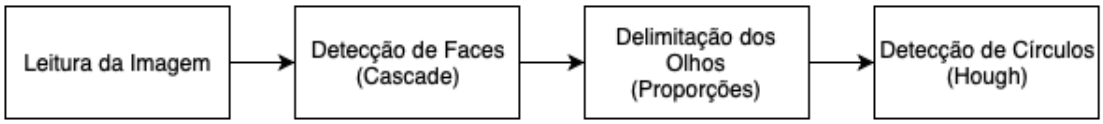
Figura 23 – Etapas de implementação da detecção de Círculos com HOG na imagem inteira.



Fonte: Autor (2019).

Por fim, uma última implementação (CIRC4) foi realizada com o objetivo de, ao invés de ser utilizado o descritor HOG genérico para a determinação da região dos olhos, a delimitação é feita por meio de uma relação de proporção em formato similar ao *cascade* de Viola e Jones para olhos. O algoritmo de implementação é descrito pela Figura 35.

Figura 24 – Etapas de implementação da detecção de Círculos com delimitação proporcional de olhos.



Fonte: Autor (2019).

Este capítulo abordou em detalhes as etapas de desenvolvimento de cada uma das técnicas utilizadas neste trabalho. Uma síntese dos algoritmos desenvolvidos com as técnicas de detecção de olhos está disponível no Apêndice A.

4 RESULTADOS

Este capítulo irá detalhar os resultados obtidos após a execução dos códigos e também fará uma análise comparativa do tempo de execução entre diferentes técnicas e parâmetros utilizados, considerando o tempo de execução para o script processar todas as imagens do banco de imagens. Ao todo, foram desenvolvidos nove algoritmos que utilizam os três métodos de detecção de objetos mencionadas neste trabalho. Optou-se por calcular o tempo de execução das técnicas através de imagens fixas e não através de captura de vídeo em tempo real por conta das diferentes condições de captura, que poderiam não ser igualitárias entre os testes. Sendo assim, o tempo de execução é o tempo que o código levou para executar as detecções nas trinta imagens do diretório.

A comparação de tempo foi feita através da inserção do argumento "time" na linha de comando do terminal, sendo que cada código foi executado dez vezes e a média e desvio padrão do tempo de execução foram calculados. As observações em relação ao uso de processador e memória foram calculados utilizando a biblioteca Psutil (RODOLA, 2009). É válido ressaltar que as próprias funções do Psutil possuem um custo computacional envolvido, mas o custo foi considerado menos significativo que o custo das outras funções do algoritmo.

A biblioteca Psutil proporciona funções de monitoramento da atividade de processos do sistema em um determinado momento. Para este trabalho, a função *(cpu_percent())* foi utilizada para obter o valor do processamento, enquanto a função *(memory_info())* retorna o valor da memória utilizada pelo código. Desse modo, foi necessário utilizar chamadas dessas funções em determinados pontos de cada código para assegurar que a análise está abrangendo não somente as instâncias críticas de utilização de memória e processamento, como também os pontos onde o algoritmo possui um custo computacional relativamente constante. Observou-se que os pontos mais críticos de processamento ocorrem durante a detecção de objetos. A partir desses valores, uma média dos resultados de cada instância do código foi obtida e, por fim, o algoritmo foi executado dez vezes para a obtenção de uma média final do valor do custo computacional de cada técnica. Em cada subseção, uma breve análise de maior utilização do processamento é feita para cada técnica utilizando valores obtidos de amostras. Como o uso de memória é estável ao longo da execução de todos os códigos, apenas o processamento é considerado nesta análise.

A qualidade das detecções de cada técnica também foi analisada, e uma matriz

de confusão representando a acurácia, proporção de predições corretas em relação a todas as predições feitas (METZ, 1978), também está inclusa neste capítulo.

Por fim, é válido ressaltar que o uso de processador apontado pelo terminal pode ultrapassar o valor de 100%. Isso acontece porque a função utilizada da biblioteca Psutil exibe o valor de processamento em função de um único núcleo. Como o computador utilizado para a execução destes scripts possui mais de um núcleo, o valor de processamento ultrapassa 100%.

4.1 Implementação do algoritmo de Viola e Jones com parâmetros otimizados e relaxados

Três scripts foram desenvolvidos para comparação de técnicas envolvendo o algoritmo de Viola e Jones. Primeiramente, decidiu-se analisar os efeitos da determinação de parâmetros otimizados para a implementação do algoritmo, realizando a detecção da face com a delimitação da região proporcional dos olhos e, subsequentemente, utilizando o *cascade* para a detecção de olhos na região de interesse. Um algoritmo "relaxado", isto é, um código cujas funções foram executadas utilizando seus valores padrão, foi executado utilizando valores padrão das funções implementadas pela biblioteca OpenCV, que consideram o parâmetro ScaleFactor padrão como 1.1, minNeighbors como 3 e não possuem um tamanho mínimo para a detecção de objeto. Em seguida, um código utilizando valores otimizados como parâmetros das funções foi executado e foram analisadas as mesmas trinta imagens do banco de dados. No caso de parâmetros como scaleFactor e minNeighbors, os valores ideais foram encontrados manualmente, utilizando o valor padrão da função como parâmetro inicial e incrementando ou decrementando o valor após a análise de resultados utilizando uma matriz de confusão. Por fim, executou-se um terceiro código que realizava a detecção de olhos percorrendo toda a imagem, sem fazer a detecção inicial do rosto.

Para o algoritmo com parâmetros otimizados, os seguintes valores foram utilizados para a função de detecção de faces:

- **ScaleFactor:** 1.15;
- **MinNeighbors:** 7;

Para a detecção de olhos, os valores otimizados dos parâmetros foram:

- **ScaleFactor:** 1.2;
- **MinNeighbors:** 9;
- **MinSize:** x/y.

O parâmetro MinSize foi realimentado na função após se analisar a proporção do tamanho de detecção do olho em relação ao tamanho da face. O código otimizado (VJ1) não somente foi executado em um tempo menor, como também utilizou menos

memória e processador durante sua execução se comparado ao código com parâmetros relaxados (VJ2). Já o algoritmo que fez a detecção de olhos percorrendo toda a imagem (VJ3) demonstrou um tempo de execução reduzido e menor uso de memória se comparado ao otimizado (VJ1), embora tenha apresentado um aumento considerável no uso de processador. Estes resultados estão detalhados na Tabela 1.

