UNIVERSIDADE ESTADUAL PAULISTA "JÚLIO DE MESQUITA FILHO"
FACULDADE DE CIÊNCIAS - CAMPUS BAURU
DEPARTAMENTO DE COMPUTAÇÃO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

MARCELO GARCIA NUÑEZ

SOFTWARE DE RASTREIO DE OLHAR COM MÚLTIPLAS INTERFACES

BAURU
Junho/2020

MARCELO GARCIA NUÑEZ

SOFTWARE DE RASTREIO DE OLHAR COM MÚLTIPLAS INTERFACES

Trabalho de Conclusão de Curso do Curso de Bacharelado em Ciência da Computação da Universidade Estadual Paulista "Júlio de Mesquita Filho", Faculdade de Ciências, Campus Bauru.
Orientadora: Dra. Simone das Graças Domingues Prado

BAURU
Junho/2020

Marcelo Garcia Nuñez *Software* de Rastreio de Olhar com Múltiplas Interfaces/ Marcelo Garcia Nuñez. – Bauru, Junho/2020- 39 p. : il. (algumas color.) ; 30 cm.
Orientadora: Dra. Simone das Graças Domingues Prado
Trabalho de Conclusão de Curso – Universidade Estadual Paulista "Júlio de Mesquita Filho"
Faculdade de Ciências
Ciência da Computação, Junho/2020.
1. Tecnologias Assistivas 2. Rastreamento de Olhar 3. *Eye Controllers* 4. Visão Computacional 5. Interfaces 6. *OpenCV* 7. *Python*

Marcelo Garcia Nuñez

Software de Rastreio de Olhar com Múltiplas Interfaces

Trabalho de Conclusão de Curso do Curso de Bacharelado em Ciência da Computação da Universidade Estadual Paulista "Júlio de Mesquita Filho", Faculdade de Ciências, Campus Bauru.

Banca Examinadora

Dra. Simone das Graças Domingues Prado
Orientadora
Departamento de Ciência da Computação
Faculdade de Ciências
Universidade Estadual Paulista "Júlio de Mesquita Filho"

Dr. Aparecido Nilceu Marana
Departamento de Ciência da Computação
Faculdade de Ciências
Universidade Estadual Paulista "Júlio de Mesquita Filho"

Dr. Kleber Rocha de Oliveira
Campus Experimental de Rosana
Universidade Estadual Paulista "Júlio de Mesquita Filho"

Bauru, 02 de Julho de 2020.

Dedico este trabalho ao meu avô Manuel Garcia Nuñez e para a minha avó Manuela Garcia Nuñez.

Agradecimentos

Gostaria de agradecer a minha mãe Angélica Ribeiro Cunha, meu pai Camilo Garcia Nuñez, o meu irmão Victor Garcia Nuñez, e os meus avós Manuela Garcia Nuñez e Manuel Garcia Nuñez. Aos meus amigos e colegas que estiveram sempre próximos e me ajudaram imensamente. Também aos excelentes professores e mentores que eu tive o prazer de conhecer, desde o fundamental até à universidade. Em especial à minha orientadora Simone das Graças Domingues Prado, por toda a sua ajuda para o desenvolvimento desse trabalho e conhecimento compartilhado. Aos meus amigos da *Really Long Birds*, que estamos trabalhando juntos para criar jogos incríveis.

Atrás de cada hesitada há um medo ou ansiedade, algumas vezes racional, outras vezes não. Sem o medo não há hesitada. Mas acabar com o medo não é importante, mas encarar ele sim.

- Julien Smith.

Resumo

Eye controllers são dispositivos utilizados em conjunto com aplicações de visão computacional para identificar onde aponta o olhar do usuário. É possível utilizar os dados obtidos de um *eye controller* como uma entrada de dados para controlar uma interface digital apenas com os olhos. Este é um recurso muito interessante no âmbito das tecnologias assistivas, que visam fornecer ferramentas e funcionalidades inclusivas para pessoas com necessidades especiais. *Eye controllers* são ferramentas importantes para pessoas com suas capacidades motoras comprometidas, mas que são capazes de efetuar movimentos com os olhos. Este trabalho revisa algoritmos e métodos envolvidos com visão computacional e rastreamento de olhar, assim como traz a proposta de um programa que permite ao usuário controlar apenas com os olhos interfaces digitais que podem ser facilmente customizadas. A implementação faz uso da biblioteca de visão computacional *OpenCV* em *Python* para efetuar o rastreamento de olhar, e para o desenvolvimento das interfaces é utilizado o *PyQt*. Foi desenvolvido um *software* capaz de rastrear a direção do olhar do usuário. As funções e lógica por trás dos processos desse método são descritas passo a passo, assim como é avaliado o *software* considerando as características de sua implementação e possíveis alternativas e melhorias.

Palavras-chave: Tecnologias Assistivas, Rastreamento de Olhar, *Eye Controllers*, Visão Computacional, Interfaces, *OpenCV*, *Python*.

Abstract

Eye controllers are devices that are used together with computer vision applications to identify where does the gaze of the user points towards. It is possible to use the data obtained from an eye controller to use as an input and control a digital interface only using the eyes. This is an interesting resource in the field of assistive technologies, which aims to provide inclusive tools and functionalities for people with special needs. Eye controllers are specially important for people with compromised motor functions, but are still capable of moving their eyes. This work revise some algorithms and methods utilized in computer vision for gaze tracking, and also proposes the development of a program that allows the user to control only with their eyes digital interfaces that are easily customizable. The implementation makes use of the computer vision *Python* library *OpenCV*, and also uses *PyQt* for developing the interfaces. An application capable of tracking the direction of the user's gaze was developed. The functions and logic behind the process of this method are described at each step. It is also discussed about some characteristics of the software's implementation, and about possible alternatives and enhancements.

Keywords: Assistive Technologies, Gaze Tracking, Eye Controllers, Computer Vision, Interfaces, *OpenCV*, *Python*.

Lista de figuras

Sumário

1 Introdução

Rastreamento ocular é um tópico estudado por vários campos do conhecimento, como Neurologia, Psicologia, e Computação. Isto devido sua vasta área de aplicação tanto para pesquisas quanto produtos (MORIMOTO; MIMICA, 2004). Uma definição de rastreamento ocular é o de um conjunto de tecnologias que possibilitam a captura e análise de dados coletados dos movimentos oculares (BARRETO, 2012). Rastreamento ocular é aplicado dentre vários campos, como por exemplo realidade virtual, diagnóstico de doenças, jogos eletrônicos, robótica, estudos de comportamento humano, *marketing*, e também no desenvolvimento de tecnologias assistivas como *eye controllers* (PUTRA; CAHYAWAN; PERDANA, 2011).

Eye trackers são dispositivos capazes de estimar a direção do olhar de uma pessoa para fazer rastreamento de olhar (MORIMOTO; MIMICA, 2004). Existem diversas aplicações para os *eye trackers*, de modo que esses equipamentos podem ser chamados de *eye controllers* quando utilizados como ferramentas de interação, de modo a permitir o usuário interagir com algum dispositivo eletrônico apenas com os olhos. Este uso dos *eye controllers* é o que torna esta tecnologia de grande interesse como uma tecnologia assistiva, principalmente para indivíduos com necessidades especiais e condições motoras comprometidas (BARRETO, 2012).

Um exemplo de condição que pode comprometer seriamente as habilidades motoras de um indivíduo é a paralisia cerebral. A qual se trata de um retardo no desenvolvimento psicomotor que pode se manifestar de diversas maneiras, seja como alterações pequenas em como se gesticula, ou danos mais severos, como causar a incapacidade de movimento e fala (FERREIRA et al., 2011). Para pessoas com paralisia cerebral, e que dispõem somente dos movimento dos olhos, *eye controllers* podem ser ferramentas extremamente úteis para fornecer mais autonomia ao permitir que o usuário interaja com uma interface virtual para múltiplos fins.

A interação entre humano-computador (IHC) é uma área que se dedica à pesquisa de todos os aspectos da interação entre humano e computador. Entre as questões que compõem a IHC, um fator muito importante é como um indivíduo se comunica com uma interface. Sendo uma interface o local em que duas entidades se comunicam e interagem (PRATES; BARBOSA, 2007).

Eye controllers, quando usados em um *software* que possui uma interface própria e com suporte para este tipo de equipamento como dispositivo de entrada de dados, podem fornecer teclados virtuais para digitação, simular um ponteiro de mouse com cliques, ou qualquer outra funcionalidade que o *software* permitir apenas com o movimento dos olhos.

Eye trackers e *eye controllers* podem funcionar com dispositivos diferentes e que variam também em sua utilização assim como outros aspectos, como conforto, tempo de uso projetado,

custo e precisão. Portanto, as técnicas de cada dispositivo são usualmente divididas em duas categorias, intrusivos e não intrusivos (MORIMOTO; MIMICA, 2004).

Técnicas de rastreamento ocular intrusivas são aquelas que requerem contato físico do equipamento com o usuário, de modo que apesar de serem usualmente mais precisas, são menos utilizadas caso o rastreio ocular precise ocorrer por mais tempo (MORIMOTO; MIMICA, 2004). Dentre estas técnicas está o uso de lentes de contato com um espelho, ou sensores magnéticos, que captam com precisão os movimentos oculares. Também existem técnicas que fazem uso de eletrodos, conhecidas como Electro-Oculograma, captam o potencial elétrico do olho para conseguir dados relativos a sua movimentação (BARRETO, 2012).

