UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA ELÉTRICA
PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

DESENVOLVIMENTO DE UM SISTEMA EMBARCADO PARA PUPILOMETRIA

Rafael Augusto da Silva

Uberlândia, MG

2016

DESENVOLVIMENTO DE UM SISTEMA EMBARCADO PARA PUPILOMETRIA

Rafael Augusto da Silva

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Uberlândia, como requisito parcial para obtenção do título de Mestre em Ciências - Área de concentração: Processamento Digital de Sinais.
Aprovada em 29 de Fevereiro de 2016.

Banca examinadora:

Antônio C. P. Veiga, Dr - Orientador (UFU)
Luciano Xavier Medeiros, Dr (UFTM)
Gilberto Arantes Carrijo, PhD (UFU)

DESENVOLVIMENTO DE UM SISTEMA EMBARCADO PARA PUPILOMETRIA

Rafael Augusto da Silva

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Uberlândia, como requisito parcial para obtenção do título de Mestre em Ciências - Área de concentração: Processamento Digital de Sinais.

Prof. Dr. Antônio C. P. Veiga Orientador	Prof. Dr. Darizon A. Andrade Coordenador do curso de Pós-Graduação

Uberlândia, MG

2016

Dados Internacionais de Catalogação na Publicação (CIP)
Sistema de Bibliotecas da UFU, MG, Brasil.

S586d 2016

Silva, Rafael Augusto da, 1990-
Desenvolvimento de um sistema embarcado para pupilometria / Rafael Augusto da Silva. - 2016.
114 f. : il.

Orientador: Antônio Cláudio Paschoarelli Veiga.
Dissertação (mestrado) - Universidade Federal de Uberlândia, Programa de Pós-Graduação em Engenharia Elétrica.
Inclui bibliografia.

1. Engenharia elétrica - Teses. 2. Pupilometria - Teses. 3. Processamento de Sinais - Teses. 4. Processamento de imagens - Teses. 5. Visão por computador - Teses. I. Veiga, Antônio Cláudio Paschoarelli, 1963-. II. Universidade Federal de Uberlândia. Programa de Pós-Graduação em Engenharia Elétrica. III. Título.

CDU: 621.3

A ciência realmente só tem alcançado tornar
mais intensa e forte uma certeza:
- a velha certeza socrática da nossa irreparável ignorância.
De cada vez sabemos mais - que não sabemos nada.

Eça de Queiroz

Agradecimentos

Ao Programa de Pós-Graduação em Engenharia Elétrica, por possibilitar a realização de excelentes trabalhos de pesquisa. Em especial à Cinara pela paciência e pronto auxílio.

Aos meus pais, Romes e Maria Angélica, pelo amor incondicional, apoio e incentivo que sempre ofereceram à nossa família. Obrigado pelo exemplo!

Aos meus irmãos, Romes e Renato, com quem eu posso sempre contar.

À minha avó Aranista, pelo imenso carinho e presença constante.

Ao meu orientador Prof. Dr. Paschoarelli, pelo apoio, paciência e confiança.

Aos meus colegas e amigos de pós-graduação. Ao Cláriton Bernadelli e Alessandro Gontijo pelo apoio e conselhos que tornaram possível a realização deste trabalho. Ao Tiaguinho, grande companheiro de laboratório.

Aos meus amigos de longa data que me dão coragem para continuar sempre.

À FAPEMIG pelo apoio financeiro a este trabalho e projetos relacionados.

À DEUS, por tornar possível todas as conquistas.

Resumo

da Silva, R. A. **&** ***Veiga***, A. C. P. Desenvolvimento de um sistema embarcado para pupilometria.
Processamento de Imagens - Video Tracking, FEELT-UFU, Uberlândia,
2016, 113p.

Devido a 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.

Este trabalho apresenta o desenvolvimento de um algoritmo de processamento de imagens capaz de efetuar o rastreamento pulilar dinâmico em tempo real, utilizando linguagens de código aberto, plataformas de desenvolvimento e bibliotecas de uso gratuito. São sugeridos aperfeiçoamentos nas etapas de processamento de imagens que resultam obtenção de bom desempenho da aplicação, voltada a dispositivos genéricos.

