

GUSTAVO CARVALHO SANTOS

**DESENVOLVIMENTO DE UMA INTERFACE
GRÁFICA PARA DISPOSITIVO EMBARCADO
DE PUPILOMETRIA**

UBERLÂNDIA

2017

GUSTAVO CARVALHO SANTOS

**DESENVOLVIMENTO DE UMA INTERFACE GRÁFICA
PARA DISPOSITIVO EMBARCADO DE
PUPILOMETRIA**

Trabalho apresentado na Universidade Federal de Uberlândia como requisito para conclusão do curso de Engenharia Eletrônica e de Telecomunicações.

Orientador: Prof. Dr. Antônio Cláudio Paschoarelli Veiga

UBERLÂNDIA

2017

GUSTAVO CARVALHO SANTOS

**DESENVOLVIMENTO DE UMA INTERFACE GRÁFICA
PARA DISPOSITIVO EMBARCADO DE
PUPILOMETRIA**

Trabalho apresentado na Universidade Federal de Uberlândia como requisito para conclusão do curso de Engenharia Eletrônica e de Telecomunicações.

Trabalho aprovado. **UBERLÂNDIA**, 20 de Dezembro de 2017:

**Prof. Dr. Antônio Cláudio
Paschoarelli Veiga**
Orientador

Prof. Dr. Gilberto Arantes Carrijo
Convidado 1

**Prof.^a Dr.^a Milena Bueno Pereira
Carneiro**
Convidado 2

**UBERLÂNDIA
2017**

*Este trabalho é dedicado a minha família,
que sempre me incentivou a acreditar em meus sonhos.*

Agradecimentos

Aos meus pais, Edilson e Vivian e meu irmão Guilherme, pelo apoio e amor.

Ao meu orientador Prof. Dr. Paschoarelli, pelo auxílio, atenção e confiança.

Ao meu coorientador Rafael A. da Silva, pelo apoio e paciência.

Aos meus amigos e familiares que sempre estiveram comigo.

*“A verdade pode ser intrigante. Pode dar algum trabalho lidar com ela.
Pode ser contra-intuitiva. Ela pode contradizer preconceitos profundamente enraizados.
Pode não se coadunar com o que queremos desesperadamente que seja verdade.
Mas nossas preferências não determinam o que é verdade.
(Carl Sagan)*

Resumo

A pupila do olho humano é hábil a realizar movimentos de acordo com as circunstâncias em que ela é exposta. Elementos como luminosidade, uso de substâncias químicas e patologias podem causar mudanças na dinâmica da pupila. Muitos softwares e dispositivos já foram criados para a realização da pupilometria (processo em que se aplicam estímulos de luz a fim de medir a resposta pupilar), mas são escassos os softwares que possuem interface gráfica e proporcionam ao usuário a configuração de parâmetros do processo. Dessa forma, objetiva-se neste trabalho criar uma interface que facilite o processo de pupilometria para o usuário, tornando mais dinâmico o processo.

Palavras-chave: Pupila, Softwares, Pupilometria, Resposta pupilar, Interface gráfica.

Abstract

The pupil of the human eye is able to perform movements according to the circumstances in which it is exposed. Elements such as luminosity, use of chemical substances and pathologies can cause changes in pupil dynamics. Many software and devices have already been created to perform the pupilometry (a process in which light stimuli are applied in order to measure the pupillary response), but there are few softwares that have a graphical interface and provide the user with the configuration of process parameters. In this way, the objective of this work is to create an interface that facilitates the pupilometry process for the user, making the process more dynamic.

Keywords: Pupil, Softwares, Pupilometry, Pupillary response, Graphical interface.

Lista de ilustrações

Figura 1 – Etapas de um sistema de processamento digital de imagens.	17
Figura 2 – Máscara de dimensão 3 x 3 e seus coeficientes.	19
Figura 3 – Processo de convolução da máscara pela imagem no processo de filtragem.	19
Figura 4 – Detecção de bordas através do operador de Canny.	20
Figura 5 – Exemplo do espaço de Hough correspondente à imagem de bordas de um olho humano.	23
Figura 6 – Processo de votação efetuado pela Transformada Circular de Hough.	23
Figura 7 – Raspberry Pi 2®	25
Figura 8 – Câmera Pi NoIR	26
Figura 9 – Exemplo de Periféricos conectados a Raspberry Pi	27
Figura 10 – Portas da Raspberry Pi 2®	27
Figura 11 – Circuito de controle dos LEDs.	28
Figura 12 – Visor contendo câmera e iluminação LED infravermelho.	29
Figura 13 – Protótipo do dispositivo de captura.	30
Figura 14 – Diagrama do algoritmo de rastreamento pupilar.	31
Figura 15 – Captura de imagens da câmera durante execução do software no dispositivo Raspberry Pi.	32
Figura 16 – Diferentes filtros de suavização.	33
Figura 17 – Detecção da circunferência pupilar através da transformada de Hough.	33
Figura 18 – Software sendo executado através do terminal de texto da Raspberry Pi.	34
Figura 19 – Opções de parâmetros criadas para captura.	36
Figura 20 – Janela para salvar arquivos de texto	37
Figura 21 – Janela para salvar vídeos	37
Figura 22 – Barra de progresso antes de iniciar o processo de captura.	38
Figura 23 – Botão de iniciar e Barra de progresso.	38
Figura 24 – Visão geral da interface de captura	39
Figura 25 – Parâmetros de Análise.	40
Figura 26 – Selecionar vídeo e salvar arquivo de texto.	40
Figura 27 – Salvando o arquivo de texto gerado na Análise.	41
Figura 28 – Selecionando o arquivo de vídeo gerado na Captura.	42
Figura 29 – Visão geral da interface de Análise.	43
Figura 30 – Tela Sensível ao Toque de 3,5 polegadas.	44
Figura 31 – Layout da Interface de Captura pra tela sensível ao toque.	45
Figura 32 – Execução da interface de Captura Touchscreen.	46
Figura 33 – Texto de auxílio ao usuário na Captura Touchscreen.	46

Figura 34 – Formato dos nomes dos arquivos gerados no processo de captura em touchscreen.	47
Figura 35 – Layout da Interface de Análise pra tela sensível ao toque.	47
Figura 36 – Execução da interface de Análise Touchscreen.	48
Figura 37 – Texto de auxílio ao usuário na Análise Touchscreen.	49
Figura 38 – Formato dos nomes dos arquivos gerados no processo de análise em touchscreen.	49

Lista de abreviaturas e siglas

DSP	<i>Digital Signal Processor.</i>
FPGA	<i>Field-programmable gate array.</i>
GND	<i>Ground</i>
GPIO	<i>General-purpose input/output.</i>
HDMI	<i>High-Definition Multimedia Interface.</i>
LED	<i>Light-emitting diode.</i>
OpenCV	<i>Open Source Computer Vision Library.</i>
PLR	<i>Pupillary light reflex.</i>
PWM	<i>Pulse-Width Modulation.</i>
SoC	<i>System on a Chip.</i>
USB	<i>Universal Serial Bus.</i>

Lista de símbolos

® *Marca Registrada.*

mA *Miliampere.*

V *Volts.*

Sumário

1	INTRODUÇÃO	14
1.1	Motivação	14
1.2	Objetivo	15
1.3	Estrutura deste trabalho	15
2	PROCESSAMENTO DIGITAL DE IMAGENS	17
2.1	Etapas de um Sistema de Processamento Digital de imagens	17
2.1.1	Aquisição	18
2.1.2	Suavização	18
2.1.3	Detecção de Bordas	20
2.1.4	Segmentação	21
2.1.5	Reconhecimento	21
2.2	Transformada de Hough	22
2.3	Visão Computacional	24
2.3.1	OpenCV	24
3	DESCRIÇÃO DO SISTEMA	25
3.1	Raspberry Pi	25
3.2	Câmera Pi NoIR	26
3.3	Periféricos: Teclado, Monitor, Mouse	26
3.4	Alimentação	27
3.5	Circuito do LED	28
3.6	Visores	29
4	ALGORITMO	31
4.1	Funcionamento	31
4.1.1	Aquisição da Imagem	32
4.1.2	Pré-Processamento e Reconhecimento	32
4.2	Execução	34
5	DESENVOLVIMENTO DA INTERFACE GRÁFICA PARA DESKTOP	35
5.1	Software QT Creator	35
5.2	Captura	35
5.2.1	Parâmetros de captura	35
5.2.2	Salvando os arquivos gerados na captura	36
5.2.3	Barra de progresso	38

5.2.4	Visão geral da Interface de Captura	38
5.3	Análise	39
5.3.1	Parâmetros de Análise	39
5.3.2	Arquivos	40
5.3.3	Visão geral da Interface de Análise	42
6	DESENVOLVIMENTO DA INTERFACE PARA TELA SENSÍVEL AO TOQUE	44
6.1	Mudanças no Sistema	44
6.2	Captura	45
6.2.1	Arquivos	46
6.3	Análise	47
6.3.1	Arquivos	48
6.4	Comparações entre Sistemas	49
7	CONCLUSÕES E TRABALHOS FUTUROS	51
7.1	Principais Contribuições	51
7.2	Trabalhos Futuros	51
	REFERÊNCIAS	52
	APÊNDICES	55
	APÊNDICE A – CÓDIGO FONTE DAS INTERFACES GRÁFICAS	56
A.1	Código: Interface de Captura para Desktop	56
A.2	Código: Interface de Análise para Desktop	60
A.3	Código: Interface de Captura para Tela sensível ao toque	66
A.4	Código: Interface de Análise para Tela sensível ao toque	70

1 Introdução

1.1 Motivação

A pupila humana efetua movimentos de acordo com as condições às quais está sujeita. Esses movimentos podem ser contrações (mioses) ou dilatações (midríases) estimuladas pelo sistema nervoso parassimpático e simpático respectivamente. Fatores como luminosidade, uso de substâncias químicas e patologias podem causar mudanças na dinâmica pupilar (FERRARI, 2008).

Em virtude de interesses clínicos na determinação de características pupilares, a pupilometria dinâmica, que consiste na análise dos reflexos pupilares ao estímulo luminoso, tem sido abordada cientificamente e sistemas automáticos desenvolvidos de modo a realizá-la em tempo real.

Os movimentos da pupila consistem basicamente em acomodação, reflexo pupilar à luz (PLR) e hippus. A pupilometria analisa os movimentos pupilares através da mensuração de diversos componentes: Amplitude máxima (diferença entre tamanho inicial e mínimo durante o PLR), latência, velocidade de contração e dilatação, tamanhos máximo e mínimo da pupila.

O teste do reflexo pupilar à luz (PLR) já foi empregado em trabalhos passados na investigação de condições como alcoolismo (TAN et al., 1984), uso de drogas (GRÜNBERGER et al., 1990), (ROSSE et al., 1995), síndrome de Down (SACKS; SMITH, 1989), depressão (FOTIOU et al., 2000), (SOKOLSKI; DEMET, 1996), mal de Alzheimer (FOTIOU et al., 2000), (GRANHOLM et al., 2003), mal de Parkinson (GRANHOLM et al., 2003), (RIZOS et al., 2004), insuficiência cardíaca (KEIVANIDOU et al., 2010), déficit de atenção, diabetes (EUSTACE S. MURNAGHAN, 1981), AIDS (MACLEAN; DHILLON, 1993), autismo (FAN et al., 2009), entre outros.

Devido a não existência de dispositivos de baixo custo e código aberto que realizam a pupilometria, foi desenvolvido um pupilômetro utilizando o sistema embarcado Raspberry Pi 2 (SILVA, 2016). O dispositivo mencionado desempenha a etapa de captura de ciclos pupilares utilizando controle luminoso ativo, e também é capaz de efetuar o processamento da sequência de vídeo, retornando a resposta pupilar aferida. A junção de todas as etapas do sistema em um só dispositivo embarcado proporciona vantagens como a portabilidade e versatilidade, além do baixo custo de desenvolvimento decorrente do uso de ferramentas abertas - software e hardware.

Na ferramenta citada, parâmetros como raio mínimo, raio máximo, duração dos estímulos luminosos são predeterminados no algoritmo, impossibilitando seu ajuste por

usuários que não possuam familiaridade com o código fonte. Além disso, toda a execução do programa é feita sem interface gráfica, apenas usando o terminal de texto do Raspberry Pi®.

1.2 Objetivo

A fim de facilitar o uso do dispositivo de pupilometria para todos os públicos, objetiva-se neste trabalho desenvolver uma interface gráfica que possibilite a configuração de parâmetros da pupilometria de forma interativa e simples. Por conseguinte o processo de pupilometria se tornará mais ágil e versátil.

Para alcançar o objetivo citado, alguns objetivos específicos devem ser atingidos. São eles:

- Conhecer e utilizar o software QT Creator;
- Desenvolver uma interface gráfica de Captura de vídeo para pupilometria;
- Desenvolver uma interface gráfica de Análise de vídeo para pupilometria;
- Testar o funcionamento e identificar futuros aperfeiçoamentos para a interface gráfica;

1.3 Estrutura deste trabalho

A estrutura deste trabalho foi dividida em sete capítulos, com os respectivos títulos:

1. Introdução;
2. Processamento Digital de Imagens
3. Descrição do Sistema;
4. Algoritmo;
5. Desenvolvimento da Interface Gráfica para Desktop;
6. Desenvolvimento da Interface Gráfica para Tela sensível ao toque;
7. Conclusões e Trabalhos Futuros;

Apêndice A - Código Fonte das Interfaces Gráficas;

A Introdução tem como propósito delimitar e apresentar os objetos de estudo deste trabalho, bem como justificar e dar relevância para as questões pesquisadas. Além disso, ela ambienta o leitor à estrutura que observará nos capítulos seguintes.

O Capítulo 2, Processamento Digital de Imagens, tem por objetivo abordar os elementos básicos de um sistema de processamento de imagens, destacando os fundamentos da suavização de imagens, detecção de bordas, Transformada de Hough e visão computacional.

O Capítulo 3, Descrição do Sistema, tem por objetivo descrever o dispositivo de pupilometria, apresentando o dispositivo embarcado escolhido, os periféricos utilizados e os visores desenvolvidos.

No Capítulo 4, é apresentado o algoritmo - ainda sem interface gráfica- do dispositivo de pupilometria desenvolvido, elucidando seu funcionamento etapa por etapa e sua execução, ilustrando ao leitor a necessidade do desenvolvimento de uma interface.

O Capítulo 5, Desenvolvimento da Interface Gráfica para Desktop, tem por objetivo descrever as duas interfaces desenvolvidas para Desktop. A primeira interface realiza o processo de captura do fluxo de vídeo da pupila, a segunda tem por objetivo fazer a análise dos dados e retornar os valores aferidos.

O Capítulo 6, Desenvolvimento da Interface Gráfica para Tela sensível ao toque, tem por propósito relatar as duas interfaces com as mesmas funcionalidades, desenvolvidas com o intuito de obter maior mobilidade ao sistema, dispensando o uso de periféricos.

No Capítulo 7 estão presentes as conclusões, com as contribuições deste trabalho e sugestões de trabalhos futuros com o tema pupilometria.

Os apêndices contêm os códigos fonte das interfaces desenvolvidas para captura e análise do fluxo de vídeo para Desktop e Tela sensível ao toque.