As técnicas de rastreamento ocular não intrusivas usualmente são baseadas em reconhecimento de padrões em vídeo, e apesar de serem mais confortáveis para usos prolongados, não apresentam a mesma precisão que as técnicas intrusivas (MORIMOTO; MIMICA, 2004). Dispositivos de *eye tracking* e *eye control* captam a imagem dos olhos e procuram o centro da pupila como ponto de referência para determinar a direção em que o olho está apontando. Alguns *eye trackers* são capazes de emitir luz infravermelha, a qual não causa desconforto para o usuário e permite se fazer o rastreamento de olhar de forma mais precisa (BARRETO, 2012).

A forma como o *software* efetua o rastreamento ocular por vídeo entra em um campo das Ciências da Computação chamado Visão Computacional, o qual se dedica aos estudos de processamento de imagens ou vídeos para a obtenção de dados. *OpenCV* é uma biblioteca para se trabalhar com visão computacional, criada em 1999 pela *Intel* como um projeto em código aberto e de livre uso (CULJAK et al., 2012). *OpenCV* ainda é amplamente utilizada, e possui versões tanto em *C++* quanto *Python*.

1.1 Problema

Apesar da grande quantidade de dispositivos como computadores e celulares, assim como *softwares* que se encontram no mercado, não são todos que dispõem de recursos de acessibilidade para indivíduos com necessidades especiais. Diante este cenário, existem diversas pesquisas no campo de tecnologias assistivas para adaptar *softwares* e dispositivos de modo que se isto torne possível ou facilite o seu uso.

No âmbito de tecnologias assistivas em *eye controllers*, o que se torna uma complicação é a criação de uma interface para o *software* de *eye controller* que lide com a extensa variedade de programas que existem, que devido seus propósitos e funcionalidades únicas podem exigir uma interface própria, e modo de uso próprio. Desta forma não se torna adequado criar uma única interface para o *software* de *eye controller*, isto para que este *software* não se torne restrito à uma forma de interface. Portanto, um aspecto importante é o de que o *software* de *eye controller* possua uma interface que possa ser adaptável, no sentido de que o usuário pode escolher qual modelo deseja usar. Como escolher usar uma interface controlando um cursor

com os olhos com um mouse, ou selecionar entre ícones um por um movendo os olhos para a direção do próximo ícone desejado.

Um outro fator que pode impossibilitar indivíduos de utilizar *eye controllers*, além da falta de suporte e compatibilidade que possa existir com outros *softwares*, está na obtenção desses dispositivos. *Eye trackers* e *eye controllers* podem ser equipamentos caros, e se tornam ainda mais custosos se considerar que para usar um também é preciso a compra de outros dispositivos, como um computador e monitor. Logo se torna interessante ponderar formas alternativas de utilizar *eye controllers* com dispositivos mais acessíveis, como *webcams*.

Uma forma mais acessível de *eye controller* seria pela construção de um *software* capaz de reconhecer onde está a pupila do usuário por meio de um dispositivo de vídeo comum como uma *webcam*. Isto de modo a oferecer várias interfaces que o usuário possa utilizar para interagir com olhar, as que servem como um intermediário entre a entrada de dados pela *webcam* e o programa final que se deseja utilizar.

1.2 Justificativa

O uso de computadores, celulares, e dispositivos inteligentes estão cada vez mais presentes no cotidiano e impactam drasticamente a vida das pessoas, de modo que afetam em como se comunicam, produzem, se expressam e aprendem. Logo a incapacidade de utilizar destes dispositivos pode se mostrar como um fator que exclui e limita a autonomia do indivíduo. Dentre os fatores que impedem o acesso à recursos tecnológicos pode-se citar: baixo poder aquisitivo, questões geográficas, ou por limitações na condição física do indivíduo.

Desta forma, mesmo com a existência de alguma tecnologia assistiva que se encontra disponível, torna-se interessante procurar por formas alternativas e ainda mais acessíveis para aqueles que não possuem poder aquisitivo o suficiente para obter estes dispositivos. Por isso, propõem-se realizar o rastreio de olhar utilizando uma *webcam*, o qual é um aparelho de vídeo que se encontra com maior facilidade e com melhor custo que um *eye tracker* ou *eye controller*.

Outro aspecto relevante do projeto é o de que embora existam *softwares* de rastreamento de olhar para utilizar o movimento dos olhos como um cursor de um mouse, ou digitar teclas em um teclado virtual, existem muitos *softwares* que exigem formas diferentes de entrada de dados. Como resultado, apesar de existirem mecanismos de controlar uma interface com o olhar, pode se deparar com a situação de que não há *softwares* adequados para usar com outros programas. Desta forma um aspecto interessante do projeto é o de possibilitar a troca de interfaces de acordo com a aplicação que se deseja utilizar.

Por exemplo, se o indivíduo deseja usar um *software* de estação de trabalho de áudio digital, os quais são dedicados para composição de música, o usuário pode desejar utilizar uma interface para interagir com os olhos que seja semelhante às teclas de um piano, e não uma

que se parece com as teclas de um computador.

Outro aspecto é que com os olhos se consegue apenas selecionar uma tecla de cada vez em um *eye controller*, caso a pessoa esteja jogando um jogo digital por exemplo, onde se precisa pressionar uma tecla para correr e outra para selecionar a direção, isto não seria possível em uma interface de teclado convencional. Mas se fosse uma interface alternativa, onde as teclas de selecionar a direção e o botão de correr efetuam a entrada de dados simultaneamente, isto seria possível.

Logo, permitir formas alternativas e flexíveis de interagir com um *software* fazem parte da questão de democratizar o uso dessas tecnologias. E permitir isso com equipamentos que exigem menor poder aquisitivo como *webcams* também é um passo importante que o mais pessoas tenham acesso a *eye controllers* próprios.

1.3 Objetivos

O objetivo deste trabalho é o de desenvolver um *software* que por meio de um vídeo obtido por uma *webcam* simule uma entrada de dados baseada no posicionamento da pupila e íris do usuário utilizando a biblioteca *OpenCV* em *Python*. O intuito é utilizar esta entrada de dados para permitir a interação com múltiplas interfaces..

1.4 Objetivos Específicos

- Estudar conceitos de rastreamento ocular e uso da biblioteca *OpenCV*;
- Efetuar o rastreio da posição da pupila do usuário;
- Criar interfaces usando *PyQt* e *QtDesigner*;
- Elaborar um processo de calibração para obter uma estimativa de onde o usuário está olhando;
- Construir um *software* para testar uma entrada de dados com base no posicionamento da pupila.

1.5 Desafios

Como a calibração do posicionamento do cursor sobre a interface de acordo com a movimentação da pupila requer alguns cálculos complexos que não são viáveis na implementação do programa desenvolvido diante o escopo. Desta forma a implementação fará uso de formas mais simplificadas de interface do programa.

Como resultado existirão valores de alguns parâmetros definidos empiricamente baseados nos equipamentos e dispositivos utilizados para o desenvolvimento e testes do *software*.

1.6 Organização da Monografia

Este trabalho se organiza em capítulos, onde este é o primeiro e tem o objetivo de introduzir o trabalho.

Cada capítulo deste trabalho visa abordar seu conteúdo de acordo com seus respectivos temas.

O Capítulo 2 visa introduzir a fundamentação teórica e os conceitos de rastreamento de olhar, incluindo algumas questões sobre detecção facial.

O Capítulo 3 é sobre as ferramentas, softwares e dispositivos utilizados no trabalho.

O Capítulo 4 visa relatar sobre o desenvolvimento da implementação do *software* de rastreamento de olhar e interação com interfaces customizáveis.

O Capítulo 5 visa concluir este trabalho e avaliar a implementação diante as questões teóricas apresentadas.

2 Fundamentação Teórica

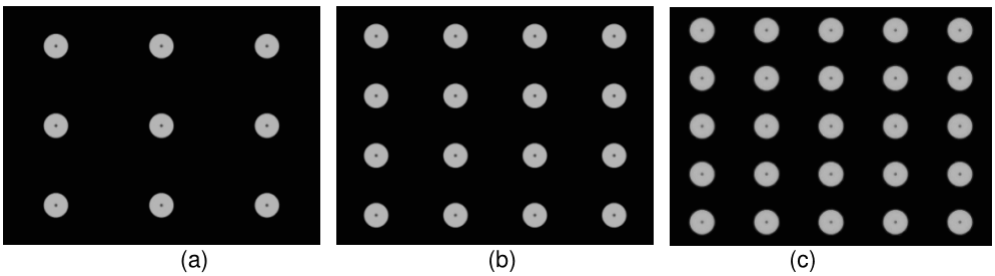
2.1 Rastreamento de Olhar

Para efetuar o rastreamento de olhar surgem várias questões em como faze-lo. Foi mencionado no Capítulo 1 que existem formas intrusivas e não intrusivas, sendo o rastreamento não intrusivo feito usualmente com base em visão computacional (MORIMOTO; MIMICA, 2004).

O processo de rastreamento de olhar por visão computacional usualmente se baseia em algumas etapas. Para identificar o ponto onde alguém está olhando primeiramente se precisa identificar a pupila do usuário. (HöFFNER, 2018).