Apresenta-se também um protótipo de sistema embarcado de rastreamento das características pupilares através da análise automática de captura em vídeo do olho humano, explorando a possibilidade de execução em tempo real. A concentração 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 e de baixo custo.

O hardware projetado permite a configuração de um estímulo luminoso aplicado em um olho e a fácil obtenção da resposta pupilar associada com acurácia, podendo servir futuramente no auxílio à análise diagnóstica da resposta pupilar em indivíduos, de forma a identificar condições anômalas na saúde e comportamento humanos.

Palavras-chave

Pupilometria, Transformada de Hough, Visão Computacional, Processamento de Sinais, Processamento de Imagens, Processamento de Vídeo.

Abstract

da Silva, R. A. **&** ***Veiga***, A. C. P. Development of embedded system for pupillometry.
Image Processing - Video Tracking, FEELT-UFU, Uberlândia,
2016, 113p.

Related to clinical interests regarding the mapping of the eye pupil and its characteristics, dynamic pupillometry consists of the measurement of the eye's responses to luminous stimuli and has been the object of recent scientific discoveries, including the development of automatic systems capable of providing real time results.

This research presents the development of an algorithm destined to process images and capable of instantaneously executing the dynamic measurement of the eye pupil, through the use of open source programming, free access development platforms and libraries.

The project also includes the creation of a embedded system prototype designed to automatically determine the pupil attributes using video captures of the human eye, exploring the possibility of real time usage. The combination of all the stages of the system in a single device presents advantages like portability and versatility, along to the low cost of development obtained by the use of open source, low priced tools.

The adopted hardware allows the configuration of a luminous stimulus to be applied in the eye to easily acquire an accurate pupil response, contributing to the future of the diagnostic analysis of the human eye and consequentially identifying undesired health indicators.

Keywords

Pupillometry, Hough Transform, Computer Vision, Signal Processing, Image Processing, Video Processing.

Sumário

Lista de Figuras

Capítulo 1

Introdução

1.1 Introdução

A pupila realiza movimentos de acordo com as condições às quais está submetida. Tais movimentos podem ser contrações (mioses) ou dilatações (midríases) estimuladas pelo sistema nervoso parassimpático e simpático, respectivamente. Fatores como o uso de drogas, frequência respiratória, batidas do coração, patologias, idade, cor da íris, nível de consciência, entre outros fatores, podem causar modificações na dinâmica pupilar.

Devido a 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 [3].

Os movimentos pupilares consistem basicamente em acomodação, reflexo pupilar à luz (PLR) e hippus. Uma variação anormal no movimento de acomodação pode ser resultado de anormalidade no sistema nervoso. 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 [4].

Sistemas de rastreamento classificados como *Active Object Tracking* [5] utilizam sensores ou transmissores para marcar o objeto a ser rastreado e são considerados invasivos,

sendo, portanto, mais adequados para aplicação a sistemas com condições controladas. Dessa forma o rastreamento passivo, método abordado neste trabalho, é geralmente preferível na realização da pupilometria apesar de acarretar em maior complexidade do mesmo.

O rastreamento passivo utiliza alguns critérios de representação de modo a determinar características tais como forma, posição, cor e textura em uma imagem. Torna-se mais complexo em casos em que o objeto de interesse apresenta estrutura deformável e padrões de cor variáveis como na situação de rastreamento da pupila, que reage muito rapidamente a variações na iluminação [6].

Nesse âmbito, existem trabalhos [7, 8] que utilizam métodos diversos, entre eles a Transformada de Hough, de rastreamento da posição pupilar em tempo real, mas cujo foco não está na análise do diâmetro pupilar. Outras publicações existentes efetuam o rastreamento da posição da pupila em tempo real [9, 10] utilizando-a somente para determinar a direção do olhar, não apresentando, entretanto, interesse significativo para o presente trabalho.