2 Processamento Digital de Imagens

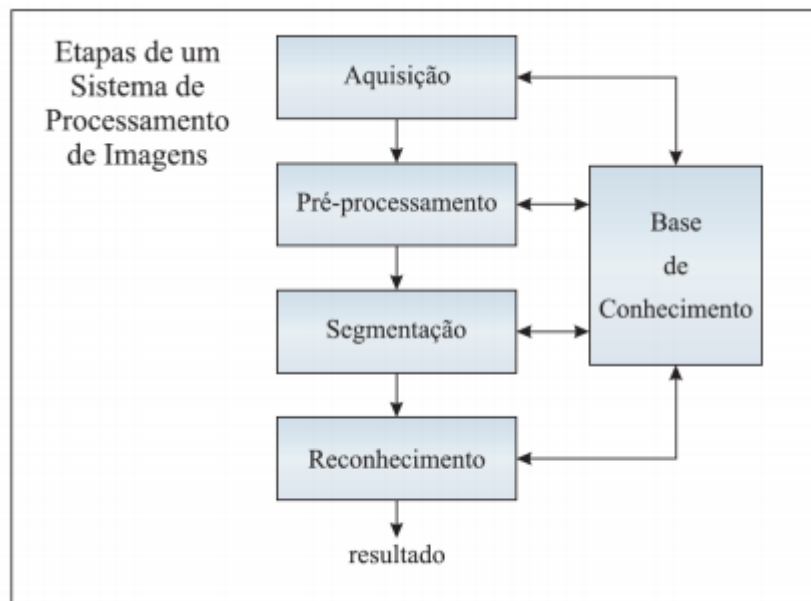
2.1 Etapas de um Sistema de Processamento Digital de imagens

Um sistema de processamento digital de imagens manipula imagens obtidas a partir de uma câmera eletrônica, de forma comparável a como o cérebro processa as imagens adquiridas pelos olhos. Pode-se analisar qualquer forma de dados – não só imagens geradas por câmeras, mas também de sensores em qualquer faixa do espectro eletromagnético (SILVA, 2016).

A informação gerada pelas câmeras ou sensores está na forma digital e é normalmente multidimensional. A captura de imagens à medida que varia o tempo produz os dados de vídeo, também importantes em inúmeras aplicações na ciência (BOVIK, 2009).

A obtenção de informações úteis a partir de imagens digitais é possível através de uma sequência de passos, ilustrada na Figura 1. As etapas de um sistema padrão de processamento de imagens consistem em aquisição, pré-processamento, segmentação e reconhecimento. São relacionados a uma base de conhecimento que permite a obtenção de informações importantes a partir da representação numérica de uma imagem (DIAS, 2014).

Figura 1 – Etapas de um sistema de processamento digital de imagens.



Fonte: (DIAS, 2014)

As próximas subseções descrevem as etapas de processamento usadas no dispositivo embarcado de pupilometria (SILVA, 2016), abordando as operações de aquisição, suavização, detecção de bordas, segmentação e reconhecimento.

2.1.1 Aquisição

Antes do processamento de imagem ou vídeo começar, é necessário que a imagem seja capturada por uma câmera e convertida em uma matriz de valores numéricos. Esse é o processo conhecido por aquisição de uma imagem (MOESLUND, 2012).

A aquisição de imagens pode ser efetuada de acordo com a aplicação desejada. Características como espectro, resolução, tempo de exposição entre outras afetam diretamente os resultados do processo posterior de análise de uma imagem (DIAS, 2014).

É importante salientar que a etapa de aquisição de imagens digitais abrange os processos de amostragem e quantização, convertendo-se na obtenção de matrizes contendo as informações numéricas de nível de intensidade para cada pixel.

2.1.2 Suavização

A suavização remete à fase de pré-processamento de imagens e consiste na retirada de ruídos de alta frequência espacial em imagens digitais, o que corresponde à aquisição de uma imagem menos nítida, ou suavizada.

Os filtros de suavização são também chamados de filtros passa baixa, pelo fato de reduzirem as variações nos níveis de cinza ao atenuar as altas frequências, que equivalem aos detalhes da imagem.

As técnicas de filtragem usadas no protótipo desenvolvido (SILVA; VEIGA; MOREIRA, 2017) atuam no domínio espacial, ou seja, atuam diretamente sobre os pixels da imagem, através de operações de convolução com matrizes chamadas máscaras. As máscaras definem os filtros espaciais e o seu uso é denominado filtragem espacial.

O tratamento executado pelas máscaras geralmente corresponde a uma média ponderada dos pixels da vizinhança, onde a cada pixel vizinho é atribuído um fator multiplicador de sua intensidade. As matrizes de convolução (máscaras) são também chamadas de kernel (LAGANIÈRE, 2011). Uma máscara genérica de tamanho 3x3 é exibida na Figura 2.

Figura 2 – Máscara de dimensão 3 x 3 e seus coeficientes.

$$\begin{vmatrix} W_0 & W_1 & W_2 \\ W_3 & W_4 & W_5 \\ W_6 & W_7 & W_8 \end{vmatrix}$$

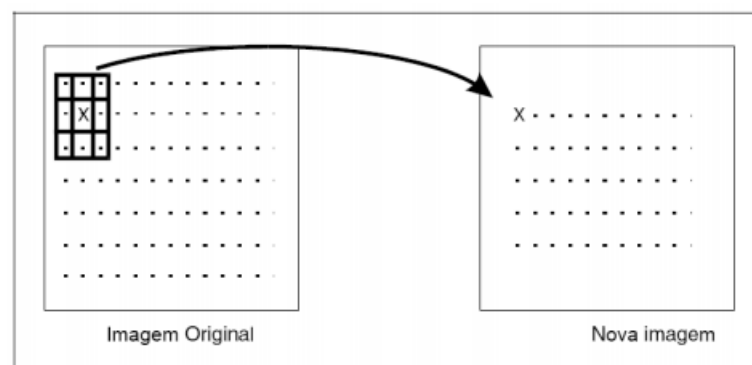
Fonte: (SILVA, 2016)

O tamanho da matriz é escolhido arbitrariamente e é necessário para se descrever uma máscara. Um filtro 3 x 3, por exemplo, representa uma máscara com 3 pixels de largura e 3 pixels de altura.

O novo valor de intensidade de um pixel após o processo de filtragem é calculado inserindo-se a máscara centrada nele. Os pixels das imediações são multiplicados pelos coeficientes de peso correspondentes e adicionados a uma soma total. Este valor pode ou não ser dividido por um fator e se torna o novo valor do pixel, na imagem de saída. Esse processo é refeito para todos os pixels da imagem a ser processada e equivale ao processo de convolução da matriz pela imagem a ser transformada (NIXON; AGUADO, 2008).

O Processo de convolução de uma máscara de dimensão 3 x 3 por uma imagem é exibido na Figura 3. Observa-se a substituição do pixel central pelo novo valor calculado por meio da máscara e a remoção dos pixels da borda na imagem original. Ao avançar o fim de cada linha a máscara passa, então, à próxima linha, onde o processo continua até que todos os pixels da imagem sejam operados.

Figura 3 – Processo de convolução da máscara pela imagem no processo de filtragem.



Fonte: (SILVA, 2016)

2.1.3 Detecção de Bordas

As bordas em uma imagem identificam os contornos nela presentes, retratando importância fundamental na segmentação e detecção de objetos e padrões. Os pontos que representam a bordas em uma imagem possuem variações bruscas de níveis de cinza. Eles representam pontos de transição entre diferentes objetos (QUEIROZ; GOMES, 2006).

Os métodos mais conhecidos para detecção de borda são os operadores de Robert, Sobel, Prewitt e Canny (HOWE, 2014). Depois da passagem dos operadores, efetua-se a etapa de limiarização que resulta em uma imagem que contém somente as bordas identificadas pelo método utilizado.

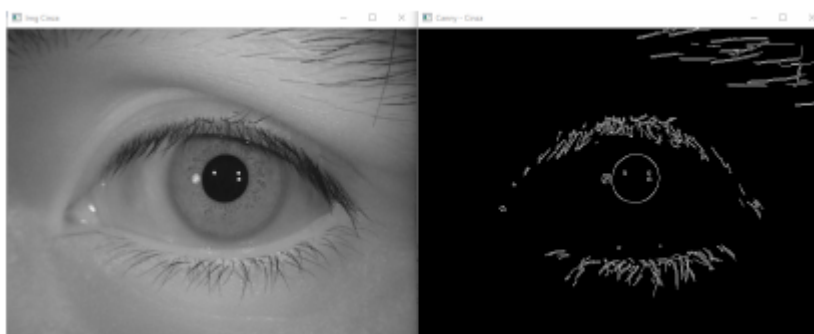
O detector de bordas amplamente utilizado em aplicações similares e escolhido no algoritmo desenvolvido (SILVA; VEIGA; MOREIRA, 2017) é o operador de Canny (CANNY, 1986). Este detector exibe boa imunidade ao ruído ao mesmo tempo em que reconhece verdadeiros pontos de borda com erro mínimo (NIXON; AGUADO, 2008). O operador de Canny é baseado em três otimizações na detecção de bordas (HOWE, 2014):

- Maximização da relação sinal ruído do gradiente;
- Um fator de localização de borda, que garante que a borda seja detectada o mais precisamente possível;
- Minimização de respostas múltiplas a uma mesma borda;

Isso fornece os prós desse detector em relação a outros existentes. Ele exige que a imagem seja anteriormente suavizada, antes que a direção e valores do gradiente sejam calculados.

A Figura 4 demonstra a identificação de bordas em uma imagem, sem a utilização prévia de um filtro de suavização.

Figura 4 – Detecção de bordas através do operador de Canny.



Fonte: (SILVA, 2016)

2.1.4 Segmentação

Procedimentos de análise de imagens exigem a extração de características, medidas e informações de forma automática e semi-automática. Dados significativos geralmente estão localizados em uma parte específica da imagem que é, dessa forma, subdivida de forma a aprimorar a região de análise em partes ou objetos específicos (QUEIROZ; GOMES, 2006).

Os métodos de segmentação proporcionam o reconhecimento de dissimilaridades entre regiões ou objetos de uma imagem, de forma a executar a discriminação dos mesmos entre si e em relação ao fundo da imagem (background).

Geralmente os algoritmos de segmentação de imagens monocromáticas se baseiam na identificação de descontinuidades e similaridades dos níveis de cinza. O reconhecimento de bordas é elemento usualmente presente nesses métodos. Pode-se usar ainda outros artifícios como o adotado depois neste trabalho, que pode utilizar dados de gradiente na manipulação de imagens sequenciais de forma a permitir a definição da região de interesse a partir das mudanças nas imagens no tempo.

No trabalho desenvolvido (SILVA, 2016), é primordial que se localize a região ao redor da íris em uma imagem e se efetue o processamento em seu entorno. A delimitação da região de interesse é determinante na efetividade do algoritmo.

2.1.5 Reconhecimento

Uma vez segmentada, a imagem passa à seguinte e última fase de processamento: o reconhecimento de objetos ou regiões na cena. O reconhecimento de padrões é elemento chave na visão computadorizada e processamento de imagem (HOWE, 2014). O reconhecimento está exposto em aplicações que vão da medicina diagnóstica e biometria à classificação de documentos, sensoriamento remoto, entre outros.

Nesta fase, pode-se perceber a indispensabilidade dos tratamentos anteriormente realizados na imagem, especialmente o processo de segmentação. Se o objeto a ser detectado não está corretamente introduzido em uma região de interesse efetiva, não se alcançara êxito no reconhecimento ou haverá prejuízos no funcionamento do algoritmo (DIAS, 2014).

Algumas propriedades de um objeto multidimensional que definem padrões são a área, volume, perímetro, superfície, entre outros. Essas podem ser mensuradas através da contagem de pixel. Do mesmo modo, a forma de um objeto pode ser caracterizada a partir de suas bordas. A apuração e remoção dos parâmetros corretos representa o problema principal no reconhecimento de padrões.

O objeto de estudo nas imagens no presente trabalho é a pupila. Propõe-se localizá-la a partir do reconhecimento de forma, aproximando o contorno pupilar através de uma

circunferência. As ferramentas mais usadas aptas a essa função são a Transformada circular de Hough e operador Integro-diferencial de Daugman, além das ferramentas de template matching (DIAS, 2014).

Resultados superiores em tempo de processamento sem implicação da precisão são comprovadamente alcançados através da transformada de Hough (DIAS, 2014). Dessa forma, essa técnica é usada no algoritmo desenvolvido (SILVA, 2016).

2.2 Transformada de Hough

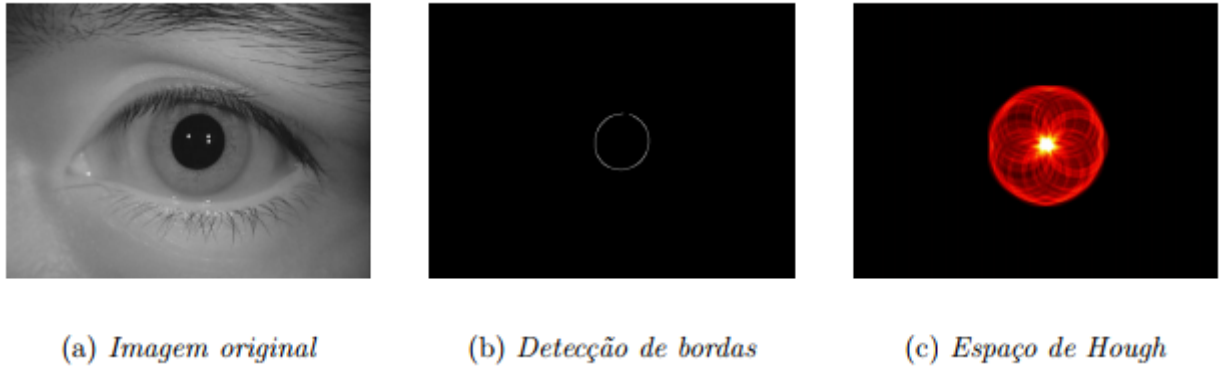
A transformada de Hough é empregada para identificar formas geométricas como linhas, círculos e elipses em imagens digitais. É possivelmente um dos métodos mais utilizados na visão computacional (HART; STORK; WILEY, 1973).

A Transformada Circular de Hough ampara no reconhecimento de circunferências (PEDERSEN, 2007). Ela e suas variantes dependem da conversão prévia das imagens para suas versões binarizadas ou em níveis de cinza. A imagem em níveis de cinza é submetida pelo reconhecimento de bordas preliminar e então passa pelo método da votação no espaço paramétrico. A transformada, no entanto, se baseia na equação de uma circunferência, apresentando três parâmetros. Essas características acarretam em uma maior complexidade computacional em sua execução, requerindo mais tempo de processamento e armazenamento em memória (HASSANEIN et al., 2015).

O emprego da Transformada de Hough apresenta alguns benefícios. Em primeiro lugar, a transformada considera cada ponto de borda independente, o que faz o processamento paralelo dos pontos possível e possibilita aplicações em tempo real (HASSANEIN et al., 2015). O método também consegue operar em formas ruidosas ou parcialmente deformadas devido à estratégia de votação. A transformada ainda é apta a detectar múltiplas ocorrências das formas buscadas, uma vez que cada ocorrência é representada em célula específica no espaço paramétrico memória (HASSANEIN et al., 2015).

A transformada apresenta desvantagens. Alto custo computacional e capacidade de armazenamento são requeridos. Quanto maior a dimensão onde se deseja operar, maior o número de cálculos necessários. A alta eficácia do algoritmo depende de métodos de redução na dimensão do acumulador, obtidos através de dados já conhecidos a respeito das formas procuradas.

Figura 5 – Exemplo do espaço de Hough correspondente à imagem de bordas de um olho humano.