Uma vez obtido o posicionamento da pupila é preciso estimar o ponto onde o usuário está olhando, o que usualmente se trata de uma transformação do posicionamento da pupila em um ponto no monitor alvo. Isto pode ser feito com um processo de calibração, que se baseia em mapear o posicionamento da pupila em relação aos pontos de referência apresentados no monitor (*Pixels* de Calibração). Como estes pontos de referência são localizações conhecidas no monitor, é possível obter coeficientes para serem utilizados na transformação de coordenadas da pupila em coordenadas no monitor. Esta forma de calibração entretanto requer que a cabeça se mantenha em uma posição fixa (PUTRA; CAHYAWAN; PERDANA, 2011). A Figura 1 mostra o exemplo de uma interface utilizada para efetuar a calibração de um software de rastreio ocular. As Figura 1(a), 1(b) e 1(c) mostram imagens utilizadas para calibrações utilizando 9, 16 e 25 pontos respectivamente.

Figura 1 – Pontos de Calibração para Transformação de Coordenadas da Pupila em Coordenadas no Monitor



Fonte: PUTRA; CAHYAWAN; PERDANA (2011)

A identificação da posição apontada pelo olhar utilizando um modelo geométrico requer mais informações do que apenas o posicionamento da pupila. É preciso efetuar uma estimativa da distância do usuário com o monitor, assim como o posicionamento da cabeça. Uma vez obtidos estes dados se efetua um *raycast* partindo da pupila. *Raycast* é como simular um

projetil saindo do olho que colide com o ponto que se está olhando, e o resultado é traduzido para ser algum ponto no monitor (HöFFNER, 2018).

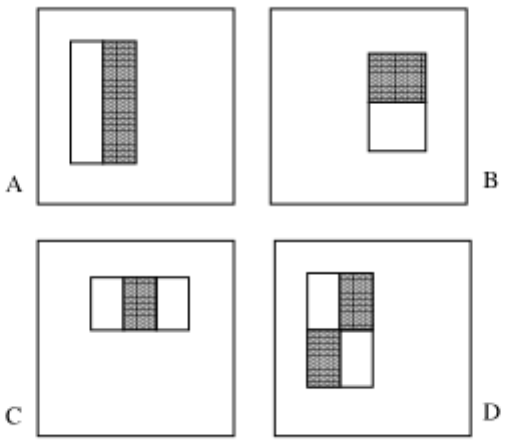
2.2 Detecção Facial

O desafio que vários métodos de rastreamento de olhar visam superar é de como obter uma solução eficaz, e com baixo custo computacional. A detecção facial é uma etapa que precede o rastreamento da pupila, já que a partir da face do usuário é possível determinar o local de busca pelo olho e sua pupila para algum ponto dentro dessa área da imagem. A detecção facial também é uma etapa muito importante para métodos que utilizam um modelo geométrico, onde o posicionamento da cabeça é uma informação importante (HöFFNER, 2018).

Um algoritmo de detecção facial é o Viola-Jones, o qual foi proposto proposto por Paul Viola e Michael Jones (2001). O algoritmo faz uso de uma abordagem com aprendizado de máquina para identificar feições *Harr-Like* utilizando classificadores em cascata (VIOLA; JONES, 2001).

Feições *Harr-Like* possuem esse nome pois seu conceito é inspirado na Transformada de Haar proposta por Alfréd Haar (BAGGIO, 2015). O Viola-Jones utiliza de classificadores para encontrar estas feições em uma imagem comparando regiões retangulares dentro da própria imagem. Ao subtrair e comparar estes pedaços de imagem se testa se as regiões verificadas de fato representam a feição que o classificador está procurando (COMPUTERPHILE, 2018). Esta comparação de regiões em retângulos pode considerar de dois até quatro retângulos como apresentado na Figura 2 (VIOLA; JONES, 2001).

Figura 2 – Regiões Retangulares de Comparação

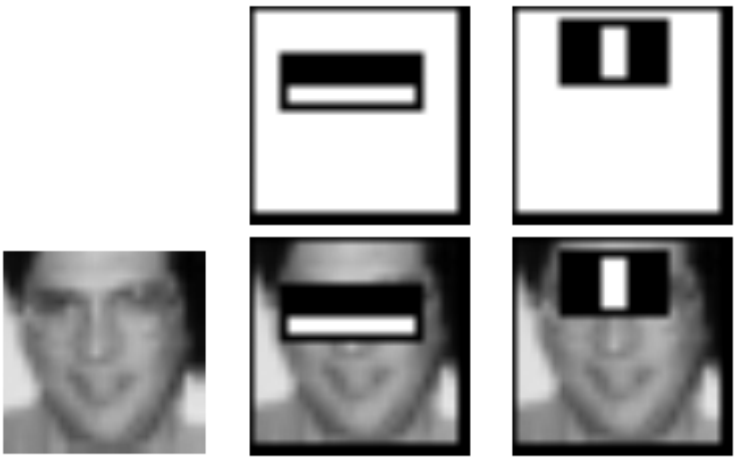


Fonte: VIOLA; JONES (2001)

As feições *Harr-Like* consideram que existem várias similaridades e padrões no rosto humano. A Figura 3 é um exemplo que mostra como que em uma imagem a região dos olhos

costuma ser mais escura do que a região superior das bochechas, e também como a região dos olhos também é mais escura do que a região do nariz (VIOLA; JONES, 2001).

Figura 3 – Feições Haar-Like



Fonte: VIOLA; JONES (2001)

O processo de classificação se refere a atribuição de uma classe a uma informação. De modo que esta classe se refere ao fenômeno que deseja que o classificador seja treinado para aprender (LORENA; CARVALHO, 2003). Os classificadores precisam passar por um processo de aprendizado que pode ser difícil de balancear questões como velocidade de processamento e evitar a aprovação de falsos positivos (COMPUTERPHILE, 2018).

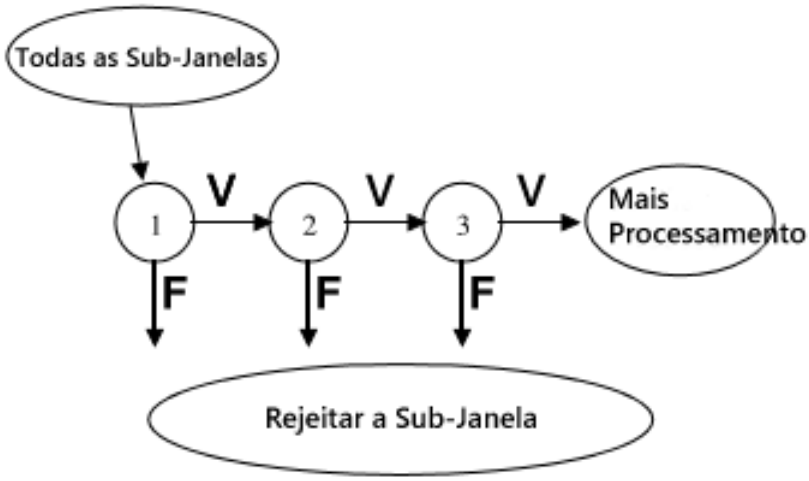
O Viola-Jones funciona como uma árvore de decisões degenerada, onde cada classificador se apresenta como um nó dessa árvore, e cada nó possui um único filho. Para que uma região seja considerada uma face, as verificações de cada classificador da árvore precisam ser satisfeitas. Caso algum classificador não seja satisfeito, a hipótese de que a região poderia ser uma face é descartada (COMPUTERPHILE, 2018).

Este processo é descrito na Figura 4, na qual os símbolos V significam que a feição verificada pelo classificador foi aprovada, e F que a verificação foi reprovada. A Figura 4 mostra que quando uma sub-janela é testada, caso ela falhe no teste de uma feição no primeiro classificador toda a sub-janela é rejeitada, e como consequência todas as verificações dos demais classificadores e processamento posterior não ocorrem. Entretanto se as feições são testadas e aprovadas consecutivamente pelos classificadores, se chega ao estado que se aprova que naquela região da imagem há uma face (VIOLA; JONES, 2001).

Como cada classificador compreende uma feição bem vaga de um rosto, é pela união desses classificadores que se obtém uma indicação mais precisa do que é um rosto. Como são necessárias várias operações, nesse método performance é algo bem crucial. Esse é um dos

motivos do algoritmo procurar descartar rapidamente as áreas em que não se tem uma face (COMPUTERPHILE, 2018).

Figura 4 – Árvore de Decisões Degenerada

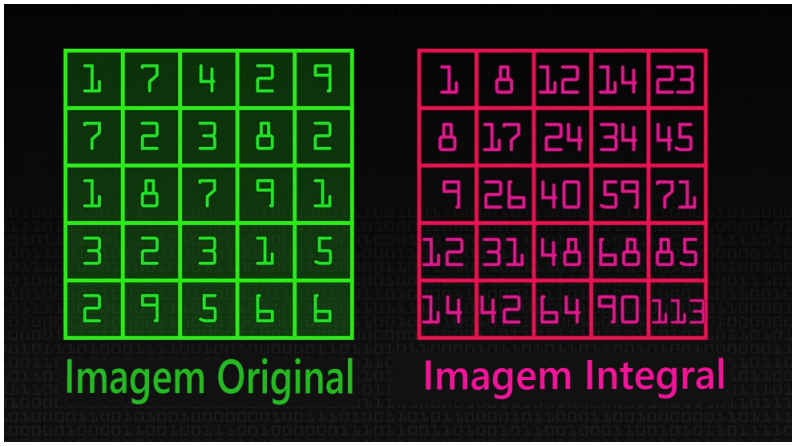


Fonte: Traduzido de VIOLA; JONES (2001)

O Viola-Jones precisa comparar várias áreas para serem aprovados ou não pelo classificador. Isso seria uma tarefa muito árdua se para comparar as áreas fosse necessário somar o valor de cada pixel da área para obter seu valor médio para a comparação. Por isso o uso de uma imagem integral auxilia na performance desse algoritmo (COMPUTERPHILE, 2018).