No caso dos algoritmos VJ1 e VJ2, as chamadas das funções de monitoramento do sistema foram feitas após a detecção de face e dos olhos, assim como durante o processamento final da imagem — momento onde está sendo realizada as alterações gráficas na imagem. No caso de VJ3, por conta da ausência da etapa de detecção de face, foram considerados apenas dois pontos de monitoramento, sendo um no momento da detecção dos olhos e o segundo no processamento final da imagem. Em uma amostra obtida para comparação, o uso máximo de processamento do código logo após a detecção de faces utilizando o *cascade* foi de 230,1% em VJ1 e 300% em VJ2. Essa diferença é provavelmente causada pelo fato de que, ao utilizar parâmetros padrão para a função, VJ2 realiza uma detecção errônea de mais de uma face em determinadas imagens. Essa diferença também é observada logo após a utilização da função de detecção de olhos: enquanto VJ1 utiliza 262%, VJ2 utiliza 301,3% do processamento. Por fim, durante o processamento final das imagens, o custo é similar. VJ1 utiliza 98,3% e VJ2 utiliza 99,7% de um núcleo da CPU.

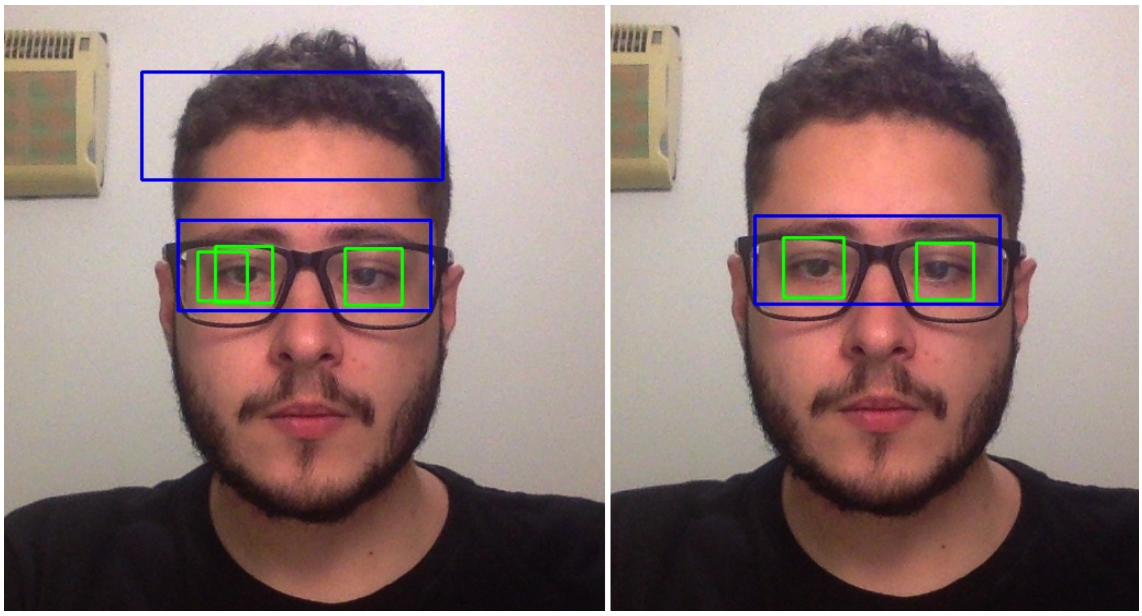
Técnica	Tempo Médio (em segundos)	Processador (%)	Memória (em MB)
Viola e Jones com Parâmetros Otimizados (VJ1)	3.874	202.38	87.51
Viola e Jones com Parâmetros Relaxados (VJ2)	4.967	243.61	105.05
Viola e Jones percorrendo Imagem Completa (VJ3)	2.687	216.39	62.53

Tabela 1 – Comparativo entre os códigos VJ1, VJ2 e VJ3

É importante destacar que o código relaxado apresentou um número considerável de falsos positivos tanto na detecção de faces como também na detecção de olhos se comparado ao otimizado. A Figura 25 apresenta a diferença na detecção de objetos entre a técnica com parâmetros relaxados (esquerda) e otimizados (direita) baseando-se na mesma imagem do banco de dados.

Enquanto o algoritmo VJ1 alcançou uma acurácia de 98.3% nas detecções de olhos, VJ2, por não utilizar os valores ideais de parâmetros, teve uma acurácia de apenas 81,5%. Os valores específicos dos acertos de cada tipo de detecção estão listados em uma matriz de confusão representada pela Tabela 7.

Figura 25 – Comparação entre uso de parâmetros relaxados e otimizados



Fonte: Autor (2019).

4.2 Implementação de HOGs para diferenciação entre olhos abertos e fechados

Com relação à implementação utilizando HOG, foi comparado o desempenho de códigos que utilizam dois detectores de HOGs diferentes, um para a detecção do olho aberto e outro para o olho fechado. Enquanto um dos scripts utilizou o *cascade* de face para delimitar uma região de detecção (HOG1), o outro percorreu a imagem inteira para realizar a detecção de objetos (HOG2). O código que percorreu a imagem inteira apresentou um tempo de execução maior e um processamento menor que o código no qual o descritor HOG é aplicado somente na região da face (região determinada pelo método de Viola e Jones). A Tabela 2 apresenta os resultados das técnicas, enquanto a Figura 26 representa a diferenciação entre olhos abertos e fechados.

Para o código HOG1, as funções de monitoramento foram utilizadas após a detecção de faces, após a utilização do descritor HOG para detecção de olhos abertos e durante o processamento final da imagem, enquanto que em HOG2 a etapa de detecção de faces é substituída pelo segundo descritor HOG para detecção de olhos fechados. No entanto, essa segunda medida somente é realizada caso não seja detectado um olho aberto na imagem.