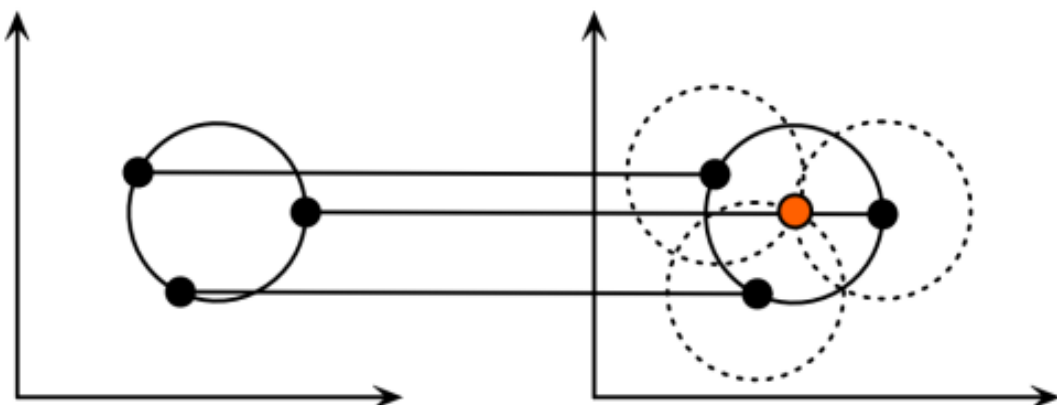


Fonte: (SILVA, 2016)

A Figura 5 ilustra o processo para realização da transformada de Hough. Primeiro há a detecção de bordas, depois é realizada a votação, processo que em cada ponto de borda da imagem desenha-se um círculo no espaço paramétrico com centro no ponto correspondente e raio igual ao procurado.

Após executar o processo para todos os pontos de borda, analisa-se os valores resultantes no acumulador que correspondem ao número de círculos passando por cada par de coordenadas. Os pontos mais votados são os centros de circunferências localizadas na imagem (PEDERSEN, 2007). A Figura 6 ilustra o processo de votação.

Figura 6 – Processo de votação efetuado pela Transformada Circular de Hough.



Fonte: (SILVA, 2016)

2.3 Visão Computacional

A evolução das tecnologias de computação, sensores, processamento de imagens e reconhecimento de padrões culminaram em melhores e mais baratas soluções de inspeção visual e ferramentas automáticas que possuem capacidade de visão (LEAVERS, 1992).

Técnicas de aprimoramento de sistemas de processamento digital de imagens envolvem a criação de algoritmos mais eficazes no processamento. Outro tratamento possível consiste na escolha de linguagens de programação mais apropriadas à aplicação desejada. Melhor performance pode ser obtida através da utilização de hardware específico e mais poderoso ou ainda da exploração do paralelismo e o gerenciamento eficiente de memória.

Atualmente existem várias ferramentas de software para aplicação de visão computacional. OpenCV, DevIL, CImg, Simd e CxImage são exemplos. Elas são bibliotecas e algoritmos voltados à manipulação de imagens, desde a captura ao reconhecimento de padrões e descrição.

O algoritmo desenvolvido (SILVA, 2016) utiliza a biblioteca OpenCV, devido às suas características: extensa documentação e utilização, e custo zero- é uma biblioteca de código aberto.

2.3.1 OpenCV

A biblioteca OpenCV (Open Source Computer Vision Library), auxilia no desenvolvimento de algoritmos para a realização de tarefas que envolvem tratamento de imagem e reconhecimento de padrões. Essa biblioteca é livre para uso acadêmico e comercial, possui interfaces C ++, C, Python e Java e suporta Windows, Linux, Mac OS, iOS e Android. A OpenCV é organizada na forma de módulos, podendo se destacar: matrizes e vetores, para armazenamento de informações, transformações geométricas nas imagens, além de filtragem linear e não linear, tratamento de cor, calibração de câmeras, extração de características e detecção de objetos.

3 Descrição do Sistema

3.1 Raspberry Pi

O sistema de pupilometria desenvolvido utiliza um dispositivo embarcado de baixo custo: o Raspberry Pi 2®, o qual no início foi desenvolvido no Reino Unido com o intuito de promover o ensino da tecnologia da informação.

O Raspberry Pi 2® é dotado do SoC (System on a Chip) BCM2836 com processador quad-core ARM Cortex-A7 funcionando a 900MHz, uma GPU VideoCore IV a 250MHz com capacidade OpenGL ES 2.0, 1 GB de memória RAM, 40 pinos GPIO (General Purpose Input Output) e quatro portas USB.

As portas de entrada e saída - GPIO (General Purpose Input Output) - são utilizadas no controle dos estímulos luminosos incidentes nos olhos durante a captura do ciclo pupilar. Podem ser individualmente configuradas, oferecendo saídas com níveis de tensão 3,3V, 5V e GND, possui também uma porta capaz de fornecer variações graduais através da modulação por largura de pulso (PWM). A Figura 7 ilustra o dispositivo mencionado.

Figura 7 – Raspberry Pi 2®



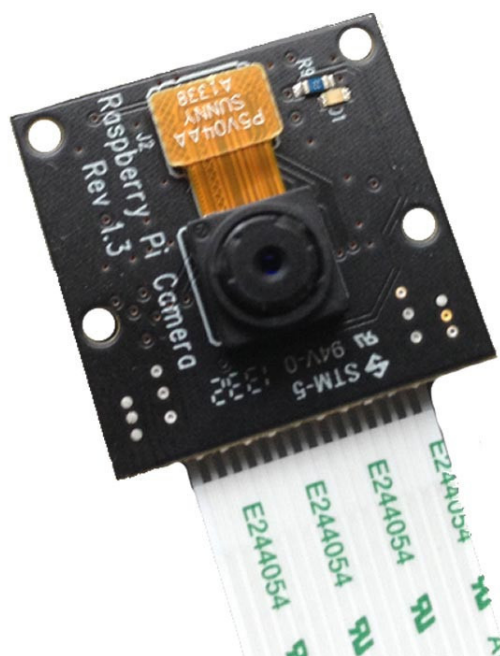
Fonte: *www.wired.com*¹

¹ Disponível em: <<http://www.wired.co.uk/article/raspberry-pi-starter-kit-projects>> acessado em out. 2017.

3.2 Câmera Pi NoIR

O sensor de imagens usado foi a câmera Pi NoIR, distribuída pela fundação Raspberry Pi (FOUNDATION, Out.2017) e dedicada ao dispositivo. Suas características abrangem a captura de imagens à resolução máxima de 2592x1944 pixels (aproximadamente 5 megapixels), vídeos à resolução máxima de 1920x1080 e taxa de quadros máxima de 90 quadros por segundo (a depender da resolução utilizada). A câmera Pi NoIR permite a captura de imagens na faixa infravermelho do espectro eletromagnético sendo, portanto, capaz de obter imagens na falta de luz visível.

Figura 8 – Câmera Pi NoIR



Fonte: www.pi-supply.com ²

3.3 Periféricos: Teclado, Monitor, Mouse

A interface de usuário para execução do software destinado à pupilometria é ambientada no Linux, e requisita a presença de periféricos como: Teclado, monitor e mouse. Teclado e mouse são conectados através de interfaces USB, ao passo que o monitor é conectado à porta HDMI presente no dispositivo Raspberry Pi 2®.

² Disponível em: <<https://www.pi-supply.com/wp-content/uploads/2014/04/raspberry-pi-noir-camera-800x800.jpg>> acessado em out. 2017.

Figura 9 – Exemplo de Periféricos conectados a Raspberry Pi

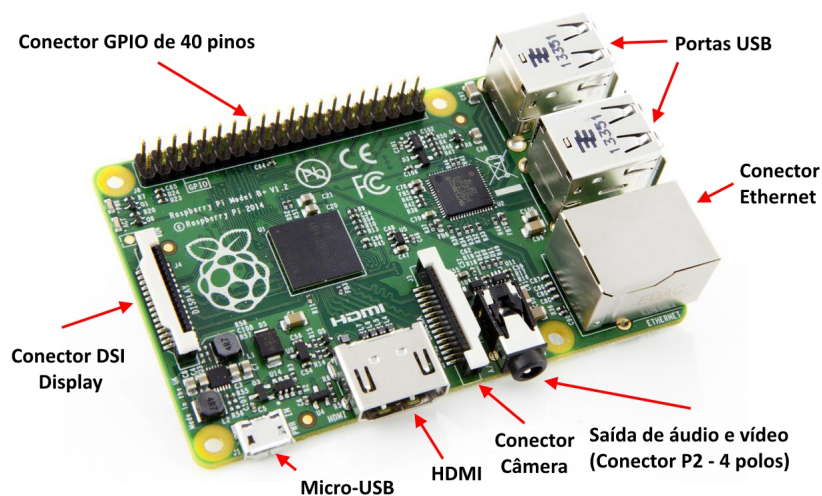


Fonte: www.wired.com.uk ³

3.4 Alimentação

O Raspberry Pi 2® requer alimentação com tensão contínua de 5V provida através da interface micro USB. Recomenda-se a utilização de uma fonte de alimentação capaz de fornecer uma corrente de 2,5A de forma a possibilitar a utilização de periféricos com maior consumo de corrente. As portas GPIO podem fornecer até 50mA distribuídos entre si, enquanto a porta HDMI drena 50mA e o módulo da câmera requer 250mA (UPTON; HALFACREE, 2014).

Figura 10 – Portas da Raspberry Pi 2®



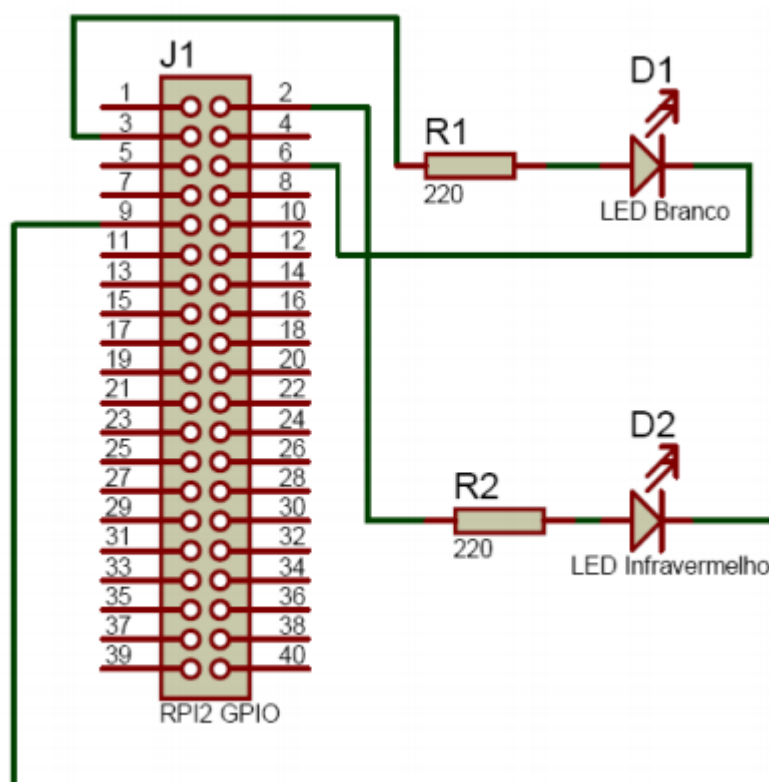
Fonte: www.flipflop.com ⁴

³ Disponível em: <<http://www.wired.co.uk/article/raspberry-pi-starter-ki-projects>> acessado em out. 2017.

3.5 Circuito do LED

Ao incorporar o emprego da câmera Pi NoIR com o emprego de LEDs infravermelho, podemos obter imagens nítidas de um dos olhos do indivíduo na penumbra sincronicamente ao estímulo do outro olho com luz visível. A resposta pupilar ao estímulo luminoso será obtida pelo software desenvolvido. Foram utilizados dois circuitos contendo um LED em série com um resistor limitador de corrente, como pode ser visto na Figura 11 .

Figura 11 – Circuito de controle dos LEDs.

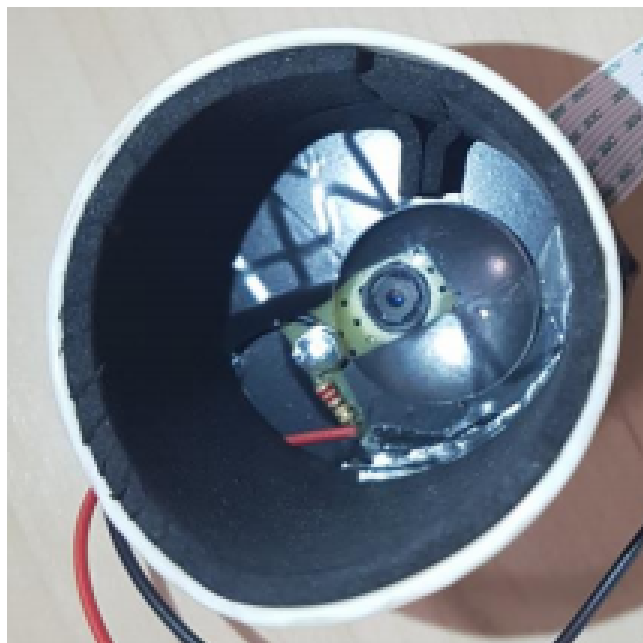


Fonte: (SILVA; VEIGA; MOREIRA, 2017)

Um dos circuitos contém um LED branco responsável pelo estímulo luminoso em um dos olhos do indivíduo sob o teste pupilométrico. O outro circuito emprega um LED infravermelho que possibilita a captura das imagens do outro olho não estimulado sob escuridão total.

⁴ Disponível em: <<https://www.flipeflop.com/wp-content/uploads/2014/08/raspberry-pi-b-plus-conectores1.jpg>> acessado em out. 2017.

Figura 12 – Visor contendo câmera e iluminação LED infravermelho.



Fonte: (SILVA; VEIGA; MOREIRA, 2017)

A intensidade luminosa emitida pode ser controlada utilizando de modulação por largura de pulso (PWM). Os dois resistores de 220 ohms utilizados limitam a corrente nos LEDs, de forma a evitar drenagem excessiva de corrente e danificação dos componentes. A Figura 12 ilustra um dos visores contendo a câmera e a iluminação infravermelho.

3.6 Visores

A atual versão do dispositivo de rastreamento pupilar é composta de dois visores cilíndricos de PVC, com 5 centímetros de diâmetro e 5 centímetros de comprimento, como pode ser visto na Figura 13. Foram adaptadas almofadas de material sintético moldadas ao diâmetro dos visores para maior conforto do usuário e bloqueio da passagem de luz ambiente.

Figura 13 – Protótipo do dispositivo de captura.



Fonte: (SILVA; VEIGA; MOREIRA, 2017)

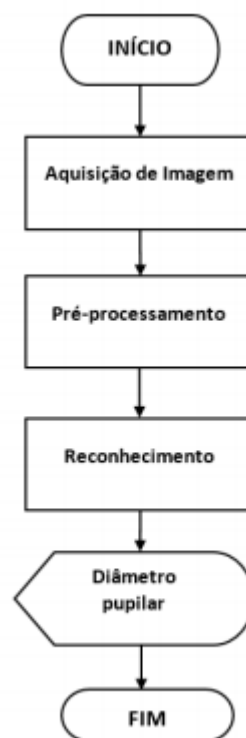
O visor esquerdo é destinado à geração do estímulo luminoso, já o visor direito efetua a filmagem do olho sob total escuridão. Nele estão acoplados o circuito de iluminação infravermelho e a câmera Pi NoIR com uma lente biconvexa para correção da distância focal.

4 Algoritmo

4.1 Funcionamento

O Algoritmo concebido (SILVA, 2016) para localização da circunferência pupilar possui as seguintes etapas: aquisição de imagens, pré-processamento (suavização das imagens) e reconhecimento (localização pupilar através da transformada de hough). Utiliza linguagem C++, usando a biblioteca de código aberto OpenCV (OPENCV, 2017) para processamento de imagens. A Figura 14 ilustra o funcionamento do algoritmo através de um fluxograma.

Figura 14 – Diagrama do algoritmo de rastreamento pupilar.