A imagem integral funciona de forma que cada valor da casa representando o *pixel* se torna o seu próprio valor somado com todos os demais valores que estão à "esquerda" e "acima" do mesmo. Desta forma, o valor do pixel equivale ao valor da área que que cobre todos os *pixels* que estão à esquerda e acima dele com o valor original do próprio *pixel* incluso. Isso como mostra a Figura 5, em que na esquerda há uma representação de uma imagem com seus valores originais e na direita a imagem integral gerada a partir da imagem na esquerda.

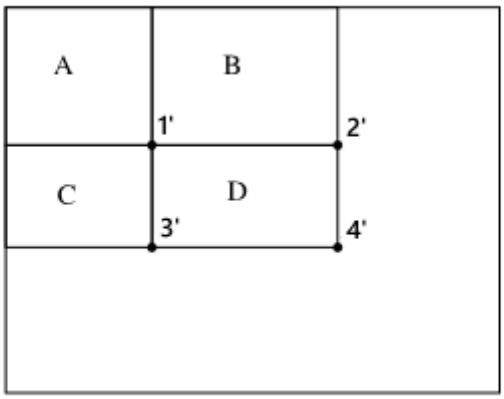
Figura 5 – Imagem Integral



Fonte: Traduzido de COMPUTERPHILE (2018)

A Figura 6 exemplifica como as imagens integrais são utilizadas para calcular áreas. O valor da imagem integral em 1' é a soma dos valores da imagem original na área A. O valor na posição 2' é igual a área de A + B, na posição 3' é A + C, e na posição 4' é A + B + C + D. Uma vez que se possui os valores pré calculados da imagem integral de 1', 2', 3' e 4', a soma para obter a área D é 1' - 3' - 2' + 1' (VIOLA; JONES, 2001).

Figura 6 – Cálculo de uma Área Utilizando Imagem Integral

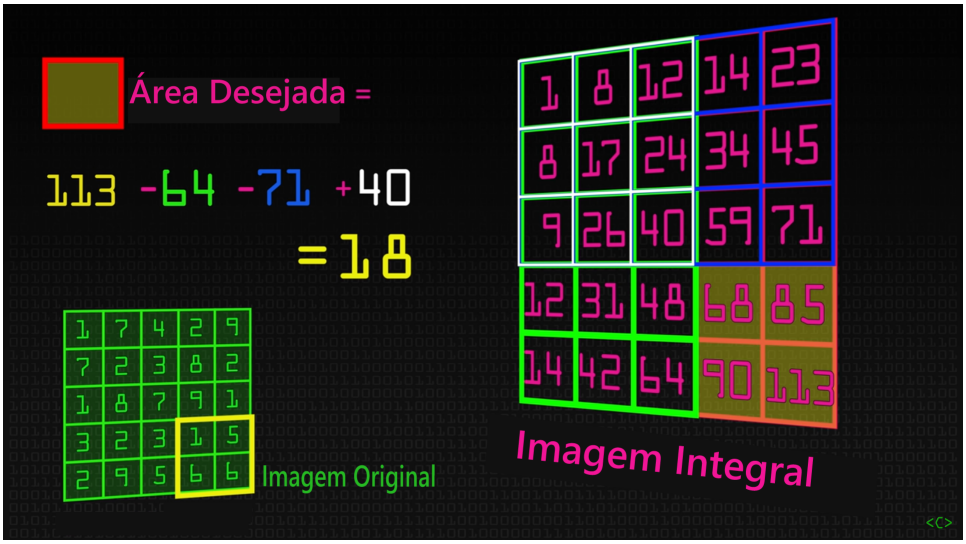


Fonte: Editado de VIOLA; JONES (2001)

A Figura 7 é outro exemplo porém com números, no qual a casa com numero 40 equivale ao pixel de índice 1 do exemplo anterior, a casa de número 71 ao 2, a casa 64 ao 3, e a casa 113 ao 4. A área desejada, a qual está com uma coloração amarelada é obtida pela soma de 4' - 3' - 2' + 1', o que equivale no exemplo à 113 - 64 - 71 + 40. O valor da área por fim é 18, o qual está condizente pelo que mostram os valores da imagem original, cuja soma seria $1 + 5 + 6 + 6 = 18$. Como o Viola-Jones precisa efetuar muitas comparações de áreas, reduzir a forma de obter o valor dessas para uma soma de quatro operações depois de obter a

imagem integral auxilia muito a performance do algoritmo. Neste exemplo a soma total da área também é obtida por apenas quatro operações, de modo que o uso de imagens integrais é benéfico em situações onde as áreas são maiores, pois não importa o tamanho basta fazer a operação de 1' - 3' - 2' + 1' (COMPUTERPHILE, 2018).

Figura 7 – Exemplo do Cálculo de uma área utilizando Imagem Integral



Fonte: Traduzido de COMPUTERPHILE (2018)

2.3 Reconhecimento de Pupila

Como mencionado anteriormente, *eye trackers* ou *eye controllers* que se baseiam em reconhecimento de vídeo usualmente utilizam a pupila do usuário como um ponto de referência para estimar a direção do olhar, de modo que existem múltiplos meios de obter esta informação.

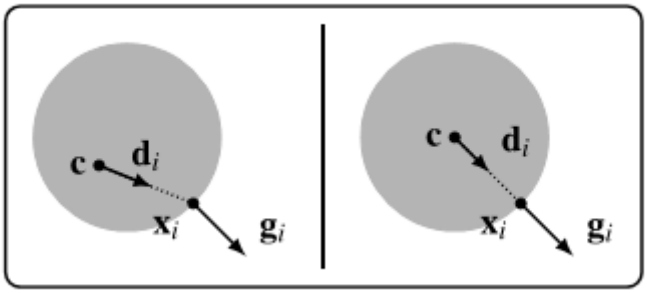
Um método de reconhecimento de pupila eficaz é o proposto por TIMM; BARTH (2011), o qual determina a posição da pupila utilizando vetores e gradientes. Os métodos utilizados partem de alguns pressupostos de como é a aparência e o formato do olho, e com isso aplicam funções matemáticas para determinar a posição da pupila (TIMM; BARTH, 2011).

A anatomia do olho humano é um fator levado em consideração pelo método. Uma parte do olho relevante para o método é a íris, cuja aparência é a de um segmento colorido do olho cercado por uma região branca, que se chama esclera. A borda entre a íris e a esclera possui um grande contraste entre as cores escuras da íris com as cores claras da esclera, de forma que se gerados vetor gradientes nesta fronteira, o vetor gradiente aponta em direção para a esclera, ou seja para uma direção para fora da íris (HöFFNER, 2018).

Os gradientes obtidos são utilizados para estimar um ponto do centro da íris que é onde está localizada a pupila. A Figura 8 auxilia a visualizar este procedimento, no qual se estima um ponto que pode ser a pupila chamado C, nesta posição é criado um vetor de deslocamento

Di apontando para o ponto Xi onde há o vetor gradiente Gi. Se Di e Gi apontam para a mesma direção isso indica que C é um ponto próximo de ser a pupila, isto é avaliado efetuando um produto escalar entre os vetores Di e Gi. Estes produtos escalares são avaliados de modo que o ponto com mais gradientes apontando para a mesma direção é o ponto definido como o centro da pupila (HöFFNER, 2018).

Figura 8 – Exemplo de Vetores Gradientes para Determinar a Posição da Pupila



Fonte: HöFFNER (2018)

Este método que utiliza gradientes proposto por TIMM; BARTH (2011) se mostra eficiente e eficaz. Assim como funciona em cenários com iluminação pouco favorável, e em situações onde a imagem ou vídeo apresenta baixa qualidade. Existem mais conceitos envolvidos em sua implementação que não foram explicados, mas estão disponíveis no próprio artigo (TIMM; BARTH, 2011).

3 Ferramentas Utilizadas

Para a o desenvolvimento do trabalho foram utilizados vários *softwares*, ferramentas, dispositivos e modelos que serão listados abaixo.

3.0.1 *Python* e Bibliotecas

A linguagem de programação utilizada principalmente para a lógica do programa desenvolvido foi *Python*, a qual é uma linguagem orientada a objetos que funciona por meio de *scripts* (PYTHON-SOFTWARE-FOUNDATION, 2020). *Python* é uma linguagem popular, e existem bastante recursos na internet como tutoriais e bibliotecas para se utilizar. A versão usada para o programa foi a 3.6.9.