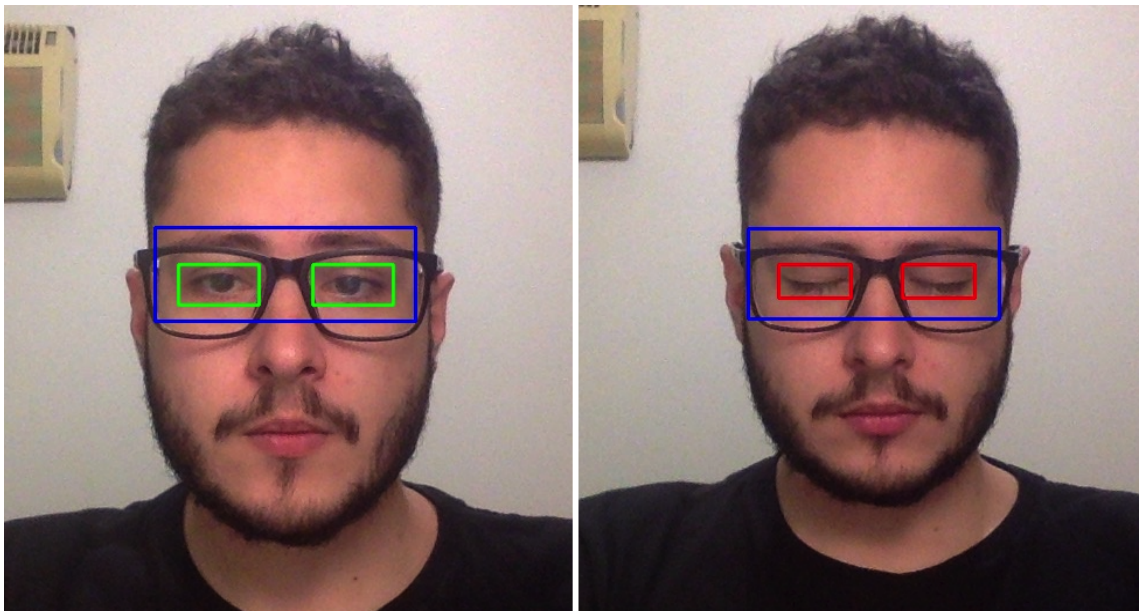
Em uma amostra retirada das execuções do código HOG1, o valor máximo de uso de processamento alcançado pela delimitação da face pelo *cascade* é de 280,2%, enquanto que a detecção feita pelo próprio HOG utilizou aproximadamente 99,9% para a computação. Por fim, o valor máximo relacionado ao processamento final da imagem dessa amostra é de 99,98%.

Para HOG2, não foram detectadas diferenças significativas para o uso de

Técnica	Tempo Médio (em segundos)	Processador (%)	Memória (em MB)
Detecção via HOGs na Região da Face (HOG1)	3.875	149.12	95.63
Detecção via HOGs na Imagem Completa (HOG2)	11.019	93.37	105.20

Tabela 2 – Comparativo entre HOG1 e HOG2

Figura 26 – Detecção de olhos abertos e fechados via HOG



Fonte: Autor (2019).

processamento do detector HOG. No entanto, pela ausência da etapa de delimitação da face pelo *cascade*, o valor médio da utilização de CPU é reduzido. Em uma amostra observada, o custo de processamento do detector para olhos abertos utilizou 100,3% do processador, enquanto o detector de olhos fechados utilizou 100%.

É válido ressaltar que ambas as técnicas apresentaram um nível ideal de acurácia, detectando corretamente todos os olhos abertos e fechados nas trinta imagens do diretório. Levando em consideração que o treinamento do modelo SVM para a aplicação do classificador HOG utilizou apenas vinte imagens, os resultados foram satisfatórios.

4.3 Implementação da detecção de círculos

A implementação da detecção de círculos através da biblioteca OpenCV pode apresentar detecções incorretas dependendo da quantidade e do formato dos objetos presentes na imagem. A Figura 27 exibe uma aplicação da função HoughCircles sem

nenhuma delimitação de região para a detecção.

Figura 27 – Detecção de círculos sem região de interesse definida

Fonte: Autor (2019).

Por conta desse problema, optou-se por utilizar a implementação da Transformada de Hough em conjunto com outras técnicas que delimitassem a região de interesse da imagem. Uma das técnicas envolve a utilização do *cascade* de Viola e Jones para detecção de face seguido de HOGs para a detecção da região dos olhos. Para isso, foi treinado um HOG genérico que detectasse o olho tanto aberto como fechado e então aplicasse a detecção de círculos na área encontrada, como foi explicado pela Figura 33 no capítulo de Metodologia. Assim como na implementação do algoritmo de Viola e Jones, optou-se por primeiramente comparar o desempenho do detector de círculos utilizando valores otimizados (CIRC1) e relaxados (CIRC2) para a função HoughCircles. No entanto, como a utilização dos valores padrão da função do OpenCV resultavam em poucos resultados positivos (param1 e param2 iguais a 100 e maxRadius sem um limite máximo), os valores de param2 e maxRadius precisaram ser aproximados experimentalmente dos valores otimizados. Esses valores otimizados são:

- **dp:** 0.8;
- **minDist:** img.shape[0]/6;
- **param1:** 160.
- **param2:** 12.
- **minRadius:** 7.
- **maxRadius:** valor calculado através da altura da região detectada dos olhos dividida por 2. Esse cálculo delimita que o valor máximo para a detecção de círculo estará condizente com o tamanho de um olho detectado.

Em todos os quesitos relacionados ao custo computacional e ao tempo médio de execução, ambos os códigos obtiveram resultados virtualmente similares, como é mostrado na Tabela 3.

Para as aplicações CIRC1 e CIRC2, quatro pontos foram escolhidos para a utilização das funções de monitoramento. Além de ser chamada após a detecção de face e detecção do descritor HOG para olho aberto, também foram obtidos valores de processamento após o uso da função de detecção de círculos e durante o processamento final da imagem. Em uma amostra obtida para comparação, a utilização máxima de processamento para a detecção de face utilizando o *cascade* é de 280,9% para CIRC1 e 280,1% para CIRC2. O maior custo computacional para a detecção das regiões dos olhos via HOG foi igual para as duas, 100% do uso de um núcleo. Para a detecção de círculos na amostra, o uso de processamento foi de 160,2% para CIRC1 e 167,5% para CIRC2. Por fim, o custo máximo envolvido com o processamento final da imagem foi de 99,7% para CIRC1 e 100% para CIRC2.