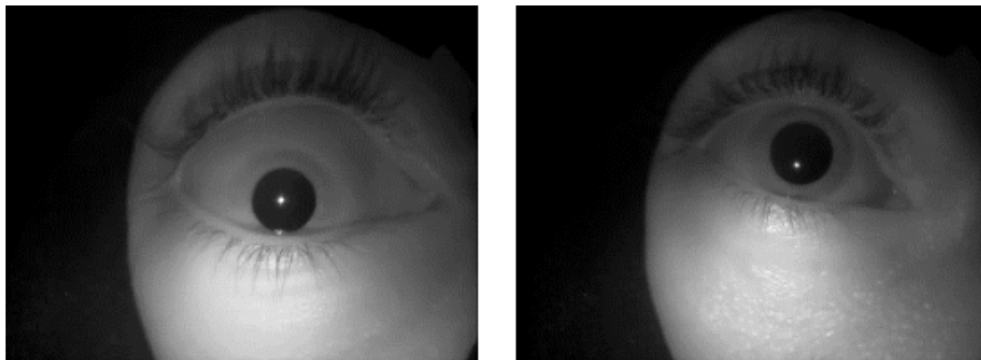
Fonte: (SILVA, 2016).

As funções implementadas para o rastreamento pupilar incluem a suavização –executado pelo filtro de mediana por meio da função ‘medianBlur’ – e a localização de circunferências pelo método de Hough – operada pelo comando ‘HoughCircles’ – que faz automaticamente a detecção de bordas pelo método de Canny (BRADSKI; KAEHLER, 2008).

4.1.1 Aquisição da Imagem

Com o intuito de obter mais velocidade de processamento o algoritmo desenvolvido foi dividido em duas etapas. A primeira etapa realiza a captura do fluxo de vídeo à resolução de 320 x 240 pixels. O algoritmo ainda controla a intensidade luminosa dos LEDs usando as funções da biblioteca Wiring Pi (PI, Out. 2017). A Figura 15 ilustra o processo de captura do fluxo de vídeo na execução do software de aquisição.

Figura 15 – Captura de imagens da câmera durante execução do software no dispositivo Raspberry Pi.



(a) *Amostra 1;*

(b) *Amostra 2;*

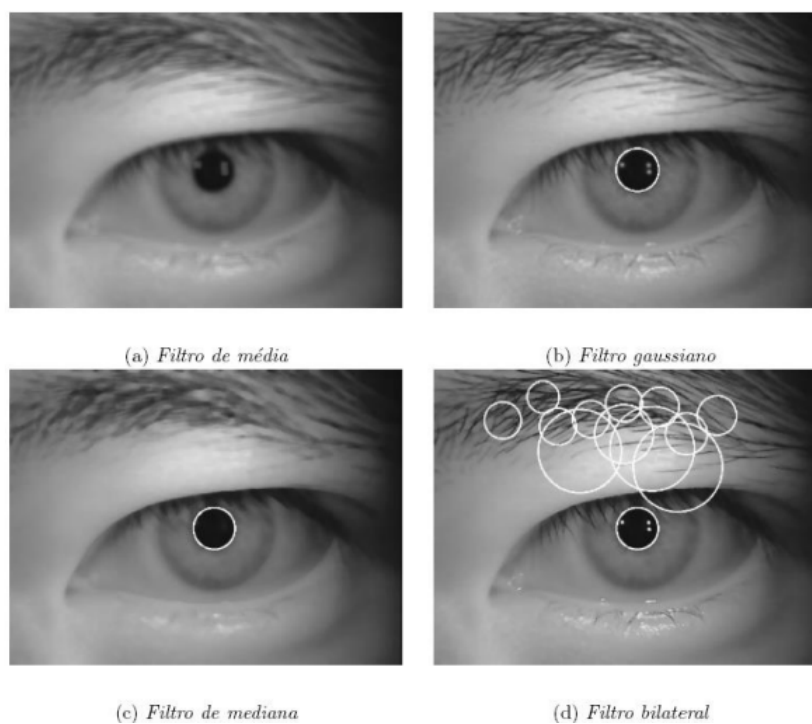
Fonte: (SILVA, 2016).

4.1.2 Pré-Processamento e Reconhecimento

O segundo algoritmo realiza o pré-processamento e reconhecimento das imagens capturadas no software anterior. A inexistência da etapa de pré-processamento da imagem na atual aplicação implica na localização de múltiplas circunferências não coincidentes à pupila. Para o sistema vigente, a melhor configuração de tratamento da imagem para o rastreamento pupilar compreende a suavização da imagem. Esta, no entanto, pode ser efetuada por diferentes filtros e utilizando-se diferentes níveis de suavização. O ajuste de tais parâmetros exerce influência considerável na taxa de localização pupilar, como pode-se observar na Figura 16

Após testes realizados (SILVA, 2016), constatou-se que o melhor filtro para suavização a ser usado é o Filtro de mediana com tamanho de mascara 11.

Figura 16 – Diferentes filtros de suavização.



Fonte: (SILVA, 2016).

Após a realização da etapa de pré-processamento o software realiza o reconhecimento da pupila através da transformada de Hough e retorna os valores aferidos. A Figura 17 ilustra um resultado da aplicação da transformada de Hough.

Figura 17 – Detecção da circunferência pupilar através da transformada de Hough.

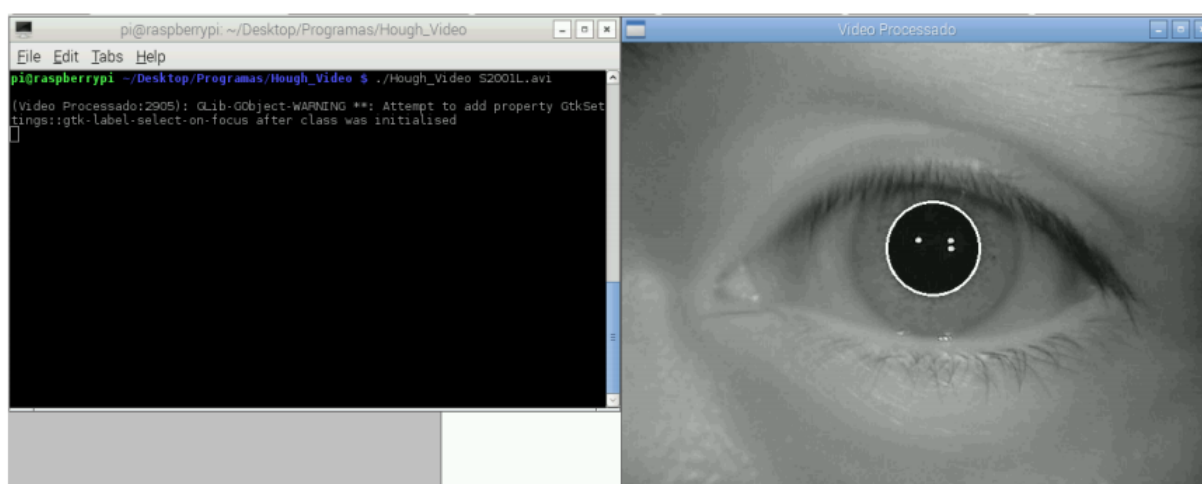


Fonte: (SILVA, 2016).

4.2 Execução

A execução dos softwares para Captura e Análise de vídeos é feita através do terminal de texto da Raspberry Pi. Dessa forma, além dos algoritmos não possuírem interface gráfica, é preciso que o usuário conheça os comandos do sistema operacional Linux para dar início à execução dos programas. Não há a possibilidade do usuário alterar parâmetros de captura e análise de forma dinâmica, apenas alterando o código fonte e recompilando o programa, o que demanda tempo e conhecimento sobre programação. A Figura 18 ilustra o software de análise sendo executado no terminal de texto da Raspberry Pi.

Figura 18 – Software sendo executado através do terminal de texto da Raspberry Pi.



Fonte: (SANTOS et al., 2017).

5 Desenvolvimento da Interface Gráfica para Desktop

5.1 Software QT Creator

O software selecionado para desenvolver a interface foi o QtCreator, desenvolvido pela empresa Trolltech, da Noruega. É um framework de desenvolvimento de interfaces gráficas multiplataforma, incluindo: Android, Blackberry, iOS, Linux e Windows. O Qt não é uma linguagem de programação, e sim um framework em C++. Para ampliar a linguagem C++ e utilizar recursos como 'signals and slots' é utilizado um pré- processador, denominado MOC (Meta-Object Compiler). A partir de arquivos escritos em Qt-extend C++, o MOC gera arquivos no padrão C++. Dessa forma, o framework e as aplicações que o utilizam podem ser compiladas por qualquer compilador compatível com C++ (QT, 2017).

O QtCreator foi escolhido para o desenvolvimento da interface devido a sua facilidade de uso, integração com as bibliotecas C++, compatibilidade multiplataforma - as interfaces criadas são facilmente exportadas entre diferentes plataformas - e devido a suas permissões de uso livre.

5.2 Captura

5.2.1 Parâmetros de captura

Com o intuito de obter máxima velocidade de processamento e ter a opção de processar um vídeo mais de uma vez com diferentes parâmetros, foram elaboradas duas interfaces que atuam separadamente. A primeira faz à aquisição de imagens da pupila. A segunda efetua o pré-processamento (suavização) e localização de circunferências através da transformada de Hough e retorna os valores dos raios pupilares, velocidade e aceleração da contração pupilar.

A primeira interface, nomeada 'Captura', faz a aquisição de vídeo do movimento pupilar. Oferece ao usuário a opção de selecionar a resolução do vídeo - "320x 240" ou "640x480- e possibilita a escolha da duração do estímulo luminoso - 50 a 200 quadros de duração com o LED aceso e 150 a 400 quadros com o LED apagado. A interface com os parâmetros de configuração de captura pode ser observada na Figura 19 .

Figura 19 – Opções de parâmetros criadas para captura.

1 Escolha a resolução que deseja utilizar: 320x240

2 Escolha o tempo que o Led ficará aceso: 120

3 Escolha o tempo que o Led ficará apagado: 280

Fonte: Autoria própria.

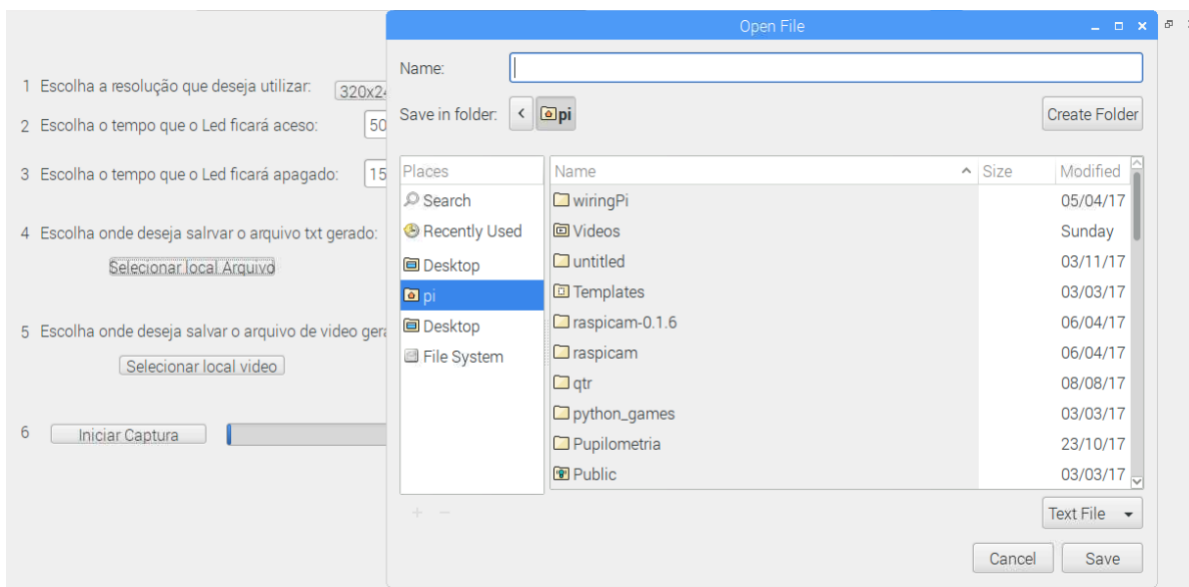
Oferecer ao usuário a opção de escolher a resolução é importante pois permite que o processo seja realizado de acordo com as necessidades e condições do contexto em que a captura de imagens da pupila é realizada. Vídeos capturados na resolução de "320 x 240" ocupam a metade da memória dos capturados com o dobro da resolução, isso pode facilitar em caso de pouca memória livre no dispositivo embarcado. Por outro lado vídeos capturados na resolução de "640 x 480", apesar de ocuparem mais memória, oferecem uma qualidade de imagem melhor, o que facilita no reconhecimento de circunferências e pode reproduzir resultados mais precisos.

A opção de variar o tempo que em o LED fica aceso ou apagado é importante pois adapta o software às necessidades do usuário e ao contexto em que a pupilometria será usada. Como citado anteriormente a pupilometria tem aplicações no constato de várias enfermidades. Portanto variando a duração do estímulo luminoso pode-se usar o software em diversas aplicações da pupilometria.

5.2.2 Salvando os arquivos gerados na captura

No software sem interface gráfica, o diretório em que eram salvos os vídeos e os arquivos de textos gerados na captura era escolhido no algoritmo, não havendo possibilidade do usuário alterar de acordo com sua preferência. A Figura 20 ilustra o botão que ao ser selecionado oferece opção ao usuário de escolher o diretório e o nome do arquivo de texto que será gerado após o processo de captura.

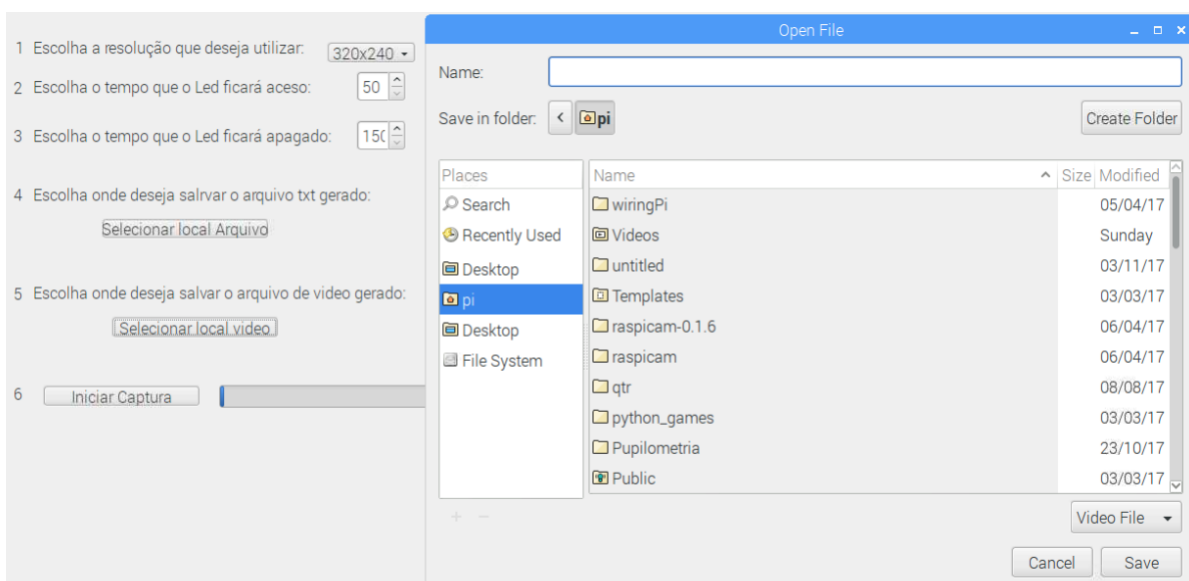
Figura 20 – Janela para salvar arquivos de texto



Fonte: Autoria própria.

A interface gráfica também oferece ao usuário a opção de escolher o diretório onde será salvo o arquivo de vídeo gerado na captura. A Figura 21 ilustra o botão criado para escolher o diretório do vídeo gerado, que ao ser selecionado abre uma janela com as opções de diretórios a serem escolhidas.

Figura 21 – Janela para salvar vídeos

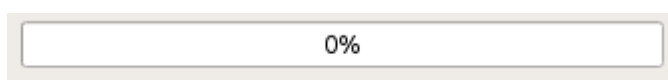


Fonte: Autoria própria.