Para construir a interface foi utilizado o *binding* de *Qt* para *Python PyQt*. *Qt* se trata de um *framework* para criação de interfaces com suporte à linguagem *C++*, enquanto *PyQt* permite que se utilize as funções do *Qt* utilizando *Python* (RIVERBANK-COMPUTING, 2020).

OpenCV é uma biblioteca *open-source* para visão computacional e aprendizado de máquina. Esta biblioteca possui suporte para *Python*, *C++* e *Java*, assim como possui implementações prontas de vários algoritmos (CULJAK et al., 2012). Como por exemplo há uma versão implementada própria do Viola-Jones descrito na fundamentação teórica (MORDVINTSEV; REVISION, 2013). A versão utilizada do *OpenCV* foi a 4.2.0.

3.1 Softwares

Foi utilizado para o trabalho o sistema operacional KDE Neon na versão 5.19. KDE Neon é uma distribuição Linux construída por cima do Ubuntu, e o mesmo é um projeto *open-source* providenciado pela comunidade KDE.

Como mencionado anteriormente, a interface do programa é feita utilizando *Qt*. A interface pode ser criada por meio de uma arquivo de extensão .ui, o qual é escrito em uma linguagem *XML* (*Extensible Markup Language*), no qual se organiza o posicionamento dos *widgets* do *Qt* de forma bem similar com um *HTML*.

O *QtDesigner* é uma ferramenta que permite a criação de arquivos de extensão .ui para *Qt* de forma visual, prática e simplificada.

3.2 Modelos

Na implementação foi utilizado um um modelo pré treinado que localiza 68 pontos de referência em um rosto. O modelo está disponível na biblioteca *Dlib*, também utilizada no projeto. Esta biblioteca faz parte de um projeto *open-source* e está disponível para *Python*, ela proporciona vários algoritmos e modelos para uso em programas de diversos perfis, seja aprendizado de máquina, ou uso em questões de álgebra linear (KING, 2009).

3.3 Dispositivos

Foi utilizado para testes uma *webcam* Logitech C920 HD Pro, que é capaz de capturar imagens a 1080p em 30 quadros por segundo (LOGITECH, 2020).

4 Desenvolvimento

O programa desenvolvido para este trabalho fornece uma breve demonstração do uso de uma *webcam* como um *eye controller*. O *software* é capaz de reconhecer entradas de dados a partir dos olhos, como identificar a direção do olhar no eixo horizontal, e identificar se o olho está fechado.

A implementação dos métodos de rastreio ocular foram muito baseados no material fornecido por CANU (2019) em "Eye motion tracking – OpenCV with Python". Tanto em questão de código quanto abordagem para a solução de problemas (CANU, 2019).

4.1 Detecção Facial e Pontos de Referência

Primeiramente são configurados o detector de faces e o preditor de pontos de referência. Ambos utilizando implementações do *Dlib*, e o preditor configurado para usar o modelo de 68 pontos "shape_predictor_68_face_landmarks.dat". Isto como mostra o Código 4.1.

Código 4.1 – Inicialização do Detector e Preditor

```
# Detector sendo inicializado
detector = dlib.get_frontal_face_detector()
# Preditor sendo inicializado
predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")
```

Fonte: Baseado em CANU (2019)

O detector de faces é utilizado a cada quadro do video da *webcam* para encontrar um rosto na imagem. Uma vez encontrado, é aplicado o preditor para encontrar demarcar alguns pontos de referência no rosto encontrado. O Código 4.2 é um trecho que se trata sobre estes procedimentos.

Código 4.2 – Uso do Detector e Preditor

```
# Imagem da webcam e convertida para escala de cinza
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
# Detector retorna todas as faces que encontrou na imagem do quadro atual
faces = detector(gray)
for face in faces:
    # Variaveis armazenando o posicionamento da face na imagem
    xi, yi = face.left(), face.top()
    xf, yf = face.right(), face.bottom()

```

```
10    # Obtendo os pontos de referencia na variavel landmarks. E fornecido por
          uma analise do preditor
11    landmarks = preditor(gray, face)
```

Fonte: Baseado em CANU (2019)

A Figura 9 mostra a posição dos pontos que o preditor com o modelo de 68 pontos consegue encontrar, assim como o índice de cada ponto. O que torna conhecer estas posições extremamente útil é que a partir dela se é possível obter a posição onde estão os olhos e restringir a busca pela pupila na região de cada olho.

Figura 9 – Modelo de Formas de 68 Pontos de Referência e seus Índices

Fonte: HöFFNER (2018)

Como é possível ver na Figura 9, os índices que correspondem ao olho esquerdo são os de número 36, 37, 38, 39, 40, 41. Isto enquanto os índices que correspondem ao olho direito são os de número 42, 43, 44, 45, 46, 47.

Com estes pontos é possível obter algumas informações interessantes. Pode-se tanto isolar para que apenas a região no interior desses pontos seja considerada para encontrar a pupila. Como também proporcionar uma forma básica de identificar se o usuário está ou não de olhos fechados ou piscando.

O processo de identificar se o usuário está ou não de olhos fechados ou piscando está descrito no segmento Código 4.3. E pode ser útil caso seja uma informação que deseje utilizar como uma forma de entrada de dados. Por exemplo se a interface está com um botão selecionado o usuário piscando de certa forma poderia seleciona-lo desta forma. Porém isto precisa ser avaliado com mais cuidado para que seja algo prático e confortável para o usuário.

Código 4.3 – Uso dos Pontos de Referência

```
def get_piscada_ratio(eye_points, facial_landmarks):
    # Ponto mais a esquerda do olho
    # Olho esquerdo: Ponto 37
    # Olho direito: Ponto 43
    left_point = (facial_landmarks.part(eye_points[0]).x,
        facial_landmarks.part(eye_points[0]).y)
    # Ponto mais a direita do olho
    # Olho esquerdo: Ponto 40
    # Olho direito: Ponto 46
    right_point = (facial_landmarks.part(eye_points[3]).x,
        facial_landmarks.part(eye_points[3]).y)

    # Ponto mdio entre os dois pontos superiores do olho.
    # Olho esquerdo: Pontos 38 e 39
    # Olho direito: Pontos 44 e 45
    center_top = midpoint(facial_landmarks.part(eye_points[1]),
        facial_landmarks.part(eye_points[2]))
    # Ponto medio entre os dois pontos inferiores do olho.
    # Olho esquerdo: Pontos 41 e 42
    # Olho direito: Pontos 47 e 48
    center_bottom = midpoint(facial_landmarks.part(eye_points[5]),
        facial_landmarks.part(eye_points[4]))

    # Distancia horizontal obtida pela hipotenusa da distancia entre o ponto
    # mais a esquerda e o ponto mais a direita do olho no eixo X e no eixo Y.
    hor_line_lenght = hypot((left_point[0] - right_point[0]), (left_point[1] -
        right_point[1]))
    # Distancia vertical obtida pela hipotenusa da distancia entre
    # o ponto medio superior e o ponto medio inferior do olho no eixo X e no
        eixo Y.
    ver_line_lenght = hypot((center_top[0] - center_bottom[0]), (center_top[1]
        - center_bottom[1]))

    # Razao entre a distancia vertical e a distancia horizontal
    # Se a distancia vertical acaba sendo baixo isso indica que a pessoa pode
        estar de olhos fechados.
    ratio = hor_line_lenght / ver_line_lenght
    return ratio

# E obtida a razao entre a distancia vertical e a distancia horizontal do olho
    esquerdo
```

```
olho_esquerdo_ratio = get_piscada_ratio([37, 38, 39, 40, 41, 42], landmarks)
# E obtida a razao entre a distancia vertical e a distancia horizontal do olho direito
olho_direito_ratio = get_piscada_ratio([43, 44, 45, 46, 47, 48], landmarks)

# E a media entre a razao obtida do olho esquerdo e a razo obtida olho esquerdo
piscada_ratio = (left_eye_ratio + right_eye_ratio) / 2
if piscada_ratio > parametro_arbitrrio_piscada:
    # Se piscada_ratio for maior que o parametro de valor empirico de comparaao o usurio piscou ou esta de olhos fechados
    PiscouOuOlhosFechados()
```

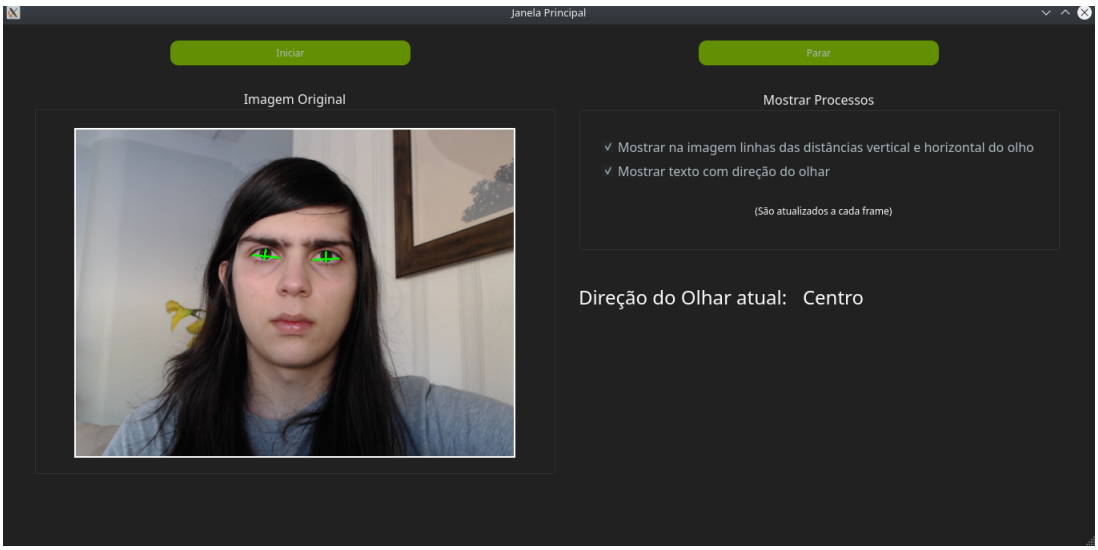
Fonte: Baseado em CANU (2019)