1.2 Objetivos deste Trabalho

O objetivo desta dissertação consiste no desenvolvimento de um algoritmo de processamento de imagens capaz de efetuar o rastreamento pulilar dinâmico em tempo real, baseado em aperfeiçoamentos realizados em sistemas já existentes [4, 2] – melhorias na predição de raio pupilar e recorte da imagem proporcionam significativo ganho em tempo de processamento. A otimização de cada etapa do sistema de processamento de imagens - da aquisição ao reconhecimento - deve resultar no bom desempenho da aplicação, voltada a dispositivos genéricos e de capacidade moderada de processamento.

O software resultante será disponibilizado gratuitamente à comunidade científica, possibilitando seu aperfeiçoamento e adaptação a necessidades específicas.

Adicionalmente, objetiva-se desenvolver um protótipo de sistema embarcado de baixo custo para rastreamento das características pupilares através da análise automática de captura em vídeo do olho humano sob estímulos luminosos controlados, explorando a possibilidade de execução em tempo real. Análises comparativas entre a execução do software em um microcomputador e sua versão embarcada serão apresentadas.

O hardware projetado pode servir futuramente no auxílio à análise diagnóstica da resposta pupilar em indivíduos, de forma a identificar condições anômalas na saúde e comportamento humanos.

1.3 Estado da Arte

Os sistemas de análise pupilométrica disponíveis atualmente consistem em sistemas dedicados que utilizam dispositivos de alta precisão e software proprietário, o que resulta no alto custo dessas soluções. A subseção 2.4 apresenta em detalhes alguns dos sistemas recentemente desenvolvidos.

O sistema aqui desenvolvido se diferencia pela utilização de linguagens de código aberto, plataformas de desenvolvimento e bibliotecas de uso gratuito. A escolha dessas ferramentas de software possibilita a alta customização do sistema, facilidade de programação, otimização do algoritmo para melhor desempenho possível, aliado à existência de vasto material de suporte disponível e custo zero de desenvolvimento.

Outra particularidade do sistema desenvolvido em relação aos sistemas comercializados atualmente é a utilização de um dispositivo comum de captura de imagens, cujo custo é muito inferior ao de sensores industriais de alta precisão. Foi utilizado o Raspberry Pi, um sistema integrado de baixo custo, o qual realiza a execução do código desenvolvido e compilado.

1.4 Estrutura deste Trabalho

No Capítulo 1 são apresentados os objetivos e motivações deste trabalho, seguido do estado da arte e soluções existentes relacionadas ao tema. A descrição da estrutura da dissertação conclui o capítulo.

O Capítulo 2 trata da teoria relacionada à análise dos movimentos pupilares, sua relevância médica e diagnóstica. São expostas algumas das abordagens existentes para análise pupilométrica do PLR.

O terceiro Capítulo (3) aborda o processamento digital de imagens, em teoria e aplicações à engenharia. Os elementos básicos de um sistema de processamento de imagens são

explanados, enfatizando os fundamentos da suavização de imagens, detecção de bordas, localização de formas pela Transformada de Hough e visão computacional. São descritos, por fim, os bancos de dados de íris utilizados nos testes.

O Capítulo 4 compreende a descrição detalhada dos softwares desenvolvidos neste trabalho - frutos da otimização de algoritmos existentes com vistas a obter melhor desempenho em precisão e velocidade de execução. São consideradas as etapas básicas necessárias no processo de desenvolvimento de software, as melhorias sugeridas e desenvolvidas, seguidas dos testes e resultados obtidos após o desenvolvimento dos algoritmos aperfeiçoados.

O Capítulo 5 contém a descrição de um sistema sugerido para aplicação do software desenvolvido em um dispositivo embarcado capaz de processar o fluxo de vídeo proveniente de uma câmera em tempo real. Os testes efetuados e resultados obtidos são também apresentados.

No Capítulo 6 estão presentes a conclusão, as contribuições feitas por esta dissertação e sugestões de possíveis trabalhos futuros relacionados.