5.2.3 Barra de progresso

Na execução do software que não possui interface gráfica não é possível acompanhar e estimar o tempo de andamento do processo de captura. Com o intuito de facilitar esse acompanhamento foi desenvolvido uma Barra de progresso que mostra na tela do usuário a porcentagem do andamento do processo de captura em tempo real. A Figura 22 ilustra à barra de progresso antes de iniciar o processo.

Figura 22 – Barra de progresso antes de iniciar o processo de captura.



Fonte: Autoria própria.

Para iniciar a captura de imagens para pupilometria foi desenvolvido um botão de iniciar, que ao ser selecionado, inicia a câmera e faz a captura de imagens. O LED que aplica o estímulo luminoso também é acionado e a barra de progresso mostra ao usuário a porcentagem do processo decorrido de acordo com a quantidade de frames capturados. A Figura 23 ilustra o botão "Iniciar Captura" desenvolvido e a Barra de progresso ilustrando o andar do processo.

Figura 23 – Botão de iniciar e Barra de progresso.



Fonte: Autoria própria.

5.2.4 Visão geral da Interface de Captura

Ao executar o software o usuário pode facilmente selecionar os parâmetros e iniciar a captura de imagens. A interface mostra passo a passo quais ações devem ser executadas, primeiro a seleção dos parâmetros, segundo a escolha do local dos arquivos gerados e por último a seleção do botão de iniciar. Ao fim do processo de captura, o software é reiniciado automaticamente, tendo a barra de progresso zerada para que se possa iniciar um novo processo. A Figura 24 ilustra a interface gráfica sendo executada.

Figura 24 – Visão geral da interface de captura

1 Escolha a resolução que deseja utilizar: 320x240

2 Escolha o tempo que o Led ficará aceso: 120

3 Escolha o tempo que o Led ficará apagado: 280

4 Escolha onde deseja salvar o arquivo txt gerado:
Selecionar local Arquivo

5 Escolha onde deseja salvar o arquivo de vídeo gerado:
Selecionar local vídeo

6 Iniciar Captura 60%

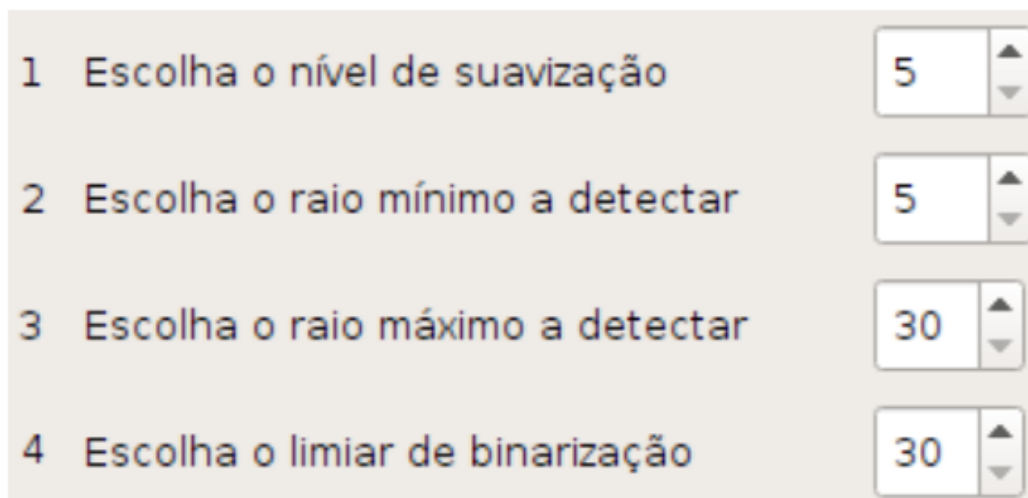
Fonte: Autoria própria.

5.3 Análise

5.3.1 Parâmetros de Análise

Nomeada de ‘Análise’ a segunda interface realiza o pré-processamento do vídeo e aplica a transformada de Hough para detectar os raios pupilares durante a exibição dos quadros. São oferecidos ao usuário quatro parâmetros editáveis para realização do processo: Nível de suavização, raio mínimo a se detectar, raio máximo a se detectar e limiar de binarização, como disposto na Figura 25.

Figura 25 – Parâmetros de Análise.



1 Escolha o nível de suavização	5
2 Escolha o raio mínimo a detectar	5
3 Escolha o raio máximo a detectar	30
4 Escolha o limiar de binarização	30

Fonte: Autoria própria.

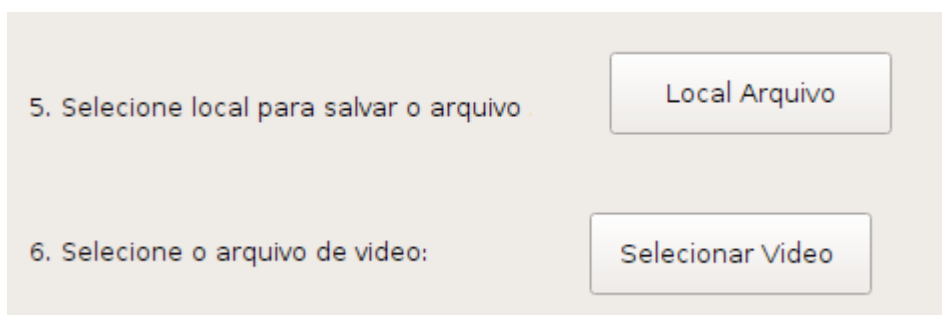
O nível de suavização varia numa escala de 5 a 19, o limiar de binarização varia de 30 a 60. Já o raio mínimo e o raio máximo a serem detectados variam numa escala de 5 a 50 e 30 a 120 pixels respectivamente.

5.3.2 Arquivos

A interface de Análise funciona examinando um vídeo de pupilometria gerado na captura escolhido pelo usuário. Após a análise é gerado um arquivo de texto onde são salvos os resultados, são exibidos os raios pupilares medidos quadro a quadro e no final do arquivo os parâmetros velocidade média, aceleração média, raio mínimo e máximo encontrados.

Dessa forma foram criados dois botões para que se possa selecionar o arquivo de vídeo a ser analisado e o local onde será salvo o arquivo de texto contendo os resultados. A Figura 26 ilustra esses botões.

Figura 26 – Selecionar vídeo e salvar arquivo de texto.

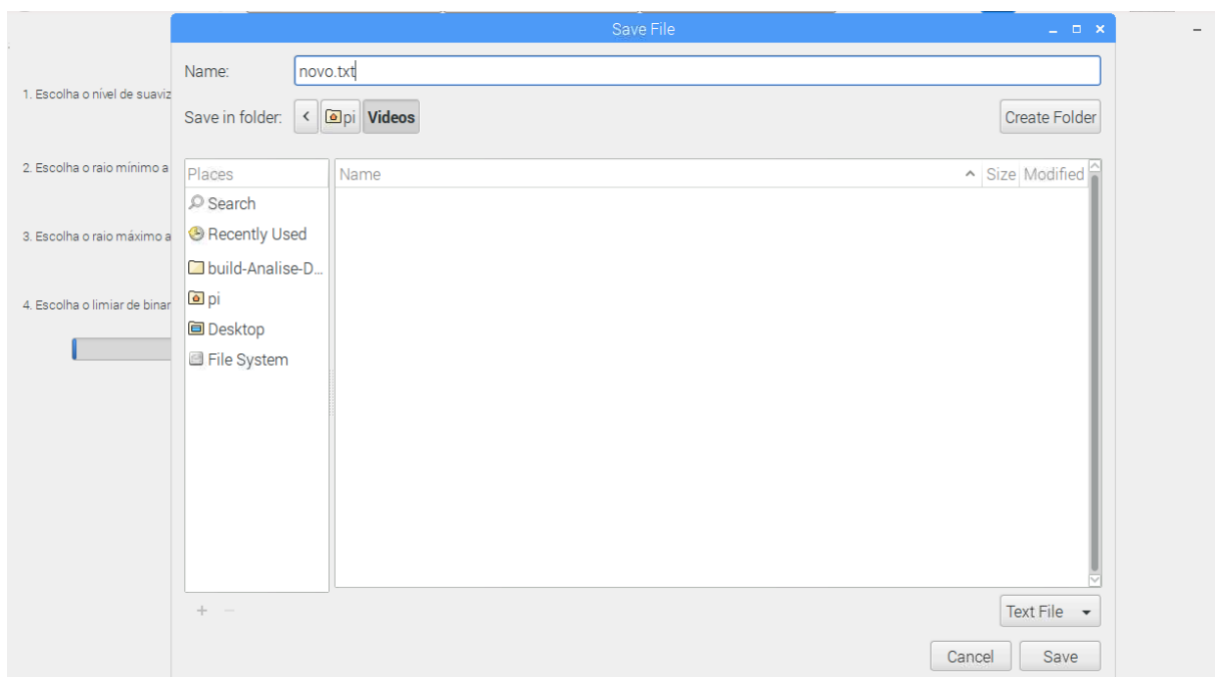


5. Selecione local para salvar o arquivo	Local Arquivo
6. Selecione o arquivo de video:	Selecionar Video

Fonte: Autoria própria.

Quando o usuário seleciona o botão "Local arquivo" uma janela é aberta contendo as opções de diretório onde poderá ser salvo o arquivo de texto que possui os resultados da análise. Ainda é possível escolher o nome do arquivo, tornando o processo bastante dinâmico. A Figura 27 mostra a janela aberta para escolha do diretório onde o arquivo de texto será salvo.

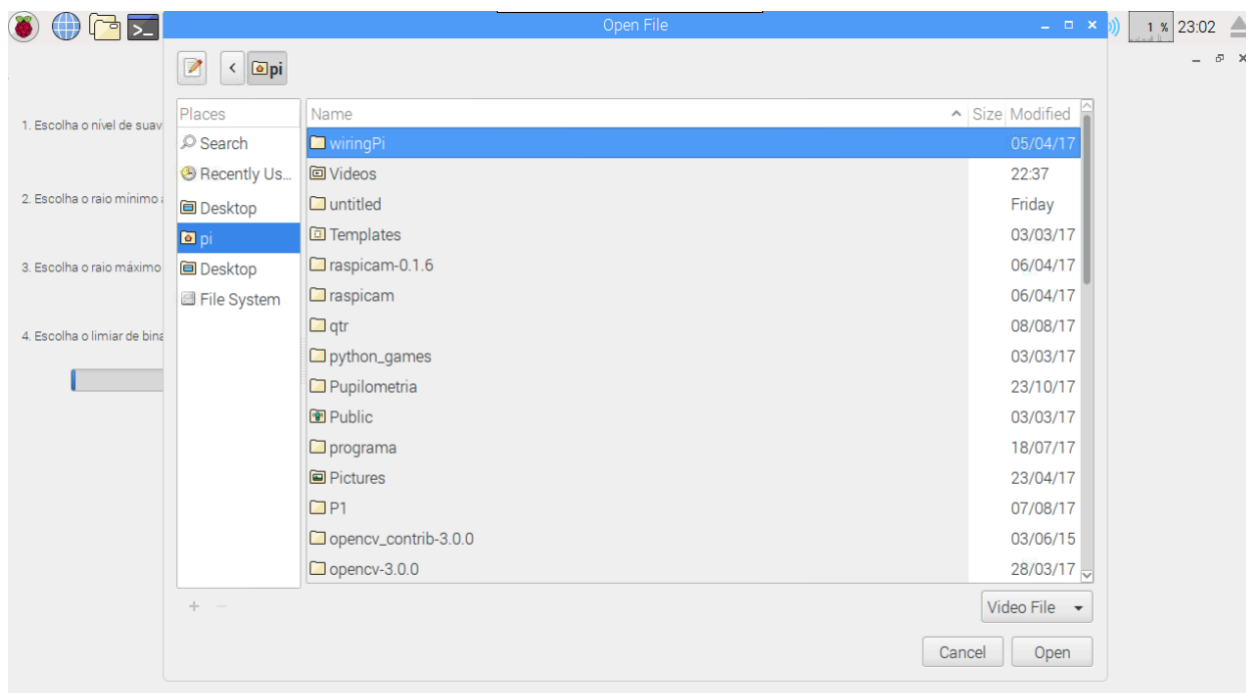
Figura 27 – Salvando o arquivo de texto gerado na Análise.



Fonte: Autoria própria.

No software sem interface gráfica o usuário precisava saber exatamente o caminho do arquivo, já na interface de análise isso não é necessário. Ao selecionar o botão "Selecionar Vídeo" uma janela contendo opções de diretórios também é aberta. Dessa forma é possível escolher um arquivo de vídeo para ser analisado. A Figura 28 mostra o processo de seleção do vídeo.

Figura 28 – Selecionando o arquivo de vídeo gerado na Captura.



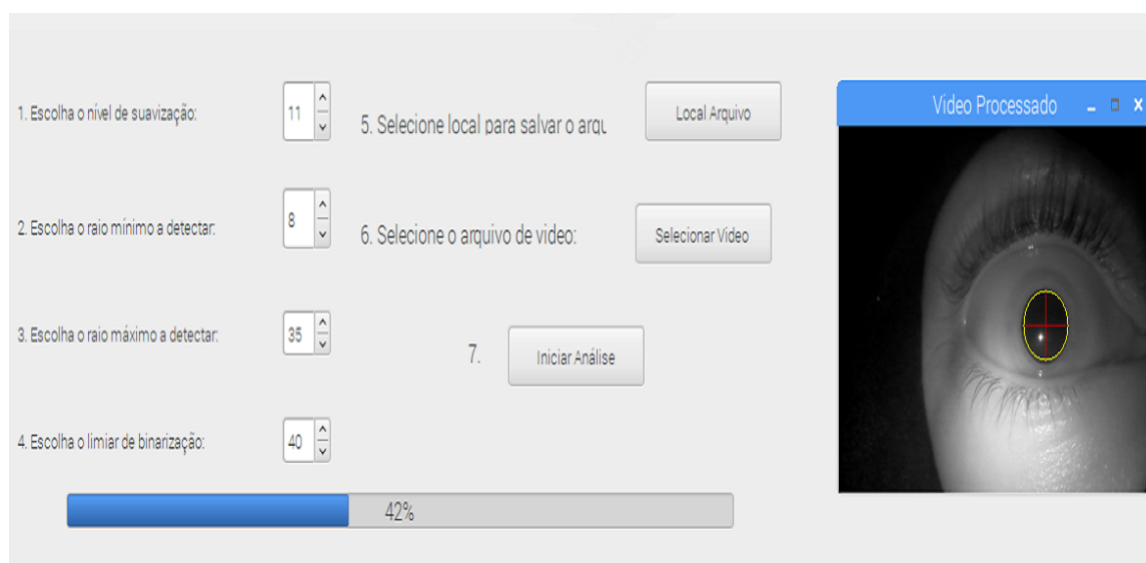
Fonte: Autoria própria.

5.3.3 Visão geral da Interface de Análise

A interface para análise pupilar também apresenta um layout bastante intuitivo, com a indicação de quais passos o usuário deve seguir para realização do exame das imagens de vídeo. Ao clicar no botão "Iniciar Análise" uma janela reproduzindo a sequência de imagens capturadas é exibida, com uma circunferência acompanhando o movimento pupilar.

Assim como na interface de captura, foi desenvolvido uma barra de progresso para auxiliar no processo de análise. A barra ajuda o usuário a estimar o tempo em que o exame das imagens pode levar, exibindo em porcentagem o andamento do processo. Após o fim do exame das imagens, o programa é reiniciado e a barra de progresso zerada. A Figura 29 ilustra a visão geral da interface de análise desenvolvida.

Figura 29 – Visão geral da interface de Análise.



Fonte: Autoria própria.