Técnica	Tempo Médio (em segundos)	Processador (%)	Memória (em MB)
Detecção de Círculos Otimizada com HOG Genérico (CIRC1)	3.876	137.83	98.47
Detecção de Círculos Relaxada com HOG Genérico (CIRC2)	3.874	138.43	98.73

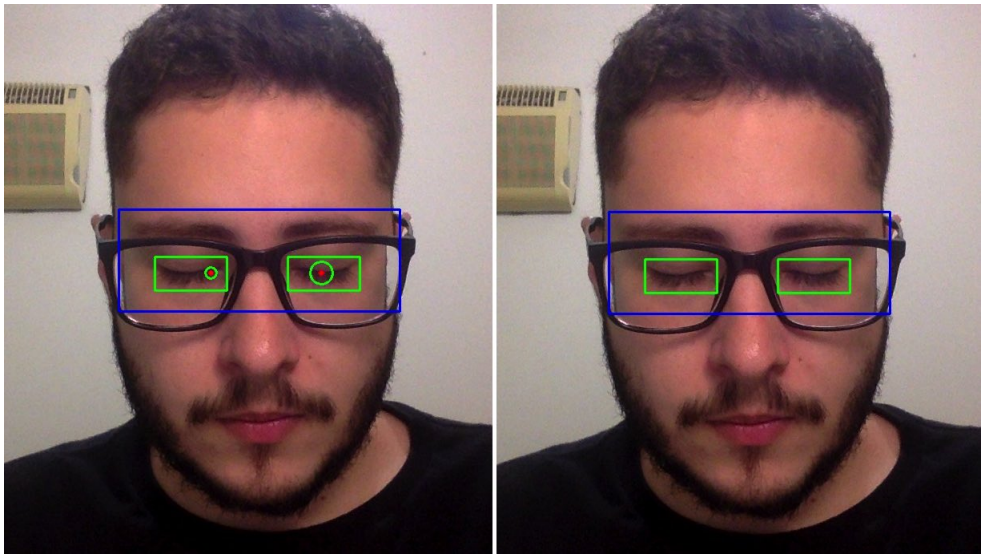
Tabela 3 – Comparativo entre os códigos CIRC1 e CIRC2

No entanto, existe uma diferença considerável no nível de detecção entre as duas técnicas. A Figura 36 exibe um exemplo de detecção de falsos positivos no algoritmo CIRC2 (esquerda), enquanto à direita é exibida a mesma imagem gerada pelo código que utiliza parâmetros otimizados (CIRC1). A acurácia de CIRC2 foi de 91,6%, enquanto que a acurácia de CIRC1 foi de 96,7%.

Para avaliar a influência da utilização do HOG como delimitador da região de interesse, também foi desenvolvido um script (CIRC4) que, em vez do HOG genérico, utilizasse relações de proporções na imagem após a detecção de faces pelo *cascade* de Viola e Jones para, finalmente, realizar a detecção de círculos nas imagens, como foi descrito na Figura 35 na seção de Metodologia. Os resultados obtidos em relação ao tempo de execução foram similares aos do algoritmo com parâmetros otimizados, enquanto o uso de memória e processamento através da técnica adaptada de Viola e Jones tenha apresentado uma leve vantagem em relação ao HOG.

O algoritmo CIRC4 possui seis pontos de monitoramento. Além da medida obtida após a detecção de face, também foi feita uma medição após a delimitação das duas regiões de olhos, duas medições após a chamada de cada função de detecção de círculos e, por fim, uma última medição após o processamento final da imagem. Em

Figura 28 – Falsos positivos detectados em CIRC2 (esq.) em relação ao CIRC1 (dir.)



Fonte: Autor (2019).

Técnica	Tempo Médio (em segundos)	Processador (%)	Memória (em MB)
Detecção de Círculos Otimizada com HOG Genérico (CIRC1)	3.876	137.83	98.47
Detecção de Círculos Otimizada com Região Proporcional dos Olhos (CIRC4)	3.722	141.21	92.88

Tabela 4 – Comparativo entre os algoritmos CIRC1 e CIRC4

uma amostra, o uso máximo de processamento para a detecção de faces pelo *cascade* foi de 286,8%, enquanto a delimitação manual das regiões dos olhos obteve um custo máximo de 96,9%. Em relação à detecção de círculos, o custo computacional máximo após a utilização da função foi de 143,5% para a detecção dos olhos no lado esquerdo do rosto e 156,4% para o lado direito. Para o processamento final, os custos foram de 107,2% para a representação gráfica do lado esquerdo do rosto e 113,5% para o lado direito.

No entanto, em questão de acurácia, CIRC4 possui resultados muito piores se comparado às todas as outras técnicas desenvolvidas. No caso, o algoritmo obteve um nível de detecção de olhos abertos mais baixo, como é mostrado na Tabela 7, e seu nível de acurácia foi de 85%.

Por fim, foi aplicada a detecção da região de interesse através de HOGs percorrendo a imagem inteira (CIRC3), removendo o uso do *cascade* para a face com o objetivo de avaliar a importância do classificador para o desempenho do método e, com isso, também é removido o ponto de monitoramento do custo computacional. Embora o tempo de execução do novo código tenha praticamente dobrado em relação ao otimizado (CIRC1), seu processamento e uso de memória reduziram,

como indicado na Tabela 5. Por meio de uma amostra, foi observado que o valor máximo de processamento para a utilização do HOG foi de 100,1%, enquanto o uso máximo para a detecção dos círculos nas regiões de interesse foi de 164,6%. Por fim, o custo máximo de processamento final da imagem na amostra escolhida foi de 148,3%.

Técnica	Tempo Médio (em segundos)	Processador (%)	Memória (em MB)
Detecção de Círculos Otimizada com HOG Genérico (CIRC1)	3.876	137.83	98.47
Detecção de Círculos Otimizada com HOG Genérico na Imagem Completa (CIRC3)	8.00	119.71	88.06

Tabela 5 – Comparativo entre os códigos CIRC1 e CIRC3

Também é válido destacar que o algoritmo CIRC3 possuiu a mesma acurácia que CIRC1, 96,7%, apesar de apresentar melhorias significativas em relação ao custo computacional. Assim como CIRC1, a técnica apenas apresentou dificuldades na identificação de olhos abertos, errando duas detecções.

Após a apresentação dos resultados de cada técnica, a próxima seção fará considerações finais em relação ao custo computacional e tempo de execução dos códigos, além de também apresentar uma análise da acurácia obtida de cada algoritmo.

4.4 Resultados Finais

Para discussão dos resultados encontrados e comparação desses resultados, a Tabela 6 reexibe os dados encontrados nos testes executados considerando o tempo de execução e o custo computacional utilizando a biblioteca Psutil.