O que acontece nesse segmento de código é que se divide a distância horizontal dos olhos pela distância vertical dos olhos para se obter a razão entre os dois. Se o valor da razão entre a distância horizontal e a distância vertical for maior que um valor obtido empiricamente, isso quer dizer os olhos se encontram fechados, ou perto de estarem fechados. A distância horizontal se dá pela distância entre os pontos de índice 37 e 40 no olho esquerdo e 43 e 46 no olho direito. Enquanto a distância vertical se dá pela distância entre o ponto médio entre 38 e 39 e o ponto médio entre 41 e 42. No olho esquerdo a distância vertical se dá entre o ponto médio entre 44 e 45 e o ponto médio entre 37 e 38.

A Figura 10 e Figura 11 mostram o *software* em funcionamento com uma representação visual das distâncias vertical e horizontal dos olhos quando estão abertos e fechados. A imagem do rosto existente nas figuras é a própria imagem capturada pela *webcam* com alguns elementos visuais de auxílio adicionados posteriormente.

Como é possível notar na Figura 10 os olhos estão abertos e a distância vertical dos olhos apesar de ser menor do que a distância horizontal possuem uma diferença entre si que não é muito grande, isto que determina que os olhos estão abertos. Nesse caso a mensagem na interface mostra que o olhar está direcionado para o centro.

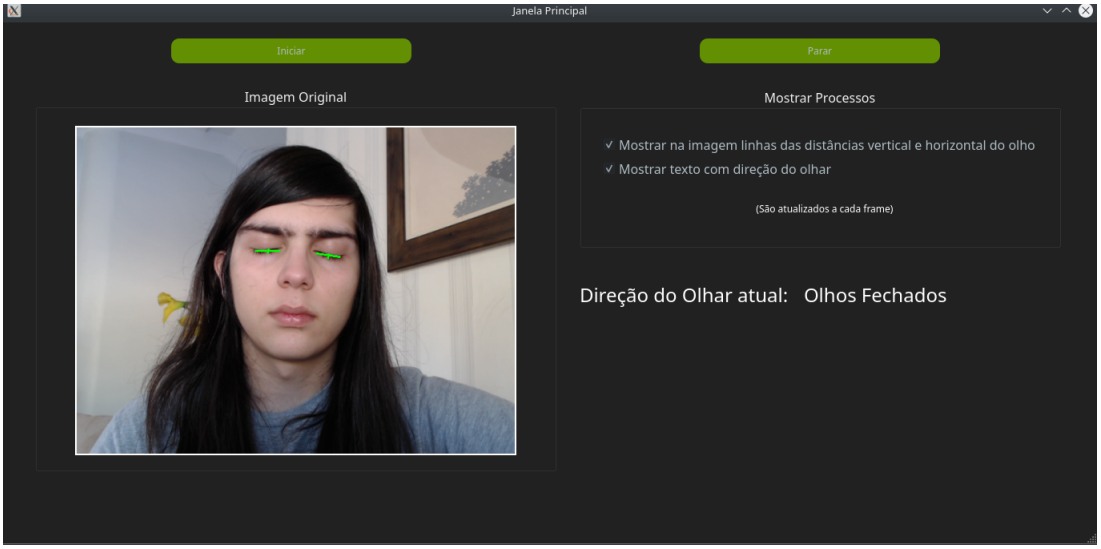
Figura 10 – Visualização das Distâncias Horizontais e Verticais em Olhos Abertos



Fonte: Elaborada pelo autor, mas interface criada a partir do projeto de Stepan Filonov disponível em https://github.com/stepacool/Eye-Tracker

Na Figura 11 os olhos estão fechados, e a diferença entre a distância vertical e horizontal é significativamente maior, o que determina que os olhos estão fechados, e nesse caso a mensagem na interface mostra que os olhos estão fechados.

Figura 11 – Visualização das Distâncias Horizontais e Verticais em Olhos Fechados



Fonte: Elaborada pelo autor, mas interface criada a partir do projeto de Stepan Filonov disponível em https://github.com/stepacool/Eye-Tracker

4.2 Binarização da Imagem

Binarizar uma imagem se trata de criar um limite binário para os valores da imagem. Se o valor é acima de X ele se torna 1, e se está abaixo de X se torna 0.

Isto é aplicado na imagem pois o método utilizado para identificar onde está a pupila se baseia nisso. O segmento de Código 4.4 descreve esse processo, e a Figura 12 mostra a imagem antes e o depois da aplicação da binarização. O trecho de código também "corta" a região dos olhos utilizando os pontos de referência. Isso de modo que apenas o conteúdo de dentro dessa área é utilizado.

Código 4.4 – Aplicação da Binarização da Imagem

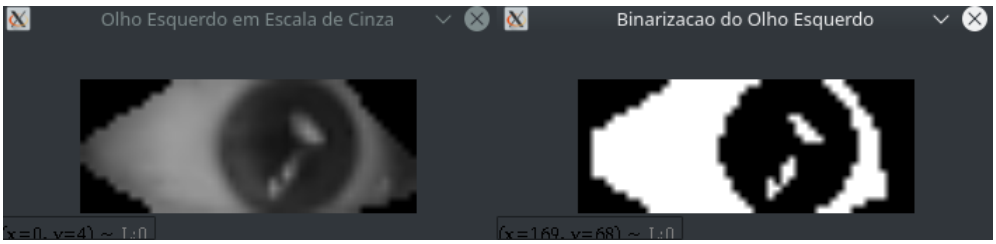
```
# E criada uma mascara na regio esquerda do esquerdo para extrair o que ha
    dentro, e eliminar o que esta de fora
height, width, _ = frame.shape
mask = np.zeros((height, width), np.uint8)
cv2.polylines(mask, [left_eye_region], True, 255, 2)
cv2.fillPoly(mask, [left_eye_region], 255)
left_eye = cv2.bitwise_and(gray, gray, mask=mask)

# E cortado um retangulo que corresponde a area da imagem do olho esquerdo
min_x = np.min(left_eye_region[:, 0])
max_x = np.max(left_eye_region[:, 0])
min_y = np.min(left_eye_region[:, 1])
max_y = np.max(left_eye_region[:, 1])

# Binarizacao aplicada sobre a imagem em escala de cinza do olho esquerdo --
    utiliza valores empiricos
gray_eye = left_eye[min_y: max_y, min_x: max_x]
_, threshold_eye = cv2.threshold(gray_eye, 70, 255, cv2.ADAPTIVE_THRESH_MEAN_C)
```

Fonte: Baseado em CANU (2019)

Figura 12 – Imagem do Olho Esquerdo Isolado e Aplicada Binarização



Fonte: Elaborada pelo autor

4.3 Avaliação da Direção do Olhar

O programa utiliza uma forma simples de identificar a direção onde o usuário está olhando. Ele se baseia em dividir igualmente a imagem de cada olho pela metade em um segmento esquerdo e outro direito. Como ambos já estão com todos os *pixels* com o valor de 0 ou 1 devido o processo de binarização, a região onde existem mais *pixels* com valor 1 são uma boa indicação de que a pupila e a íris estão por ali.

Logo é contada a quantidade de *pixels* que não são 0 em cada metade da imagem e comparado uma razão dos dois. Em seguida é divida a quantidade do lado esquerdo pela quantidade do lado direito. Desta forma quanto maior o valor desta razão, mais indica que a pupila está apontando para a esquerda, e quanto menor é o valor, mais indica que o valor está apontando para a direita.

É efetuada uma verificação do valor da razão obtido com um intervalo de dois valores arbitrários. Se ele estiver abaixo de um determinado do valor do intervalo a pupila está apontando para a direita, se estiver dentro do intervalo então a pupila está apontando para o o centro, e se está acima está apontando para a esquerda. Este processo está descrito no Código 4.5.