6 Desenvolvimento da Interface para Tela Sensível ao Toque

6.1 Mudanças no Sistema

O pupilômetro desenvolvido (SILVA; VEIGA; MOREIRA, 2017) foi idealizado para funcionamento usando monitor, teclado e mouse como citado anteriormente. A necessidade desses periféricos torna mais complexa a utilização do dispositivo integrado desenvolvido.

Uma tela sensível ao toque com uma interface otimizada torna o dispositivo mais simples, facilitando sua mobilidade e explorando suas vantagens enquanto dispositivo embarcado. A Figura 30 mostra a tela sensível ao toque de 3,5 polegadas escolhida para o adequamento do hardware.

Figura 30 – Tela Sensível ao Toque de 3,5 polegadas.



Fonte: Autoria própria.

6.2 Captura

Assim como no sistema para desktop, foram elaboradas 2 interfaces para a tela sensível ao toque. Elas incorporam basicamente as mesmas características das anteriores, porém, são adaptadas para o uso em uma tela menor. A Figura 31 ilustra o layout da interface de captura para tela touchscreen.

Figura 31 – Layout da Interface de Captura pra tela sensível ao toque.

1. Escolha a resolução 320x240

2. Tempo Led Aceso 120

3. Tempo Led Apagado 280

4. Iniciar Captura

0%

Arquivos serão salvos em: /home/pi/Videos

Fonte: Autoria própria.

É possível observar que a estrutura da interface gráfica é basicamente a mesma da desenvolvida para desktop, porém os botões são maiores, visando a adaptação para o uso dos dedos em contato direto com a tela. O tamanho da janela também foi adaptado para o padrão 320x480 da tela, objetivando facilitar a visualização dos componentes do programa.

A Figura 32 ilustra a execução do software de captura usando a interface gráfica desenvolvida para a tela de 3,5 polegadas sensível a toques.

Figura 32 – Execução da interface de Captura Touchscreen.



Fonte: Autoria própria.

6.2.1 Arquivos

Na interface gráfica desenvolvida para desktop, o usuário tem a opção ao abrir uma janela, de escolher o diretório e o nome dos arquivos gerados. Na interface concebida para a tela touchscreen não há essa opção devido a dificuldade na manipulação dos botões.

Dessa forma o botão que abria a janela para escolha de diretórios foi eliminado e os arquivos são salvos automaticamente em um diretório pré-estabelecido no algoritmo. Uma informação na tela auxilia o usuário acerca do local onde os arquivos gerados serão salvos. A Figura 33 ilustra o texto que informa o usuário sobre o diretório que recebera o vídeo e arquivo txt gerado na Captura.

Figura 33 – Texto de auxílio ao usuário na Captura Touchscreen.

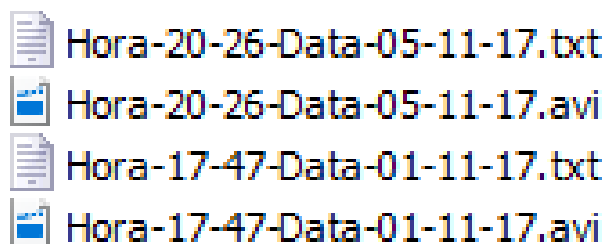
Arquivos serão salvos em: /home/pi/Videos

Fonte: Autoria própria.

Diferentemente da interface para Desktop, não há a opção do usuário escolher o

nome dos arquivos de vídeo e texto gerados no processo de captura. Deste modo, para facilitar a identificação foi criado um algoritmo que dá ao nome dos arquivos a hora e data do processo de captura. Dessa maneira, os arquivos de vídeo e texto gerados no mesmo processo terão o mesmo nome em função do horário e data. A Figura 34 ilustra os arquivos salvos.

Figura 34 – Formato dos nomes dos arquivos gerados no processo de captura em touchscreen.

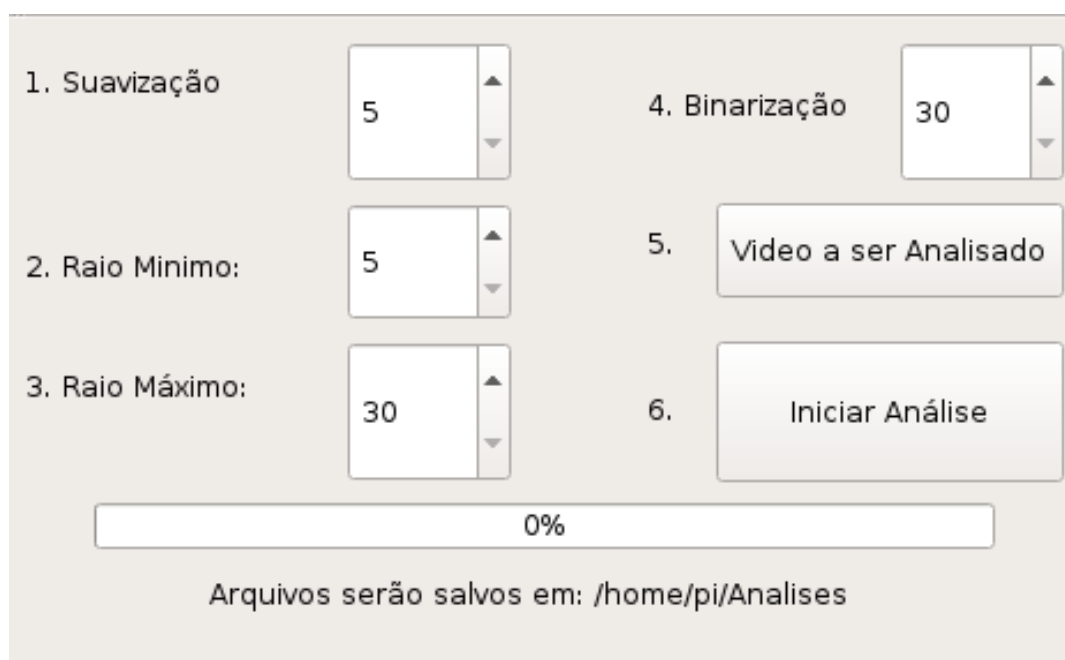


Fonte: Autoria própria.

6.3 Análise

Foi desenvolvida também uma interface de análise destinada a tela de 3,5 polegadas. Assim como no software de Captura não houve grandes mudanças em relação ao software destinado a desktop. A Figura 35 ilustra o layout da interface de análise desenvolvida para telas touchscreen.

Figura 35 – Layout da Interface de Análise pra tela sensível ao toque.



Fonte: Autoria própria.

O tamanho da janela segue o mesmo padrão da interface de captura, sendo adaptada para resolução 320x480 para facilitar a visualização dos elementos. Os botões também foram adequados para um tamanho maior, visando facilitar o uso dos dedos na operação do programa. Uma barra de progresso auxilia o usuário sobre o andamento do processo de análise, diferentemente da interface de análise para desktop, não há uma janela exibindo as imagens da pupilometria, devido a falta de espaço em tela.

A Figura 36 ilustra a execução do software de análise usando a interface gráfica desenvolvida para a tela de 3,5 polegadas sensível a toques.

Figura 36 – Execução da interface de Análise Touchscreen.



Fonte: Autoria própria.

6.3.1 Arquivos

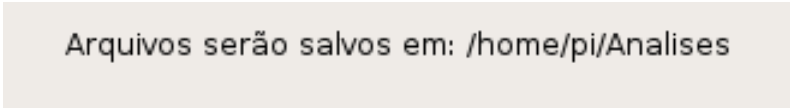
No software de Análise é preciso selecionar um arquivo de vídeo para ser analisado. Após o exame das imagens um arquivo de texto contendo os resultados é gerado. A seleção do arquivo de vídeo na interface para tela sensível ao toque funciona da mesma maneira da desenvolvida para desktop.

Ao clicar no botão "Vídeo a ser Analisado" uma janela é aberta exibindo os diretórios, onde se pode navegar e escolher o vídeo pretendido. Porém, diferentemente do software para desktop, na interface para touchscreen não é possível escolher o diretório em que

será salvo o arquivo de texto contendo os resultados, devido as dificuldades de se usar um teclado virtual na tela de 3,5 polegadas.

Dessa forma, assim como na interface de captura para touchscreen, o local em que serão salvos os arquivos contendo os resultados da análise é pré-selecionado no algoritmo e uma informação na tela do programa auxilia o usuário sobre o caminho em que esses estarão salvos. A Figura 37 ilustra o texto que auxilia o usuário a cerca do diretório pré-selecionado no algoritmo.

Figura 37 – Texto de auxílio ao usuário na Análise Touchscreen.

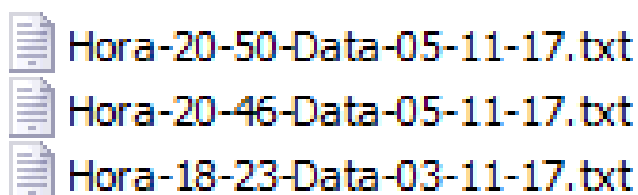


Arquivos serão salvos em: /home/pi/Analises

Fonte: Autoria própria.

O mesmo algoritmo usado anteriormente que inclui no nome dos arquivos a suas respectivas horas e datas foi usado na interface de análise para tela sensível ao toque. Ao realizar o exame das imagens, um arquivo de texto com o nome do horário da análise é gerado e salvo automaticamente. A Figura 38 ilustra os resultados das análises salvos.

Figura 38 – Formato dos nomes dos arquivos gerados no processo de análise em touchscreen.



Hora-20-50-Data-05-11-17.txt
Hora-20-46-Data-05-11-17.txt
Hora-18-23-Data-03-11-17.txt

Fonte: Autoria própria.

6.4 Comparações entre Sistemas

Foram concebidos 2 sistemas, um para Desktop funcionando com teclado, mouse e monitor e o outro atuando com a tela sensível ao toque. Ambos os sistemas apresentam vantagens e desvantagens em suas utilizações.

A interface para tela sensível ao toque foi desenvolvida com o intuito de obter mobilidade para o sistema. Dessa forma, a estrutura se tornou mais simples, sendo composta pelos visores e a Raspberry Pi ligada à tela de 3,5 polegadas. O uso de uma bateria externa pode eliminar a necessidade de uma tomada para alimentar o dispositivo embarcado, tornando-o totalmente móvel.

É notável que o processo de captura das imagens se torna muito mais rápido e prático usando a tela sensível ao toque. Por outro lado, usando o sistema para Desktop, muito tempo é perdido realizando as conexões dos periféricos ao sistema embarcado e a mobilidade é quase nula.

Em contrapartida, apesar de ser possível realizar o processo de análise com a interface para tela sensível ao toque, é notável que usando um Desktop é possível manipular o arquivo de texto gerado e observar de forma mais clara os resultados obtidos.

Uma solução para tornar mais prática ambas as etapas do processo de pupilometria seria o uso de uma tela sensível ao toque de tamanho mínimo de sete polegadas, o que facilitaria a manipulação dos elementos em tela e o uso de um teclado virtual.

7 Conclusões e Trabalhos Futuros

7.1 Principais Contribuições

A principal contribuição deste trabalho consiste na elaboração de uma interface gráfica que facilita o processo de pupilometria devido à sua fácil utilização e integração dos estágios de coleta de amostras e análise pupilométrica. Esta plataforma possibilita que usuários que não possuem entendimento sobre o código fonte do algoritmo configurem os parâmetros desejados de forma descomplicada, interativa e em tempo real, sem a obrigação de recompilar o software.

A interface concebida para tela sensível ao toque possibilita uma maior locomoção ao hardware, dispensando o uso de teclado, mouse e monitores. Para mais, o software possui compatibilidade multiplataforma, permitindo facilmente sua migração entre sistemas diferentes, compreendendo dispositivos embarcados e móveis.

7.2 Trabalhos Futuros

A interface gráfica desenvolvida neste trabalho apresentou desempenho satisfatório e satisfaz os objetivos inicialmente propostos. Porém, existem possibilidades de pesquisas complementares a serem averiguadas.

O protótipo de visores desenvolvido (SILVA, 2016) pode ser melhorado, de maneira a fazê-lo mais ergonômico, ajustável e adequado ao uso contínuo. A tecnologia de impressão 3D é aparentemente a melhor possibilidade no projeto de um visor que hospede todos os elementos de hardware, incluindo a tela sensível ao toque e uma bateria, tornando o dispositivo totalmente portátil.

A tela sensível ao toque que possui tecnologia resistiva pode ser trocada por uma tela maior de tecnologia capacitiva (possui resposta mais rápida aos toques), o que permitiria um uso da interface de forma mais confortável e ágil.

Complementarmente podem ser averiguadas implementações do algoritmo em hardwares capazes de disponibilizar um melhor desempenho no processamento massivo de imagens requisitado pelo algoritmo. Dispositivos como DSPs ou mesmo FPGAs possivelmente alcançarão desempenho superior.

Referências

- BOVIK, A. C. Introduction to Digital Video Processing. *The Essential Guide to Video Processing*, p. 1–9, 2009. Citado na página 17.
- BRADSKI, G.; KAEHLER, A. *07 Learning OpenCV: Computer Vision with the OpenCV Library*. [S.l.: s.n.], 2008. v. 1. Citado na página 31.
- CANNY, J. *A computational approach to edge detection*. *Pattern Analysis and Machine Intelligence*. [S.l.]: IEEE Transactions on, 1986. Citado na página 20.
- DIAS, A. G. d. C. *Pupilometria dinâmica: Uma proposta de rastreamento da posição e tamanho da pupila humana em tempo real*. Mestrado, 2014. Citado 4 vezes nas páginas 17, 18, 21 e 22.
- EUSTACE S. MURNAGHAN, M. D. P. Pupil constriction to dilute pilocarpine: a useful clinical sign of autonomic involvement in diabetic neuropathy. 1981. Citado na página 14.
- FAN, X. et al. Abnormal transient pupillary light reflex in individuals with autism spectrum disorders. *Journal of Autism and Developmental Disorders*, v. 39, n. 11, p. 1499–1508, Nov 2009. ISSN 1573-3432. Citado na página 14.
- FERRARI, G. L. Pupilometria dinâmica : Aplicação na detecção e avaliação da neuropatia autonômica diabética e estudo da correlação entre a resposta temporal. *Utfpr*, 2008. Citado na página 14.
- FOTIOU, F. et al. Changes in pupil reaction to light in Alzheimer’s disease patients: A preliminary report. In: *International Journal of Psychophysiology*. [S.l.: s.n.], 2000. v. 37, n. 1, p. 111–120. ISBN 3031435702. ISSN 01678760. Citado na página 14.
- FOUNDATION, R. P. *Raspberry Pi Foundation, Raspicam documentation*. [S.l.]. Disponível em: <<http://www.raspberrypi.org/documentation/hardware>>. Acessado em Out.2017 Citado na página 26.
- GRANHOLM, E. et al. Tropicamide effects on pupil size and pupillary light reflexes in Alzheimer’s and Parkinson’s disease. *International Journal of Psychophysiology*, v. 47, n. 2, p. 95–115, 2003. ISSN 01678760. Citado na página 14.
- GRÜNBERGER, J. et al. Static and dynamic pupillometry for determination of the course of gradual detoxification of opiate-addicted patients. *European Archives of Psychiatry and Clinical Neuroscience*, v. 240, n. 2, p. 109–112, 1990. ISSN 0175758X. Citado na página 14.
- HART, P. E.; STORK, D. G.; WILEY, J. Pattern classification and scene analysis richard o. duda and peter e. hart. wiley, new york, 1973, 482 pages, 22.50. *Artificial Intelligence*, v. 4, p. 139–143, 1973. Citado na página 22.
- HASSANEIN, A. S. et al. A Survey on Hough Transform, Theory, Techniques and Applications. *arXiv preprint arXiv:1502.02160*, p. 18, 2015. Citado na página 22.

HOWE, K. D. *A practical introduction to computer vision with opencv*. [S.l.]: John Wiley and Sons, 2014. Citado 2 vezes nas páginas 20 e 21.