De forma geral, a maioria das técnicas utilizadas apresentou desempenhos similares em relação ao tempo médio de execução, uso de processamento e uso de memória. Os resultados mais divergentes ocorreram quando se optou por percorrer a imagem completa para realizar a detecção de objetos. Essa decisão resultou em tempos de execução elevados, mas o uso de memória e processamento diminuíram por conta da exclusão do uso do *cascade* de detecção de face antes da utilização de outras técnicas de detecção de objetos. No entanto, o script que utilizava a técnica de Viola e Jones percorrendo com o *cascade* de olho na imagem inteira (VJ3) apresentou bons resultados em relação tanto ao uso de memória e processamento como também em relação ao tempo de execução.

Além da análise computacional dos códigos, também foi realizada uma análise da qualidade das detecções de cada técnica por meio de uma matriz de confusão, um medidor de performance para modelos de predições que classifica resultados (TING, 2017). O objetivo deste método é observar como um determinado modelo de classificação erra ao analisar os resultados obtidos. A avaliação de acurácia de uma

Técnica	Tempo de Execução Real (em segundos)	Processador (%)	Memória (em MB)
Viola e Jones com Parâmetros Otimizados (VJ1)	4.178	200	71
Viola e Jones com Parâmetros Relaxados (VJ2)	6.455	210	87
Viola e Jones percorrendo Imagem Completa (VJ3)	3.525	200	45
Detecção via HOGs na Região da Face (HOG1)	5.017	200	73
Detecção via HOGs na Imagem Completa (HOG2)	12.621	97	69
Detecção de Círculos Otimizada com HOG Genérico (CIRC1)	4.339	204	75
Detecção de Círculos Relaxada com HOG Genérico (CIRC2)	5.115	201	75
Detecção de Círculos Otimizada com HOG Genérico na Imagem Completa (CIRC3)	9.987	97	48
Detecção de Círculos Otimizada com Região Proporcional dos Olhos (CIRC4)	4.526	180	71

Tabela 6 – Comparativo entre todos os scripts desenvolvidos.

matriz de confusão utiliza como base a Equação 6 (METZ, 1978):

$$\text{Acurácia} = \frac{\text{VP + VN}}{\text{VP + VN + FP + FN}} = \frac{\text{predições corretas}}{\text{todas as predições}} \tag{6}$$

VP e VN se referem ao número de Verdadeiros Positivos e Verdadeiros Negativos, respectivamente, FP e FN se referem a Falsos Positivos e Falsos Negativos. Cada detecção correta de olho aberto ou fechado foi computada como VP ou VN, ou seja, em trinta imagens existe uma possibilidade de sessenta avaliações corretas de VP e VN. Os resultados desta Matriz de Confusão estão listados na Tabela 7.

É válido ressaltar que o número de detecções totais de um código pode ultrapassar o valor de sessenta, uma vez que podem ser detectadas mais de duas áreas como se fossem olhos, como é o caso dos resultados obtidos por VJ2. Esse tipo de identificação é representado na Figura 25.

Enquanto os resultados em relação à execução dos scripts seguiu uma determinada constante, a taxa de detecção em si divergiu muito dependendo da técnica. O algoritmo de Viola e Jones, quando otimizado com valores de parâmetros específicos para a base de dados utilizada por este trabalho, exibiu uma taxa de acertos próxima de 100% para as imagens do banco de dados. A detecção de círculos, todavia, apresentou uma taxa de acertos pior mesmo quanto utilizados parâmetros otimizados. Por conta de diversos fatores, foram detectados falsos positivos e, dependendo da escolha de parâmetros, falsos negativos no formato de cílios, óculos e outros objetos que podem interferir na detecção de olhos na imagem. Esta técnica não apresentou

Técnica	Verdadeiros Positivos	Verdadeiros Negativos	Falsos Positivos	Falsos Negativos	Acurácia
Viola e Jones com Parâmetros Otimizados (VJ1)	29	30	0	1	98.3%
Viola e Jones com Parâmetros Relaxados (VJ2)	30	23	12	0	81.5%
Viola e Jones percorrendo Imagem Completa (VJ3)	30	29	1	0	98.3%
Detecção via HOGs na Região da Face (HOG1)	30	30	0	0	100%
Detecção via HOGs na Imagem Completa (HOG2)	30	30	0	0	100%
Detecção de Círculos Otimizada com HOG Genérico (CIRC1)	28	30	0	2	96.7%
Detecção de Círculos Relaxada com HOG Genérico (CIRC2)	28	27	3	2	91.6%
Detecção de Círculos Otimizada com HOG Genérico na Imagem Completa (CIRC3)	28	30	0	2	96.7%
Detecção de Círculos Otimizada com Região Proporcional dos Olhos (CIRC4)	21	30	0	9	85%

Tabela 7 – Matriz de confusão para todos os scripts desenvolvidos.

melhorias significativas em relação às técnicas "concorrentes” nos quesitos de tempo de execução, processamento e uso de memória para a maioria dos scripts. No entanto, quando utilizado o método de detecção de círculos na região delimitada por um HOG genérico que percorre a imagem inteira, o sistema apresentou mudanças drásticas tanto no uso de processamento como no uso de memória, apesar de seu tempo de execução ter basicamente dobrado.

As aplicações que utilizaram exclusivamente HOG para diferenciar olhos abertos e fechados apresentaram uma taxa de acertos de 100%. O resultado é satisfatório, principalmente se levado em consideração que os detectores via HOG foram desenvolvidos utilizando apenas vinte imagens das trinta para treinamento – dez imagens de olhos abertos e dez imagens de olhos fechados. Como seu tempo de execução e uso de memória e processamento são similares aos encontrados nas técnicas que utilizam o algoritmo de Viola e Jones, os HOGs demonstraram ser a melhor solução para o escopo determinado.

Finalmente, a Tabela 8 exibe o tamanho de cada arquivo de classificação utilizado para este trabalho. Na Tabela, é possível observar que o arquivo para o classificador *cascade* de face possui um tamanho maior que o arquivo para o classificador *cascade* de olhos. Isso pode ser um fator determinante na diferença observada entre o custo computacional de uma chamada da função para detecção

de faces em relação a função de detecção de olhos. Com relação aos arquivos dos descritores HOG, nota-se que não existe uma diferença significativa com relação aos seus tamanhos.

Arquivo	Tamanho (em KB)
Cascade de Viola e Jones para Detecção da Face	930
Cascade de Viola e Jones para Detecção do Olho	341
Detector HOG Genérico	45
Detector HOG para Olho Aberto	42
Detector HOG para Olho Fechado	45

Tabela 8 – Tamanhos dos arquivos de classificação utilizados.

No próximo capítulo, são apresentadas as conclusões do trabalho e alguns ponderamentos sobre possíveis trabalhos futuros em relação à metodologia adotada.