Código 4.5 – Identificação da Direção do Olhar

```
1  # Funcao que retorna a razao do quanto a pupila esta para a esquerda ou para a
       direita.
2  # Quanto maior o valor mais para a esquerda a pupila esta direcionada
3  # Quanto menor o valor mais para a direita a pupila esta direcionada
4  def get_gaze_ratio(eye_points, facial_landmarks):
5      # Uma versao similar ao trecho de codigo anterior estaria nesta posicao.
6      # Entratanto o codigo anterior mostrava apenas a aplicaao do threshold e
           recorte do olho esquerdo.
7      # Esta versao suporta o olho esquerdo e o direito.
8      # O olho em si e especificado no momento que se chama o metodo
           get_gaze_ratio.
9
10     # Primeiro a imagem do olho com threshold e divida ao meio
11     # A variavel left_side_threshold armazena o lado esquerdo da imagem do olho
12     left_side_threshold = threshold_eye[0 : height, 0 : int(width / 2)]
13     # E contada a quantidade de "pixels" diferentes de zero na regiao esquerda
           do olho
14     left_side_white = cv2.countNonZero(left_side_threshold)
15
16     # Primeiro a imagem do olho com threshold e divida ao meio
17     # A variavel right_side_threshold armazena o lado esquerdo da imagem do
           olho
```

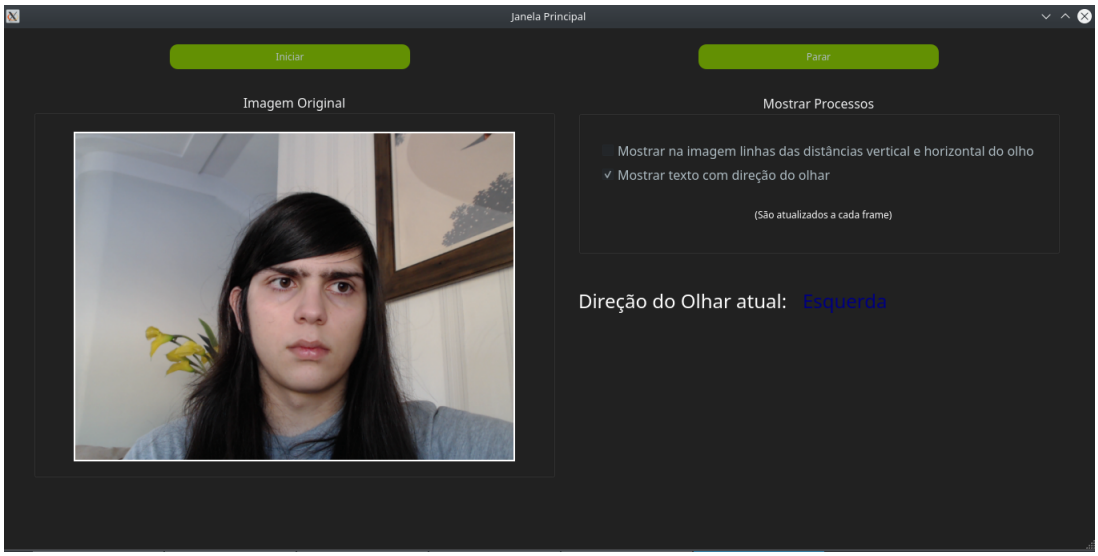
```
18      right_side_threshold = threshold_eye[0 : height, int(width / 2) : width]
19      #    contada a quantidade de "pixels" diferentes de zero na regiao direita
            do olho
20      right_side_white = cv2.countNonZero(right_side_threshold)
21
22      if left_side_white == 0:
23          # Se o lado esquedo estiver completamente branco
24          gaze_ratio = valor_empirico_baixo
25      elif right_side_white == 0:
26          # Se o lado direito estiver completamente branco
27          gaze_ratio = valor_empirico_alto
28      else:
29          # E calculada a razao entre a quantidade de pixels diferentes de zero
                entre regiao esquerda e direita do olho
30          gaze_ratio = left_side_white / right_side_white
31
32      return gaze_ratio
33
34  # Valor da razao da direcao do olhar para o olho esquerdo
35  gaze_ratio_left_eye = get_gaze_ratio([36, 37, 38, 39, 40, 41], landmarks)
36  # Valor da razao da direcao do olhar para o olho esquerdo
37  gaze_ratio_right_eye = get_gaze_ratio([42, 43, 44, 45, 46, 47], landmarks)
38  # Media razao da direcao do olhar de ambos os olhos
39  gaze_ratio = (gaze_ratio_right_eye + gaze_ratio_left_eye) / 2
40
41
42  if gaze_ratio <= valor_empirico_direita:
43      # Esta olhando para a direita
44      estaOlhandoParaDireita()
45  elif valor_empirico_direita < gaze_ratio < valor_empirico_esquerda:
46      # Esta olhando para o centro
47      estaOlhandoParaOCentro()
48  else:
49      # Esta olhando para a esquerda
50      estaOlhandoParaDireita()
```

Fonte: Baseado em CANU (2019)

A Figura 13, Figura 14 e Figura 10 mostram o *software* detectando o olhar do usuário que está direcionado para a esquerda, direita e centro respectivamente. A imagem do rosto existente nas figuras é a própria imagem capturada pela *webcam*. As três figuras possuem a sua própria mensagem na interface, a qual mostra a direção do olhar identificada, e que nas três

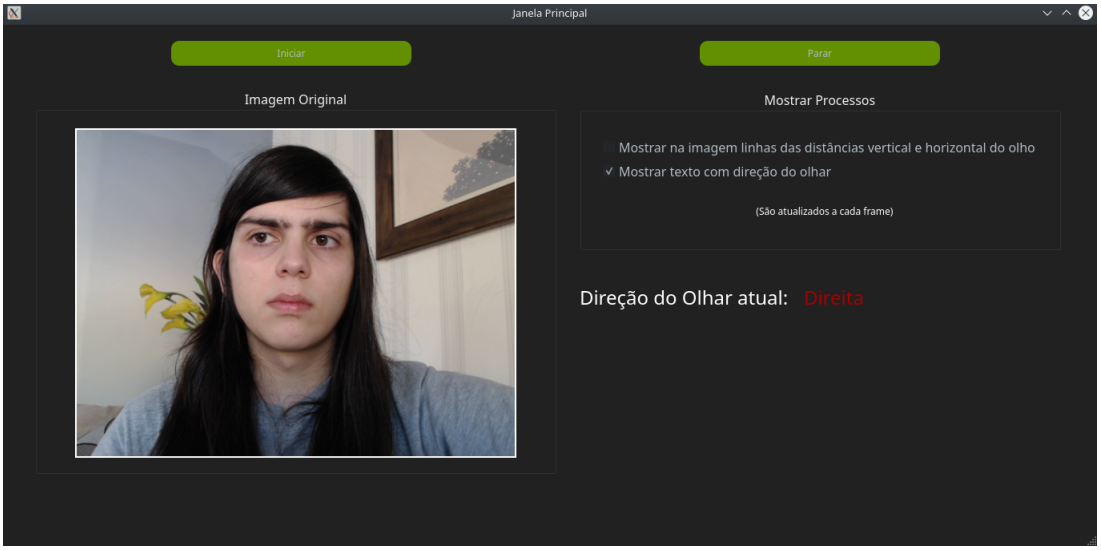
figuras esta informação está de acordo com a imagem da *webcam*. A Figura 10 contém alguns elementos visuais adicionados posteriormente para visualizar as linhas de distância vertical e horizontal dos olhos, de modo que apesar disso essa figura mostra a detecção da direção do olhar quando o mesmo está voltado para o centro.

Figura 13 – *Software* Detectando Direção do Olhar para a Esquerda



Fonte: Elaborada pelo autor, mas interface criada a partir do projeto de Stepan Filonov disponível em https://github.com/stepacool/Eye-Tracker

Figura 14 – *Software* Detectando Direção do Olhar para a Direita



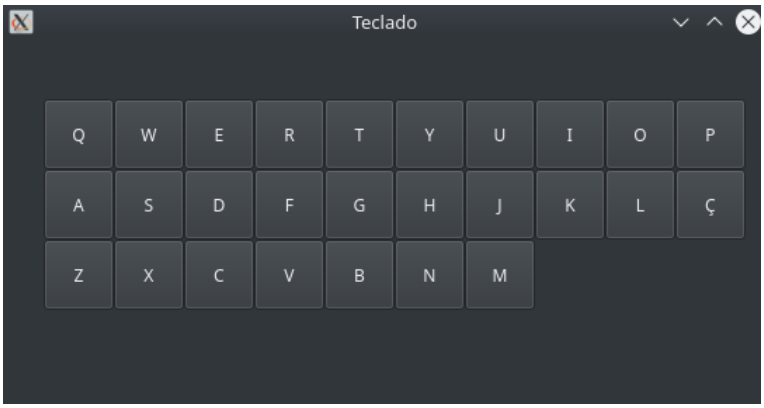
Fonte: Elaborada pelo autor, mas interface criada a partir do projeto de Stepan Filonov disponível em https://github.com/stepacool/Eye-Tracker

4.4 Interface

A interface possui duas partes da implementação. Uma que é utilizada para verificar o rastreamento de olhar, a qual mostra o vídeo da *webcam* e uma mensagem indicando a direção do olhar. Esta interface está visível na Figura 10, Figura 11, Figura 13 e Figura 14. A interface foi construída baseada na interface que Stepan Filonov desenvolveu para seu projeto "*Eye Tracker*", e o código está disponível no *Github* pelo link https://github.com/stepacool/Eye-Tracker.

A outra parte da implementação seria a das interfaces que o usuário utilizaria para interagir apenas com os olhos, e seriam feitas utilizando o *QtDesigner*. De modo que para auxiliar a visualização de como seria uma dessas interfaces, a Figura 15 mostra uma interface com uma aparência similar à de um teclado de computador físico. Esta interface permitiria o usuário digitar utilizando os olhos em algum outro *software*.