KEIVANIDOU, A. et al. Evaluation of autonomic imbalance in patients with heart failure: A preliminary study of pupillomotor function. *Cardiology Journal*, v. 17, n. 1, p. 65–72, 2010. ISSN 18975593. Citado na página 14.

LAGANIÈRE, R. *OpenCV 2 Computer Vision Application Programming Cookbook*. [S.l.: s.n.], 2011. 304 pages p. Citado na página 18.

LEAVERS, V. *Shape Detection in Computer Vision Using the Hough Transform*. [S.l.: s.n.], 1992. 201 p. Citado na página 24.

MACLEAN, H.; DHILLON, B. Pupil cycle time and human immunodeficiency virus (hiv) infection. *Eye*, v. 7 (Pt 6), p. 785–6, 1993. Citado na página 14.

MOESLUND, T. *Introduction to Video and Image Processing: Building Real Systems and Applications*. Springer London, 2012. (Undergraduate Topics in Computer Science). ISBN 9781447125037. Disponível em: <<https://books.google.com.br/books?id=fQdqsUyTJqIC>>. Citado na página 18.

NIXON, M. S.; AGUADO, A. S. Feature Extraction and Image Processing. *Academic Press*, p. 88, 2008. Citado 2 vezes nas páginas 19 e 20.

OPENCV. *Open source computer vision*. Disponível em: <<http://opencv.org/>>. Acessado em Out.2017. Citado na página 31.

PEDERSEN, S. J. K. Circular hough transform. 2007. Citado 2 vezes nas páginas 22 e 23.

PI, W. *Gpio interface library for the raspberry pi*. Disponível em: <www.wiringpi.com>. Acessado em Out.2017. Citado na página 32.

QT página oficial. *about Qt*. 2017. Disponível em: <<http://www.qt.io>>. Acessado em Out.2017. Citado na página 35.

QUEIROZ, J. E. R. de; GOMES, H. M. Introdução ao processamento digital de imagens. *RITA*, v. 13, p. 11–42, 2006. Citado 2 vezes nas páginas 20 e 21.

RIZOS, G. et al. Pupillometry in parkinson's disease correlations with neuroimaging techniques. v. 54, p. 41–42, 09 2004. Citado na página 14.

ROSSE, R. B. et al. Anxiety and pupil reactivity in cocaine dependent subjects endorsing cocaine-induced paranoia: preliminary report. *Addiction*, v. 90, n. 7, p. 981–984, 1995. ISSN 13600443. Citado na página 14.

SACKS, B.; SMITH, S. People with down's syndrome can be distinguished on the basis of cholinergic dysfunction. *Journal of Neurology, Neurosurgery & Psychiatry*, BMJ Publishing Group Ltd, v. 52, n. 11, p. 1294–1295, 1989. ISSN 0022-3050. Citado na página 14.

SANTOS, G. C. et al. Desenvolvimento de uma Interface Gráfica para Sistema Embarcado de Pupillometria. *Conferência de Estudos em Engenharia Elétrica*, p. 1–6, 2017. Citado na página 34.

SILVA; VEIGA; MOREIRA. Low Cost Portable Pupil Tracking System Embedded In Raspberry Pi 2. 2017. Citado 6 vezes nas páginas 18, 20, 28, 29, 30 e 44.

SILVA, R. A. da. Desenvolvimento de um sistema embarcado para pupilometria. *Universidade Federal de Uberlândia*, 2016. Citado 13 vezes nas páginas 14, 17, 18, 19, 20, 21, 22, 23, 24, 31, 32, 33 e 51.

SOKOLSKI, K. N.; DEMET, E. M. Increased pupillary sensitivity to pilocarpine in depression. *Progress in Neuro-Psychopharmacology and Biological Psychiatry*, v. 20, n. 2, p. 253–261, 1996. ISSN 02785846. Citado na página 14.

TAN, E. T. et al. Parasympathetic denervation of the iris in alcoholics with vagal neuropathy. *Journal of Neurology, Neurosurgery and Psychiatry*, v. 47, n. 1, p. 61–64, 1984. ISSN 0022-3050. Citado na página 14.

UPTON, E.; HALFACREE, G. *Raspberry Pi User Guide, 3rd Edition*. [S.l.: s.n.], 2014. Citado na página 27.

Apêndices

APÊNDICE A – Código Fonte das Interfaces Gráficas

A.1 Código: Interface de Captura para Desktop

```

1 #include "mainwindow.h"
2 #include "ui_mainwindow.h"
3 #include <opencv2/opencv.hpp>
4 #include <raspicam/raspicam_cv.h>
5 #include <wiringPi.h>
6 #include <iostream>
7 #include <fstream>
8 #include <unistd.h>
9 #include <opencv2/highgui.hpp>
10 #include <opencv2/imgcodecs.hpp>
11 #include <opencv2/imgproc.hpp>
12 #include <QFileDialog>
13 #include <QMessageBox>
14 #include <raspicam/raspicam.h>
15 #include <QApplication>
16 #include <QProcess>
17
18 using namespace std;
19 using namespace cv;
20 QString filename;
21 QString filename1;
22 int Aceso=0;
23 int Apagado=0;
24 int Conta1;
25 int count1 = 0;
26 int resolucao1;
27 int resolucao2;
28
29 MainWindow::MainWindow(QWidget *parent) :
30 QMainWindow(parent),
31 ui(new Ui::MainWindow)
32 {
33 ui->setupUi(this);

```

```
34 QWidget::showMaximized();
35 }
36
37 MainWindow::~MainWindow()
38 {
39     delete ui;
40 }
41
42 void MainWindow::on_pushButton_clicked()
43 {
44
45     std::string str(filename.toLatin1().data());
46     Aceso= ui->spinBox->value()+200;
47     Apagado= (ui->spinBox_2->value());
48     if (ui->comboBox->currentIndex()==0)
49     {
50         resolucao1= 320;
51         resolucao2=240;
52
53     }
54     if(ui->comboBox->currentIndex()==1){
55         resolucao1=640;
56         resolucao2=480;}
57
58
59
60
61
62 Mat FrameAtual, suave_image;
63
64
65
66 ofstream arquivo;
67
68 raspicam::RaspiCam_Cv capture;
69 capture.set(CV_CAP_PROP_FORMAT, CV_8UC1 );
70 capture.set( CV_CAP_PROP_FPS, 60 );
71 capture.set ( CV_CAP_PROP_FRAME_WIDTH, resolucao1);
72 capture.set ( CV_CAP_PROP_FRAME_HEIGHT, resolucao2 );
73 capture.set( CV_CAP_PROP_FPS, 30 );
74
75
```

```
76 if(!capture.open()) {cerr<<"Erro ao abrir camera\n";printf ("-1")
    ;}
77 usleep(500000);
78
79 bool stop(false);
80
81 wiringPiSetup();
82 pinMode(0, OUTPUT);
83 digitalWrite(0,LOW);
84
85 double FrameRate = capture.get(CV_CAP_PROP_FPS);
86 cout<<"\nFPS camera: "<<FrameRate;
87 arquivo.open(filename1.toStdString().c_str(), ios::out);
88
89 Size S = Size(resolucao1,resolucao2);
90 int fourcc = CV_FOURCC('H','2','6','4');
91 VideoWriter saida_Video;
92 saida_Video.open(str, fourcc, 12, S, false);
93 double time=cv::getTickCount();
94 Conta1= Aceso+ Apagado;
95
96 ui->progressBar->setMaximum(Conta1);
97 while (count1<Conta1) {
98
99 capture.grab();
100 capture.retrieve(FrameAtual);
101
102 saida_Video.write(FrameAtual);
103 count1++;
104
105 ui->progressBar->setValue(count1);
106 ui->progressBar->update();
107
108 if(count1==200)
109 digitalWrite(0,HIGH);
110
111
112 if(count1==Aceso)
113 digitalWrite(0,LOW);
114
115
116 cout<<"\n"<<count1<<" " <<cv::getTickCount() /double ( cv::
```

```
        getTickFrequency());
117 arquivo<<count1<<"\t"<<cv::getTickCount() /double ( cv::
        getTickFrequency())<<endl;
118
119 }
120
121 double secondsElapsed= double ( cv::getTickCount()-time ) /double
        ( cv::getTickFrequency() ); //tempo em segundos
122 cout<<endl<< secondsElapsed<<" segundos para capturar "<<count1<<
        " frames : FPS = "<< ( float ) ( ( float ) ( count1) /
        secondsElapsed ) <<endl;
123
124
125 arquivo.close();
126 QApplication->quit();
127 QProcess::startDetached(QApplication->arguments()[0], QApplication->arguments());
128
129
130
131
132 }
133
134
135
136
137 void MainWindow::on_pushButton_2_clicked()
138 {
139     filename = QFileDialog::getSaveFileName(
140
141     this,
142     tr("Open File"),
143     "/home/pi",
144     "Video File (*.avi);;"
145
146 );
147 }
148
149
150 void MainWindow::on_pushButton_3_clicked()
151 {
152
153
```

```
154 filename1 = QFileDialog::getSaveFileName(  
155  
156 this,  
157 tr("Open File"),  
158 "/home/pi",  
159 tr(" Text File (*.txt);;")  
160  
161 );  
162  
163  
164 }
```

A.2 Código: Interface de Análise para Desktop

```
1  
2 #include "mainwindow.h"  
3 #include "ui_mainwindow.h"  
4 #include <opencv2/opencv.hpp>  
5 #include <iostream>  
6 #include <fstream>  
7 #include <QFileDialog>  
8 #include <QMessageBox>  
9 #include <QApplication>  
10 #include <QProcess>  
11  
12 QString filename;  
13 QString filename1;  
14 int suav;  
15 int raiomin;  
16 int raiomax;  
17 int bincanny;  
18 using namespace std;  
19 using namespace cv;  
20  
21 MainWindow::MainWindow(QWidget *parent) :  
22 QMainWindow(parent),  
23 ui(new Ui::MainWindow)  
24 {  
25 ui->setupUi(this);  
26 }  
27  
28 MainWindow::~MainWindow()
```

```
29 {
30 delete ui;
31 }
32
33 void MainWindow::on_pushButton_3a_clicked()
34 {
35 filename1 = QFileDialog::getSaveFileName(
36
37 this,
38 tr("Save File"),
39 "/home/pi",
40 " Text File (*.txt);;"
41
42 );
43
44 if (filename1!= NULL)
45 { QMessageBox::information(this, tr("File Name"), filename1);
46 }
47 else
48 QMessageBox::information(this, tr("File Name"), "Nenhum arquivo
         selecionado");
49
50 }
51
52 void MainWindow::on_pushButton_2_clicked()
53 {
54 filename = QFileDialog::getOpenFileName(
55
56 this,
57 tr("Open File"),
58 "/home/pi",
59 "Video File (*.avi);;"
60
61 );
62
63 if (filename!= NULL)
64 { QMessageBox::information(this, tr("File Name"), filename);
65 }
66 else
67 QMessageBox::information(this, tr("File Name"), "Nenhum arquivo
         selecionado");
68 }
```

```
69
70 void MainWindow::on_pushButton_clicked()
71 {
72
73     suav= ui->spinBox->value();
74     raiomin= ui->spinBox_2->value();
75     raiomax= ui->spinBox_3->value();
76     bincanny= ui->spinBox_4->value();
77
78
79     const int64 inicio = getTickCount();
80     std::string str(filename.toLatin1().data());
81     Mat FrameAtual, suave_image;
82     int numFrames = 0;
83     vector<Vec3f> circles;
84     ofstream arquivo;
85     double raio_max = 0;
86     double raio_min = 50;
87     double latencia, amplitude, quadro_raio_min, quadro_raio_max,
88         quadro_acomodacao, veloc_media, acel_media;
89     bool marcador=false;
90     bool marcador_latencia = false;
91     namedWindow("Video Processado", CV_WINDOW_AUTOSIZE);
92
93
94     VideoCapture capture(str);
95     if (!capture.isOpened())
96     printf ("1");
97
98     bool stop(false);
99
100    double FrameRate = capture.get(CV_CAP_PROP_FPS);
101    int NumeroFrames =capture.get(CV_CAP_PROP_FRAME_COUNT);
102    int conta1= NumeroFrames -1;
103    int Delay = 1000 / 22;
104    double raio[NumeroFrames],posicaoX[NumeroFrames], posicaoY[
105        NumeroFrames], velocidade[NumeroFrames], aceleracao[
106        NumeroFrames];
107    arquivo.open(filename1.toStdString().c_str(), ios::out);
```

```
108
109 while (!stop) {
110   if (!capture.read(FrameAtual))
111     break;
112   ++numFrames;
113
114   cvtColor(FrameAtual, FrameAtual, CV_BGR2GRAY);
115
116   medianBlur(FrameAtual, suave_image, suav);
117
118
119
120   HoughCircles(suave_image, circles, CV_HOUGH_GRADIENT, 3,
121               FrameAtual.rows / 1.5, bincanny, 50, raiomin, raiomax);
122
123   vector<Vec3f>::const_iterator itc = circles.begin();
124
125   cvtColor(FrameAtual, FrameAtual, CV_GRAY2BGR);
126
127   while (itc != circles.end()) {
128     circle(FrameAtual, Point((*itc)[0], (*itc)[1]), (*itc)[2], Scalar
129           (0, 255, 255), 1);
129     line(FrameAtual, Point((*itc)[0] - 20, (*itc)[1]), Point((*itc)
130           [0] + 20, (*itc)[1]), Scalar(0, 0, 200), 1, 8);
130     line(FrameAtual, Point((*itc)[0], (*itc)[1] - 20), Point((*itc)
131           [0], (*itc)[1] + 20), Scalar(0, 0, 200), 1, 8);
131     ++itc;
132   }
133   imshow("Video Processado", FrameAtual);
134
135
136   if (!circles.empty()) {
137     cout << "\nRaio encontrado no frame " << numFrames << ": " <<
138           circles[0][2] << " na posicao " << circles[0][0] << ", " <<
139           circles[0][1];
138     raio[numFrames] = circles[0][2];
139     posicaoX[numFrames] = circles[0][0];
140     posicaoY[numFrames] = circles[0][1];
141
142     if (numFrames > 2 && raio[numFrames] - raio[numFrames - 1] > 10)
143       raio[numFrames] = raio[numFrames - 1];
```



```
144
145 velocidade[numFrames] = (raio[numFrames] - raio[numFrames - 1]);
146 aceleracao[numFrames] = (velocidade[numFrames] - velocidade[
    numFrames - 1]);
147
148 if (numFrames < 250 && raio_max < circles[0][2]) {
149     raio_max = circles[0][2];
150     quadro_raio_max = numFrames;
151 }
152 if (numFrames > 200 && raio_min > circles[0][2]) {
153     raio_min = circles[0][2];
154     quadro_raio_min = numFrames;
155 }
156 if (numFrames > 340 && abs(circles[0][2] - raio_max) < 2 && marcador
    == false) {
157     quadro_acomodacao = numFrames;
158     marcador = true;
159 }
160
161 if (numFrames > 200 && raio[numFrames] < raio[numFrames - 1] &&
    raio[numFrames - 1] < raio[numFrames - 2] && raio[numFrames -
    2] < raio[numFrames - 3] && marcador_latencia == false) {
    /
162     latencia = numFrames;
163     marcador_latencia = true;
164 }
165
166
167 arquivo << raio[numFrames] << "\t" << velocidade[numFrames] << "\
    t" << aceleracao[numFrames] << endl;
168
169 }
170 else {
171     cout << "\nRaio nao encontrado no frame " << numFrames << ".";
172     arquivo << endl;
173 }
174
175 waitKey(Delay);
176 ui->progressBar->setMaximum(NumeroFrames);
177 ui->progressBar->setValue(numFrames);
178 ui->progressBar->update();
179 }
```

```
180
181 amplitude = raio_max - raio_min;
182 veloc_media = amplitude / (quadro_raio_max - quadro_raio_min);
183 acel_media = veloc_media / (quadro_raio_max - quadro_raio_min);
184 cout << "\n\n\nProcessamento finalizado. Tempo total de
      processamento: " << (getTickCount() - inicio) * 1000 /
      getTickFrequency() << "ms"<< endl;
185 cout << endl << "\n----Parametros pupilometria----" << endl << "
      Raio Max: " << raio_max << "px; \nRaio Min: " << raio_min << "
      px; \nAmplitude: " << amplitude << "px; \nQuadro com raio
      Minimo: "
186 << quadro_raio_min << "; \nQuadro onde ocorre acomodacao: " <<
      quadro_acomodacao << "; \nLatencia: frame Num " << latencia <<
      "; \nVelocidade media constricao: " << veloc_media << "px/
      quadro; \nAceleracao media constricao: " << acel_media << "px/
      quadro^2;" << endl;
187 arquivo << endl << "----Parametros pupilometria---- " << endl <<
      "Raio Max: " << raio_max << "px; \nRaio Min: " << raio_min << "
      px; \nAmplitude: " << amplitude << "px; \nQuadro com raio
      Minimo: "
188 << quadro_raio_min << "; \nQuadro onde ocorre acomodacao: " <<
      quadro_acomodacao << "; \nLatencia: frame Num " << latencia <<
      "; \nVelocidade media constricao: " << veloc_media << "px/
      quadro; \nAceleracao media constricao: " << acel_media << "px/
      quadro^2;" << endl;
189
190 if (raio[conta1] < raio_max) {
191 cout << "Acomodacao atingiu" << raio[conta1] * 100 / raio_max <<
      "por cento do diametro inicial da pupila" << endl;
192 arquivo << "Acomodacao atingiu " << raio[conta1] * 100 / raio_max
      << " por cento do diametro inicial da pupila" << endl;
193 }
194
195 waitKey(0);
196 arquivo.close();
197 qApp->quit();
198 QProcess::startDetached(qApp->arguments()[0], qApp->arguments());
199
200
201 }
```