5 CONCLUSÕES

Esse trabalho avaliou o desempenho de diversas técnicas de detecção de olhos desenvolvidas através de ferramentas de Visão Computacional. Com o objetivo de simular o posicionamento de um motorista dentro de um automóvel, trinta imagens foram adquiridas para realizar a execução dos testes necessários. Desse modo, os códigos desenvolvidos foram executados em cada uma das imagens do banco de imagens e seu tempo de execução e uso de processamento e de memória foram avaliados.

Uma das contribuições deste trabalho é a metodologia para o levantamento dos valores otimizados para os parâmetros das funções utilizadas. Sendo a aplicação bem definida com relação aos aspectos de detecção, utilizar os parâmetros mais apropriados possível é recomendável. Sem esses valores otimizados, a detecção da região da face e do olho apresentou falhas em determinadas ocasiões, sejam elas na forma de falsos positivos ou de falsos negativos. Esses parâmetros ainda influenciaram no desempenho de determinados scripts, podendo economizar tempo de execução ou uso de memória por conta das delimitações das regiões de interesse. Embora os valores otimizados encontrados se apliquem apenas ao banco de imagens utilizado por este trabalho, a metodologia adotada pode eventualmente ser generalizada para encontrar valores otimizados para qualquer usuário.

Outra contribuição é o estudo da utilização de técnicas complementares com o objetivo de encontrar uma otimização no desempenho e performance dos códigos. As aplicações utilizando Transformada de Hough, por exemplo, utilizaram passos das técnicas de Viola e Jones e HOGs para a delimitação da região de interesse. Na maioria dos casos, essa delimitação resultou na melhora do tempo de execução dos códigos, com exceção do script VJ3. No entanto, essas etapas adicionais impactaram no custo computacional dos códigos, como é o caso dos códigos HOG1 e CIRC1.

Embora o comparativo entre as técnicas abordadas tenha resultado em análises importantes para o tema, não foi discutida a viabilidade de uma solução embarcada para este sistema. É importante ressaltar que alguns dos equipamentos utilizados para o desenvolvimento deste trabalho (modelo de webcam e computador) são impraticáveis para utilização em veículos. Em busca de soluções mais portáteis, estudos futuros poderiam avaliar a possibilidade de utilização de computadores de pequeno porte como o Raspberry Pi em junção de uma webcam compacta para que a detecção de olhos possa ser feita em tempo real sem problemas com o processamento das imagens.

REFERÊNCIAS

ACHARJYA, P. P.; DAS, R.; GHOSHAL, D. A study on image edge detection using the gradients. **International Journal of Scientific and Research Publications**, v. 2, 2012.

APPLE. **About Face ID advanced technology**. 2019. Disponível em: https://support.apple.com/en-us/HT208108. Acesso em: 1 dez. 2019.

AUDI. **Driver assistance systems**. 2013. Disponível em: https://www.audi-mediacenter.com/en/foray-into-the-worlds-largest-market-segment-the-audi-a3-sedan-and-s3-sedan-3249/driver-assistance-systems-3350. Acesso em: 3 dez. 2019.

BALLANTYNE, M.; BOYER, R. S.; HINES, L. Woody bledsoe: His Life and Legacy. **AI Magazine**, v. 17, p. 7–20, 1996.

BALLARD, D. H. Generalizing the hough transform to detect arbitrary shapes. **Pattern Recognition**, v. 13, p. 111 – 122, 1981.

BENSON, D. **The impact of facial recognition technology on society**. 2017. Disponível em: http://www.cs.tufts.edu/comp/116/archive/fall2017/dbenson.pdf.

CHEN, S.; LIU, C. Precise eye detection using discriminating hog features. **International Conference on Computer Analysis of Images and Patterns**, v. 6854, p. 443 – 450, 2011.

CORTES, C.; VAPNIK, V. Support-vector networks. **Machine Learning**, v. 20, p. 273 – 297, 1995.

CROW, F. C. Summed-area tables for texture mapping. **Computer Graphics**, v. 18, p. 207 – 212, 1984.

DALAL, N.; TRIGGS, B. Histograms of oriented gradients for human detection. **2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition**, v. 1, p. 886–893, 2005.

DLIB. **dlib C++ Library**. 2002. Disponível em: http://dlib.net.

DUDA, R. O.; HART, P. E. Use of the hough transformation to detect lines and curves in pictures. **Graphics and Image Processing**, v. 1, p. 11 – 15, 1972.

FERASSINI, R. B. **Sistema de Detecção de Sonolência, por Meio de Visão Computacional**. Trabalho de Conclusão de Curso (Graduação em Engenharia Elétrica) — Universidade de São Paulo, 2014.

FREUND, Y.; SCHAPIRE, R. E. A decision-theoretic generalization of on-line learning and an application to boosting. **Journal of Computer and System Sciences**, v. 14, p. 119 – 139, 1996.

Paul V. C. Hough. **Method and Means for Recognizing Complex Patterns**. 1962. US3069654A, 18 dez. 1962.

JAN, T. von et al. Don't sleep and drive: – VW's fatigue detection technology. **Proceedings - 19th International Technical Conference on the Enhanced Safety of Vehicles (ESV), Washington, D.C., June 6-9, 2005**, 2005.

LOPES, M. T. **Histograma de Gradientes Orientados Utilizando Processamento Paralelo em GPU**. Trabalho de Conclusão de Curso (Graduação em Engenharia da Computação) — Universidade Tecnológica Federal do Paraná, 2015.

LUCION, V. **VISÃO COMPUTACIONAL E SINAIS BIOMÉDICOS PARA DETERMINAR SONOLÊNCIA EM MOTORISTAS**. Trabalho de Conclusão de Curso (Graduação em Engenharia da Computação) — Universidade Tecnológica Federal do Paraná, 2017.

MACIONKI, A.; STUMPF, W. K. **Sistema de Monitoramento Veicular e Sonolência em Motoristas**. Trabalho de Conclusão de Curso (Graduação em Engenharia da Computação) — Universidade Positivo, 2009.

MARRONI, L. S. **Aplicação da Transformada de Hough para Localização dos Olhos em Faces Humanas**. Dissertação (Mestrado em Engenharia Elétrica) — Universidade de São Paulo, 2002.

Robert K. McConnell. **Method of and Apparatus for Pattern Recognition**. 1986. US4567610A, 28 jan. 1986.