Figura 15 – Teclado Virtual Simples Criado no *OpenCv*



Fonte: Elaborada pelo autor

Ao testar o seu uso, percebeu-se alguns problemas. Apesar do *Qt* utilizar arquivos de extensão .ui que se assemelham com *HTML*, a manipulação de seus elementos pelo *Python* não é como a relação entre a manipulação de elementos do *HTML* no *JavaScript*.

4.5 Considerações do Método Usado

Um aspecto interessante da implementação é o de que nos segmentos de rastreamento de olhar do *software* estão presentes alguns dos conceitos descritos no Capítulo 2. Principalmente os relacionados à detecção facial. No código isto pode não se mostrar aparente pois o *OpenCV* possui implementações próprias que são práticas de usar, de modo que tarefas não muito simples como detecção facial podem ser feitas em poucas linhas de código.

Como resultado da maneira que o rastreamento de olhar é feito, o ideal é que o rosto esteja na direção da *webcam*, e que nenhuma parte do rosto esteja muito obstruída. Isto pois

o programa visa identificar o que é um rosto antes de procurar pelos olhos. E se o rosto não é detectado, o *software* é incapaz de localizar a posição dos olhos.

O método de reconhecimento da direção do olhar utilizado, que procura identificar qual metade do olho possui *pixels* mais escuros, é uma implementação simples de um método de rastreamento de olhar e possui suas propriedades, que possuem alguns aspectos positivos e outros negativos.

Como esta forma de rastreamento de olhar não utiliza nenhum processo de calibração, são utilizados alguns valores obtidos empiricamente para determinar algumas situações, como por exemplo o quão para a esquerda a íris e pupila precisam estar para se considerar que se está olhando para a esquerda. Nas condições de teste efetuadas eles se mostraram funcionais, entretanto isto pode não ser o ideal para um *software* que será usado por várias pessoas.

5 Conclusão

A proposta do trabalho era de implementar um *software* de rastreio ocular capaz de simular uma entrada de dados baseada no posicionamento da pupila. O sistema desenvolvido foi capaz de satisfatoriamente cumprir os seus requisitos, de modo que o *software* mostra como é possível fazer o rastreamento de olhar sem equipamentos intrusivos utilizando dispositivos de fácil acesso como *webcams*.

O *software* entretanto apresenta algumas limitações. O mesmo não possui um suporte implementado para interagir com outras interfaces utilizando os olhos. Assim como o *software* possui um numero reduzido de graus de liberdade para entrada de dados, de modo que se baseia em identificar estados do olhar, como se o olho está fechado, olhando para a esquerda, olhando para a direita, ou olhando para o centro. Este tipo de método não é muito apropriado para controlar interfaces como um cursor, ou simular um ponteiro de *mouse* de computador. Entretanto uma implementação como a deste *software* desenvolvido ainda poderia ser usada para navegar por interfaces selecionando os ícones de forma linear, de modo que para se chegar em um elemento da interface, seria necessário navegar pelos demais até chegar no elemento desejado.

Outro aspecto do *software* é de que a identificação da direção do olhar pode ser usada para funções não direcionais. Por exemplo em um jogo estilo "*endless runner*", onde olhar para a esquerda poderia estar atribuído ao comando do personagem controlado pelo jogador se abaixar, e olhar para a direita para o personagem saltar.

A implementação do *software* desenvolvido, em conjunto aos conceitos estudados no trabalho, permitem que sejam feitas melhorias no *software* e que sejam feitas também outras abordagens para efetuar o rastreamento ocular.

5.1 Trabalhos Futuros

Existem várias possibilidades para trabalhos futuros envolvendo a área de rastreamento de olhar. Fazer a implementação do uso dos dados obtidos pelo rastreamento de olhar com uma interface seria um passo muito importante. Assim como poderiam ser utilizados outros métodos para efetuar rastreamento de olhar, uma vez que existem abordagens mais precisas e que também consideram a orientação da cabeça do usuário. Alguns métodos utilizam recursos diferentes para atingir ótimos resultados, como o rastreamento de olhar usando aprendizado profundo.

Uma melhoria que poderia ser feita é a de permitir o rastreamento de olhar também no eixo vertical, identificando assim se o olhar do usuário está para para cima ou para baixo.

Um processo de calibração também seria interessante para melhorar a precisão do rastreio de olhar, e remover o uso dos valores definidos empiricamente.

Existem alguns recursos bem úteis para o desenvolvimento de aplicações de rastreamento de olhar disponíveis para uso, como por exemplo a biblioteca *Gaze* que fornece algumas ferramentas que facilitam no desenvolvimento de aplicações do gênero (HöFFNER, 2018).

Referências

BAGGIO, D. L. *OpenCV 3.0 Computer Vision with Java*. [S.l.]: Packt Publishin, 2015. Disponível em: <https://www.packtpub.com/application-development/opencv-30-computer-vision-java>. Acessado em: 16-06-2020.

BARRETO, A. M. *Eye Tracking Como Método de Investigação Aplicado às Ciências da Comunicação*. 2012. Disponível em: <http://revistacomunicando.sopcom.pt/ficheiros/20130108-tracking.pdf>. Acessado em: 07-03-2020.

CANU, S. *Eye motion tracking – Opencv with Python*. 2019. Disponível em: <https://pysource.com/2019/01/07/eye-detection-gaze-controlled-keyboard-with-python-and-opencv-p-1/>. Acessado em: 10-06-2020.

COMPUTERPHILE. *Detecting Faces (Viola Jones Algorithm) - Computerphile*. 2018. Disponível em: <https://www.youtube.com/watch?v=uEJ71VlUmMQ>. Acessado em: 13-06-2020.

CULJAK, I.; ABRAM, D.; PRIBANIC, T.; DZAPO, H.; CIFREK, M. *A brief introduction to OpenCV*. 2012. Disponível em: <https://ieeexplore.ieee.org/abstract/document/6240859>. Acessado em: 08-03-2020.

FERREIRA, A. O.; FERREIRA, S. B. L.; SILVEIRA, D. S. da; FERREIRA, A. F. *EVALUATION OF WEB ACESSIBILITY: A STUDY AIMED FOR USERS WITH CEREBRAL PALSY*. 2011. Disponível em: <http://www.iadisportal.org/digital-library/evaluation-of-web-acessibility-a-study-aimed-for-users-with-cerebral-palsy>. Acessado em: 07-03-2020.

HöFFNER, S. *Gaze Tracking Using Common Webcams*. 2018. Disponível em: <https://www.researchgate.net/publication/326588105_Gaze_Tracking_Using_Common_Webcams>. Acessado em: 10-06-2020.

KING, D. E. Dlib-ml: A machine learning toolkit. *Journal of Machine Learning Research*, v. 10, p. 1755–1758, 2009. Acessado em: 20-06-2020.

LOGITECH. *C920 HD Pro Webcam*. 2020. Disponível em: <https://www.logitech.com/pt-br/product/hd-pro-webcam-c920>. Acessado em: 16-06-2020.

LORENA, A. C.; CARVALHO, A. C. de. *Introdução aos Classificadores de Margens Largas*. 2003. Disponível em: <https://web.icmc.usp.br/SCATUSU/RT/BIBLIOTECA_113_RT_195.pdf>. Acessado em: 13-06-2020.

MORDVINTSEV, A.; REVISION, A. K. *Face Detection using Haar Cascades*. 2013. Disponível em: <https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_objdetect/py_face_detection/py_face_detection.html>. Acessado em: 16-06-2020.

MORIMOTO, C. H.; MIMICA, M. R. *Eye gaze tracking techniques for interactive applications*. 2004. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1077314204001109>. Acessado em: 07-03-2020.

PRATES, R. O.; BARBOSA, S. D. J. *Introdução à Teoria e Prática da Interação Humano Computador fundamentada na Engenharia Semiótica*. [S.l.]: PUC-Rio, 2007. Disponível em: <http://www3.serg.inf.puc-rio.br/docs/JAI2007_PratesBarbosa_EngSem.pdf>. Acessado em: 07-03-2020.

PUTRA, I. K. G. D.; CAHYAWAN, A.; PERDANA, Y. *Low-Cost Based Eye Tracking and Eye Gaze Estimation*. 2011. Disponível em: <https://www.researchgate.net/publication/268290417_Low-Cost_Based_Eye_Tracking_and_Eye_Gaze_Estimation>. Acessado em: 07-03-2020.

PYTHON-SOFTWARE-FOUNDATION. *https://docs.python.org/3/tutorial/index.html*. 2020. Disponível em: <https://docs.python.org/3/tutorial/index.html>. Acessado em: 13-06-2020.

RIVERBANK-COMPUTING. *What is PyQt?* 2020. Disponível em: <https://riverbankcomputing.com/software/pyqt/intro>. Acessado em: 20-06-2020.

TIMM, F.; BARTH, E. *Accurate Eye Centre Localisation by Means of Gradients*. 2011. Disponível em: <https://www.researchgate.net/publication/221415814_Accurate_Eye_Centre_Localisation_by_Means_of_Gradients>. Acessado em: 10-06-2020.

VIOLA, P.; JONES, M. J. *Rapid Object Detection using a Boosted Cascade of Simple Features*. 2001. Disponível em: <https://www.researchgate.net/publication/3940582_Rapid_Object_Detection_using_a_Boosted_Cascade_of_Simple_Features>. Acessado em: 13-06-2020.