A.3 Código: Interface de Captura para Tela sensível ao toque

```
1
2 #include "mainwindow.h"
3 #include "ui_mainwindow.h"
4 #include <opencv2/opencv.hpp>
5 #include <raspicam/raspicam_cv.h>
6 #include <wiringPi.h>
7 #include <iostream>
8 #include <fstream>
9 #include <unistd.h>
10 #include <opencv2/highgui.hpp>
11 #include <opencv2/imgcodecs.hpp>
12 #include <opencv2/imgproc.hpp>
13 #include <QFileDialog>
14 #include <QMessageBox>
15 #include <raspicam/raspicam.h>
16 #include <QApplication>
17 #include <QProcess>
18 #include <time.h>
19 #include <cstdio>
20 using namespace std;
21 using namespace cv;
22 QString filename;
23 QString filename1;
24 int Aceso=0;
25 int Apagado=0;
26 int Conta1;
27 int count1 = 0;
28 int resolucao1;
29 int resolucao2;
30
31 MainWindow::MainWindow(QWidget *parent) :
32 QMainWindow(parent),
33 ui(new Ui::MainWindow)
34 {
35 ui->setupUi(this);
36 QWidget::showMaximized();
37 }
38
39 MainWindow::~MainWindow()
40 {
```

```
41 delete ui;
42 }
43
44
45 void MainWindow::on_pushButton_clicked()
46 {
47
48
49 time_t rawtime;
50 struct tm * timeinfo;
51 char buffer [80];
52 char buffer1[80];
53
54
55 time (&rawtime);
56 timeinfo = localtime (&rawtime);
57
58 strftime (buffer,80,"/home/pi/Videos/Hora-%H-%M-Data-%d-%m-%y.txt
    ",timeinfo);
59 strftime (buffer1,80,"/home/pi/Videos/Hora-%H-%M-Data-%d-%m-%y.
    avi",timeinfo);
60 puts(buffer);
61 puts(buffer1);
62
63
64 Aceso= ui->spinBox->value()+200;
65 Apagado= (ui->spinBox_2->value());
66 if (ui->comboBox->currentIndex()==0)
67 {
68   resolucao1= 320;
69   resolucao2=240;
70
71 }
72 if(ui->comboBox->currentIndex()==1){
73   resolucao1=640;
74   resolucao2=480;}
75
76
77
78
79
80 Mat FrameAtual, suave_image;
```

```
81
82
83
84 ofstream arquivo;
85
86 raspicam::RaspiCam_Cv capture;
87 capture.set(CV_CAP_PROP_FORMAT, CV_8UC1 );
88 capture.set( CV_CAP_PROP_FPS, 60 );
89 capture.set ( CV_CAP_PROP_FRAME_WIDTH, resolucao1);
90 capture.set ( CV_CAP_PROP_FRAME_HEIGHT, resolucao2 );
91 capture.set( CV_CAP_PROP_FPS, 30 );
92
93
94 if(!capture.open()) {cerr<<"Erro ao abrir camera\n";printf ("-1")
    ;}
95 usleep(500000);
96
97 bool stop(false);
98
99 wiringPiSetup();
100 pinMode(0, OUTPUT);
101 digitalWrite(0,LOW);
102
103 double FrameRate = capture.get(CV_CAP_PROP_FPS);
104 cout<<"\nFPS camera: "<<FrameRate;
105
106 arquivo.open(buffer);
107
108
109
110
111
112 Size S = Size(resolucao1,resolucao2);
113 int fourcc = CV_FOURCC('H','2','6','4');
114 VideoWriter saida_Video;
115 saida_Video.open(buffer1, fourcc, 12, S, false);
116 double time=cv::getTickCount();
117 Conta1= Aceso+ Apagado;
118
119 ui->progressBar->setMaximum(Conta1);
120 while (count1<Conta1) {
121
```

```
122 capture.grab();
123 capture.retrieve(FrameAtual);
124
125 saida_Video.write(FrameAtual);
126 count1++;
127
128 ui->progressBar->setValue(count1);
129 ui->progressBar->update();
130
131
132 if(count1==200)
133     digitalWrite(0,HIGH);
134
135
136 if(count1==Aceso)
137     digitalWrite(0,LOW);
138
139
140 cout<<"\n"<<count1<<"    "<<cv::getTickCount() /double ( cv::
        getTickFrequency());
141 arquivo<<count1<<"\t"<<cv::getTickCount() /double ( cv::
        getTickFrequency())<<endl;
142
143 }
144
145 double secondsElapsed= double ( cv::getTickCount()-time ) /double
        ( cv::getTickFrequency() ); //tempo em segundos
146 cout<<endl<< secondsElapsed<<" segundos para capturar "<<count1<<
        " frames : FPS = "<< ( float ) ( ( float ) ( count1) /
        secondsElapsed ) <<endl;
147
148 arquivo.close();
149
150 qApp->quit();
151 QProcess::startDetached(qApp->arguments()[0], qApp->arguments());
152
153
154
155
156 }
157
158
```

```
159
160
161 void MainWindow::on_pushButton_2_clicked()
162 {
163     filename = QFileDialog::getSaveFileName(
164
165     this,
166     tr("Open File"),
167     "/home/pi",
168     "Video File (*.avi);;"
169
170 );
171 }
172
173
174 void MainWindow::on_pushButton_3_clicked()
175 {
176
177
178     filename1 = QFileDialog::getSaveFileName(
179
180     this,
181     tr("Open File"),
182     "/home/pi",
183     tr(" Text File (*.txt);;"
184
185 );
186
187
188 }
```

A.4 Código: Interface de Análise para Tela sensível ao toque

```
1
2 #include "mainwindow.h"
3 #include "ui_mainwindow.h"
4 #include <opencv2/opencv.hpp>
5 #include <iostream>
6 #include <fstream>
7 #include <QFileDialog>
8 #include <QMessageBox>
9 #include <time.h>
```

```
10 #include <cstdio>
11 #include <QApplication>
12 #include <QProcess>
13
14 QString filename;
15 QString filename1;
16 int suav;
17 int raiomin;
18 int raiomax;
19 int bincanny;
20 using namespace std;
21 using namespace cv;
22
23 MainWindow::MainWindow(QWidget *parent) :
24 QMainWindow(parent),
25 ui(new Ui::MainWindow)
26 {
27 ui->setupUi(this);
28 }
29
30 MainWindow::~MainWindow()
31 {
32 delete ui;
33 }
34
35
36
37
38 void MainWindow::on_pushButton_2_clicked()
39 {
40 filename = QFileDialog::getOpenFileName(
41
42 this,
43 tr("Open File"),
44 "/home/pi",
45 "Video File (*.avi);;"
46
47 );
48
49 if (filename != NULL)
50 { QMessageBox::information(this, tr("File Name"), filename);
51 }
```



```
52 else
53 QMessageBox::information(this, tr("File Name"), "Nenhum arquivo
    selecionado");
54 }
55
56 void MainWindow::on_pushButton_clicked()
57 {
58
59 suav= ui->spinBox->value();
60 raiomin= ui->spinBox_2->value();
61 raiomax= ui->spinBox_3->value();
62 bincanny= ui->spinBox_4->value();
63
64
65 const int64 inicio = getTickCount();
66 std::string str(filename.toLatin1().data());
67 Mat FrameAtual, suave_image;
68 int numFrames = 0;
69 vector<Vec3f> circles;
70 ofstream arquivo;
71 double raio_max = 0;
72 double raio_min = 50;
73 double latencia, amplitude, quadro_raio_min, quadro_raio_max,
    quadro_acomodacao, veloc_media, acel_media;
74 bool marcador=false;
75 bool marcador_latencia = false;
76 time_t rawtime;
77 struct tm * timeinfo;
78 char buffer [80];
79
80
81 time (&rawtime);
82 timeinfo = localtime (&rawtime);
83
84 strftime (buffer,80, "/home/pi/Analises/Hora-%H-%M-Data-%d-%m-%y.
    txt",timeinfo);
85 puts(buffer);
86
87
88
89 VideoCapture capture(str);
90 if (!capture.isOpened())
```

```
91 printf ("1");
92
93 bool stop(false);
94
95 double FrameRate = capture.get(CV_CAP_PROP_FPS);
96 int NumeroFrames = capture.get(CV_CAP_PROP_FRAME_COUNT);
97 int conta1= NumeroFrames -1;
98 int Delay = 1000 / 22;
99 double raio[NumeroFrames],posicaoX[NumeroFrames], posicaoY[
    NumeroFrames], velocidade[NumeroFrames], aceleracao[
    NumeroFrames];
100
101
102 arquivo.open(buffer);
103
104 while (!stop) {
105     if (!capture.read(FrameAtual))
106         break;
107     ++numFrames;
108
109     cvtColor(FrameAtual, FrameAtual, CV_BGR2GRAY);
110
111     medianBlur(FrameAtual, suave_image, suav);
112
113
114
115     HoughCircles(suave_image, circles, CV_HOUGH_GRADIENT, 3,
        FrameAtual.rows / 1.5, bincanny, 50, raioMin, raioMax);
116
117
118     vector<Vec3f>::const_iterator itc = circles.begin();
119
120     cvtColor(FrameAtual, FrameAtual, CV_GRAY2BGR);
121
122     while (itc != circles.end()) {
123         circle(FrameAtual, Point((*itc)[0], (*itc)[1]), (*itc)[2], Scalar
            (0, 255, 255), 1);
124         line(FrameAtual, Point((*itc)[0] - 20, (*itc)[1]), Point((*itc)
            [0] + 20, (*itc)[1]), Scalar(0, 0, 200), 1, 8);
125         line(FrameAtual, Point((*itc)[0], (*itc)[1] - 20), Point((*itc)
            [0], (*itc)[1] + 20), Scalar(0, 0, 200), 1, 8);
126         ++itc;
```

```
127 }
128
129
130 if (!circles.empty()) {
131 cout << "\nRaio encontrado no frame " << numFrames << ": " <<
    circles[0][2] << " na posicao " << circles[0][0] << "," <<
    circles[0][1];
132 raio[numFrames] = circles[0][2];
133 posicaoX[numFrames] = circles[0][0];
134 posicaoY[numFrames] = circles[0][1];
135
136 if (numFrames > 2 && raio[numFrames] - raio[numFrames - 1] > 10)
137 raio[numFrames] = raio[numFrames - 1];
138 velocidade[numFrames] = (raio[numFrames] - raio[numFrames - 1]);
139 aceleracao[numFrames] = (velocidade[numFrames] - velocidade[
    numFrames - 1]);
140
141 if (numFrames < 250 && raio_max < circles[0][2]) {
142 raio_max = circles[0][2];
143 quadro_raio_max = numFrames;
144 }
145 if (numFrames > 200 && raio_min > circles[0][2]) {
146 quadro_raio_min = numFrames;
147 }
148 if (numFrames > 340 && abs(circles[0][2] - raio_max) < 2 && marcador
    == false) {
149 quadro_acomodacao = numFrames;
150 marcador = true;
151 }
152
153 if (numFrames > 200 && raio[numFrames] < raio[numFrames - 1] &&
    raio[numFrames - 1] < raio[numFrames - 2] && raio[numFrames -
    2] < raio[numFrames - 3] && marcador_latencia == false) {
154 latencia = numFrames;
155 marcador_latencia = true;
156 }
157
158
159 arquivo << raio[numFrames] << "\t" << velocidade[numFrames] << "\
    t" << aceleracao[numFrames] << endl;
160
161 }
```

```
162 else {
163     cout << "\nRaio nao encontrado no frame " << numFrames << ".";
164     arquivo << endl;
165 }
166
167
168 waitKey(Delay);
169 ui->progressBar->setMaximum(NumeroFrames);
170 ui->progressBar->setValue(numFrames);
171 ui->progressBar->update();
172 }
173
174 amplitude = raio_max - raio_min;
175 veloc_media = amplitude / (quadro_raio_max - quadro_raio_min);
176 acel_media = veloc_media / (quadro_raio_max - quadro_raio_min);
177 cout << "\n\n\nProcessamento finalizado. Tempo total de
        processamento: " << (getTickCount() - inicio) * 1000 /
        getTickFrequency() << "ms"<< endl;
178 cout << endl << "\n----Parametros pupilometria----" << endl << "
        Raio Max: " << raio_max << "px; \nRaio Min: " << raio_min << "
        px; \nAmplitude: " << amplitude << "px; \nQuadro com raio
        Minimo: "
179 << quadro_raio_min << "; \nQuadro onde ocorre acomodacao: " <<
        quadro_acomodacao << "; \nLatencia: frame Num " << latencia <<
        "; \nVelocidade media constricao: " << veloc_media << "px/
        quadro; \nAceleracao media constricao: " << acel_media << "px/
        quadro^2;" << endl;
180 arquivo << endl << "----Parametros pupilometria---- " << endl <<
        "Raio Max: " << raio_max << "px; \nRaio Min: " << raio_min << "
        px; \nAmplitude: " << amplitude << "px; \nQuadro com raio
        Minimo: "
181 << quadro_raio_min << "; \nQuadro onde ocorre acomodacao: " <<
        quadro_acomodacao << "; \nLatencia: frame Num " << latencia <<
        "; \nVelocidade media constricao: " << veloc_media << "px/
        quadro; \nAceleracao media constricao: " << acel_media << "px/
        quadro^2;" << endl;
182
183 if (raio[conta1] < raio_max) {
184     cout << "Acomodacao atingiu" << raio[conta1] * 100 / raio_max <<
        "por cento do diametro inicial da pupila" << endl;
185     arquivo << "Acomodacao atingiu " << raio[conta1] * 100 / raio_max
        << " por cento do diametro inicial da pupila" << endl;
```

```
186 }  
187  
188  
189 waitKey(0);  
190 arquivo.close();  
191 qApp->quit();  
192 QProcess::startDetached(qApp->arguments()[0], qApp->arguments());  
193  
194  
195 }
```