MERCEDES-BENZ. **Safety:** Mercedes-Benz USA. 2018. Disponível em: https://www.mbusa.com/mercedes/benz/safety#module-3. Acesso em: 13 jun. 2019.

METZ, C. E. Basic principles of roc analysis. **Seminars In Nuclear Medicine**, v. 8, p. 283–298, 1978.

NHTSA. **Drowsy driving:** asleep at the wheel. 2018. Disponível em: https://www.cdc.gov/features/dsdrowsydriving/index.html. Acesso em: 13 jun. 2019.

OPENCV. **Cascade Classification - OpenCV**. 1999. Disponível em: https://docs.opencv.org/2.4/modules/objdetect/doc/cascade_classification.html.

OPENCV. **Feature Detection - OpenCV**. 1999. Disponível em: https://docs.opencv.org/2.4/modules/imgproc/doc/feature_detection.html?highlight=houghcircles.

OPENCV. **Haar Cascades - OpenCV**. 1999. Disponível em: https://github.com/opencv/opencv/tree/master/data/haarcascades.

OPENCV. **OpenCV**. 1999. Disponível em: https://opencv.org.

RODOLA, G. **Psutil**. 2009. Disponível em: https://psutil.readthedocs.io/.

SAN, N. N.; AYE, N. Eye detection system using orientation histogram. **International Journal of Advanced Research in Computer Engineering Technology**, v. 2, p. 1352 – 1356, 2013.

TAIGMAN, Y. et al. Deepface: closing the gap to human-level performance in face verification. **2014 IEEE Conference on Computer Vision and Pattern Recognition**, v. 1, p. 1701–1708, 2014.

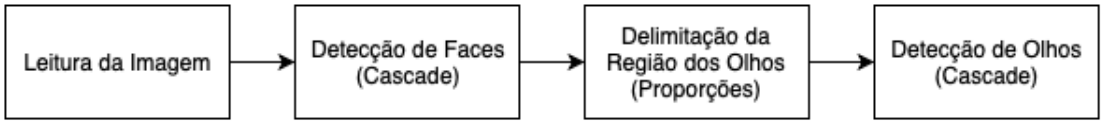
TING, K. M. Confusion matrix. **Encyclopedia of Machine Learning and Data Mining**, 2017.

VIOLA, P.; JONES, M. Robust real-time object detection. **International Journal of Computer Vision**, 2001.

APÊNDICE A

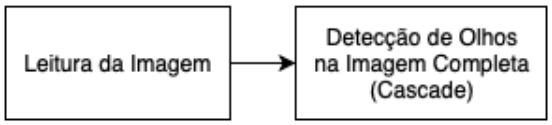
Este apêndice apresenta todos os diagramas utilizados para a representação do funcionamento dos códigos desenvolvidos para este trabalho e tem por objetivo facilitar uma eventual consulta dos funcionamentos.

Figura 29 – Etapas de execução de VJ1 e VJ2.



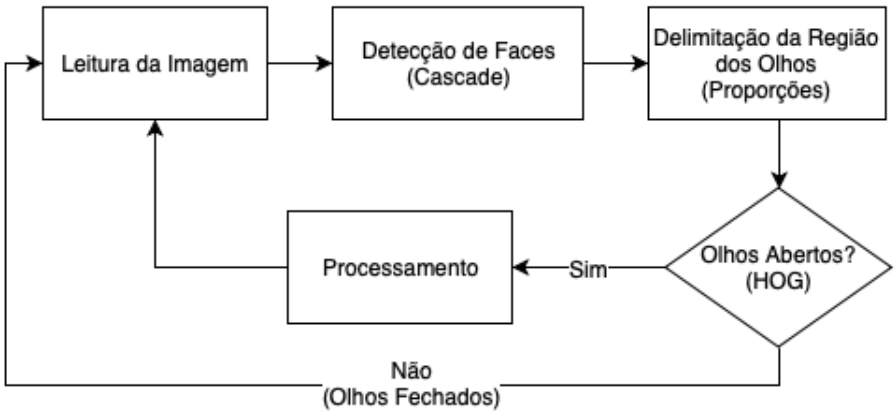
Fonte: Autor (2019).

Figura 30 – Etapas de execução de VJ3.



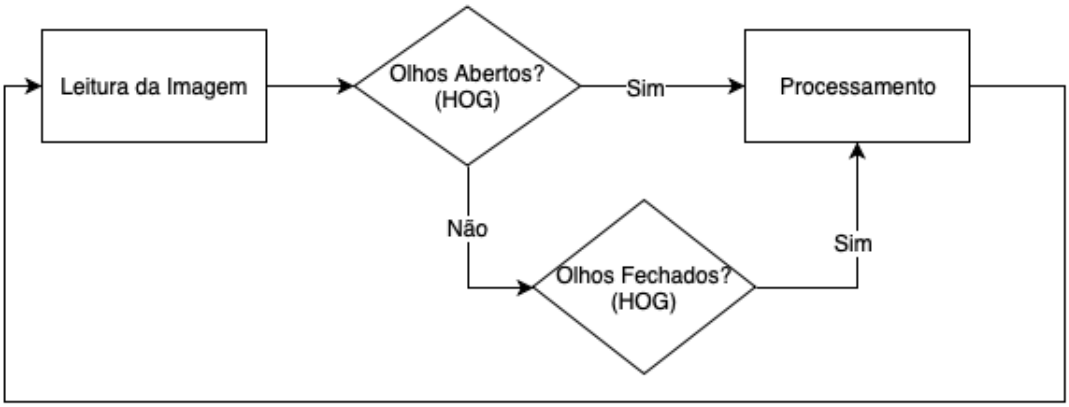
Fonte: Autor (2019).

Figura 31 – Etapas de execução de HOG1.



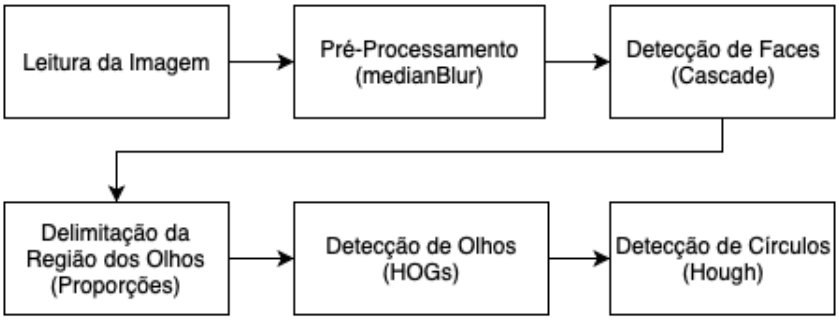
Fonte: Autor (2019).

Figura 32 – Etapas de execução de HOG2.



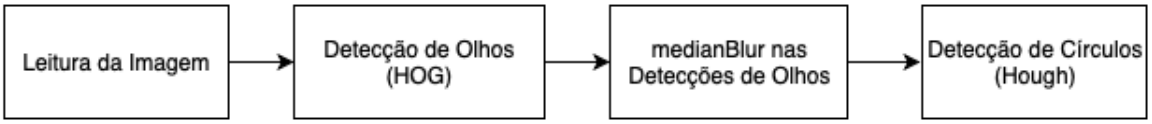
Fonte: Autor (2019).

Figura 33 – Etapas de execução de CIRC1 e CIRC2.



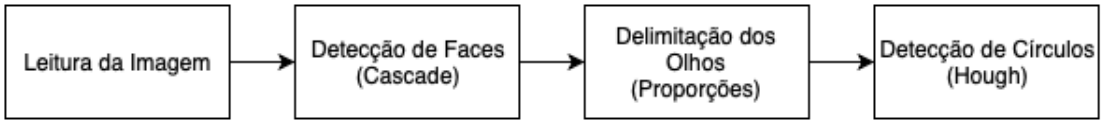
Fonte: Autor (2019).

Figura 34 – Etapas de execução de CIRC3.



Fonte: Autor (2019).

Figura 35 – Etapas de execução de CIRC4.



Fonte: Autor (2019).

APÊNDICE B

Este apêndice lista as imagens utilizadas para a avaliação dos algoritmos deste trabalho. São um total de trinta imagens, sendo que em quinze delas o usuário está com o olho aberto e no restante ele está com o olho fechado.

Figura 36 – Imagens utilizadas para a aplicação das técnicas

Fonte: Autor (2019).

APÊNDICE C

As Tabelas a seguir apresentam o resultado das execuções (dez execuções) de cada algoritmo em relação ao uso de processamento, memória e tempo de execução para analisar toda a base de imagens.

	VJ1	VJ2	VJ3	HOG1	HOG2	CIRC1	CIRC2	CIRC3	CIRC4
1	198,86	243,13	202,88	149,38	93,42	138,68	140,99	119,51	141,45
2	204,79	239,11	214,07	150,45	93,5	138,02	141,85	120,96	141,04
3	200,98	246,36	219,96	149,98	93,95	129,4	138.81	119,26	139,41
4	197,86	250,39	220,68	149,72	93,52	136,74	138,71	120,35	141,28
5	205,81	237,09	204,48	149,21	93,94	140,66	137,24	120,92	141,51
6	200,65	239,19	202,41	147,64	93,35	141,24	140,57	119,08	141,27
7	206,73	238,95	200,29	147,87	93,46	137,84	133,22	117,93	140,68
8	201,09	246,92	222,51	149,58	92,88	138,98	136,99	119,93	143,27
9	203,16	246,86	256,59	148,56	93,37	138,17	137,41	118,48	142,63
10	203,87	248,11	220,08	148,82	92,39	138,6	138,57	120,71	139,62
Média	**202,38**	**243,61**	**216.395**	**149.121**	**93.378**	**137.83**	**138.43**	**119.71**	**141.21**
Desvio Padrão	**2,95**	**4,70**	**16.60**	**0.90**	**0.46**	**3.24**	**2.46**	**1.04**	**1.17**

Tabela 9 – Dados obtidos em relação ao uso de CPU.

	VJ1	VJ2	VJ3	HOG1	HOG2	CIRC1	CIRC2	CIRC3	CIRC4
1	87,82	104,7	62,07	95,31	105,11	98,91	98,44	87,94	93,03
2	87,61	105,28	62,25	95,71	105,15	98,65	98,66	87,94	92,68
3	87,27	105,04	62,23	95,52	105,18	98,05	98,81	88,21	92,61
4	87,4	105,23	62,62	95,7	105,36	98,23	98,92	88,21	92,76
5	87,84	104,93	62,41	96,13	105,12	98,15	98,82	88,33	93,02
6	87,65	104,86	62,51	96,07	105,2	98,1	98,97	87,97	93,27
7	87.55	105,07	62,01	95,56	105,09	98,53	98,62	88,12	92,83
8	87,23	105,11	62,41	95,38	105,21	98,63	99,06	87,95	92,7
9	87,5	105,2	62,16	95,77	105,43	98,28	99,02	87,94	93,17
10	87,31	105,15	62,53	95,21	105,2	99,17	97,98	88,04	92,73
Média	**87,51**	**105,05**	**62,32**	**95,63**	**105,20**	**98,47**	**98,73**	**88,06**	**92,88**
Desvio Padrão	**0,21**	**0,18**	**0,20**	**0,30**	**0,10**	**0,37**	**0,32**	**0,14**	**0,22**

Tabela 10 – Dados obtidos em relação ao uso de memória.

	VJ1	VJ2	VJ3	HOG1	HOG2	CIRC1	CIRC2	CIRC3	CIRC4
1	3,821	5,124	2,787	3,897	10,973	3,816	3,777	8,033	3,736
2	3,871	5,051	2,681	3,779	10,984	3,778	3,82	8,026	3,794
3	3,983	4,813	2,543	3,85	10,886	4,379	3,86	8,13	3,808
4	3,828	4,855	2,54	3,832	11,064	3,869	3,825	7,894	3,701
5	3,969	5,158	2,775	3,886	10,954	3,974	3,89	7,859	3,619
6	3,957	4,872	2,823	4,006	10,962	3,753	3,943	7,978	3,734
7	3,794	5,259	3,032	3,951	11,017	3,785	4,084	7,952	3,832
8	3,766	4,904	2,532	3,886	10,998	3,804	3,953	8,056	3,631
9	3,838	4,864	2,588	3,823	10,973	3,789	3,779	8,087	3,684
10	3,918	4,769	2,569	3,847	11,381	3,818	3,809	8,028	3,686
Média	**3,87**	**4,96**	**2,68**	**3,87**	**11,02**	**3,87**	**3,87**	**8,00**	**3,72**
Desvio Padrão	**0,07**	**0,16**	**0,16**	**0,06**	**0,13**	**0,18**	**0,09**	**0,08**	**0,07**

Tabela 11 – Dados obtidos em relação ao tempo de execução.