Os apêndices contêm os códigos fonte dos programas desenvolvidos para rastreamento pupilar a partir de imagens existentes e a partir de captura simultânea à aplicação de estímulos luminosos.

Capítulo 2

Análise dos Movimentos Pupilares

2.1 Introdução

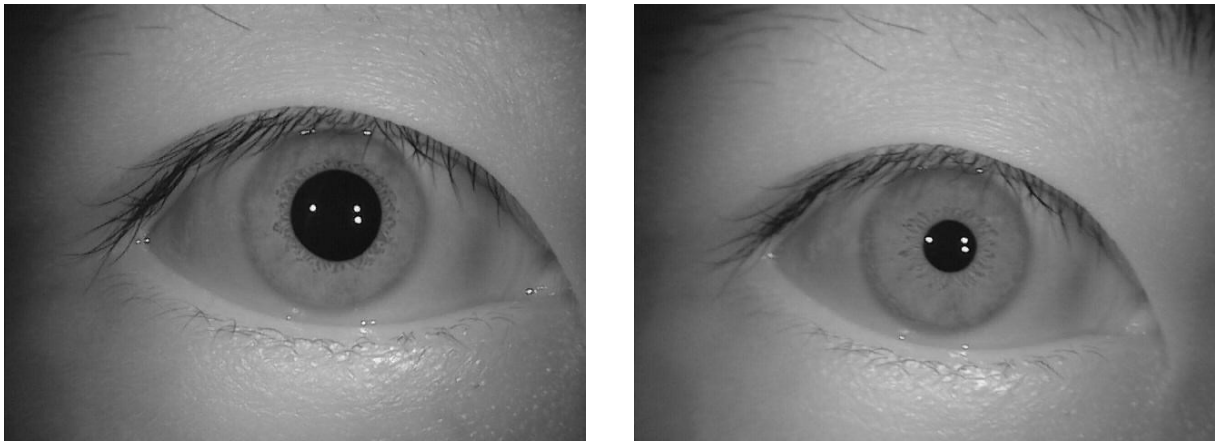
A análise da biometria humana progrediu consideravelmente nos últimos 15 anos. Como os avanços tecnológicos continuam, novas aplicações relacionadas a essa área de estudo aparecem a cada dia. Há uma necessidade crescente de métodos de análise biométrica não-invasivos e desobstrutivos.

O olho funciona de forma a converter luz – que consiste em radiação eletromagnética em diferentes comprimentos de onda – em impulsos nervosos e transmiti-los ao cérebro [11]. Essa conversão é efetuada pela retina, localizada na parte posterior do olho.

Segundo Pamplona [11] a íris forma um disco que possui diâmetro de aproximadamente 12mm. Seu papel é controlar a quantidade de luz que atinge a retina. Em condições de baixa luminosidade a íris se contrai, diminuindo e resultando no aumento da área pupilar. Em caso de alta luminosidade incidente, o processo inverso ocorre, dilatando a íris e contraindo a pupila. O diâmetro pupilar varia de 1,5 mm a 8 mm, em média.

O esfíncter da pupila é um fino músculo orientado à circunferência pupilar que se contrai para diminuir o tamanho pupilar e o músculo dilatador da pupila é um músculo radial que se contrai para aumentar o diâmetro da pupila [4]. Ambos os músculos são independentemente conectados ao sistema nervoso autônomo e se movem separadamente.

Existem modelos que expressam o diâmetro pupilar em função da iluminação inci-



(a) *Dilatação pupilar* (b) *Contração pupilar*

Figura 2.1: *Movimentos pupilares causados pela variação de iluminação incidente.*

dente, representando a adaptação pupilar a alterações abruptas na luz ambiente. Esses modelos matemáticos buscam descrever parâmetros importantes como latência, velocidade de constrição e dilatação, e ainda os movimentos rápidos de baixa amplitude que constituem o *hippus*.

2.2 Pupilometria

A pupilometria dinâmica possui diversas aplicações em vários setores da indústria e representa objeto de estudo, relacionado ao desenvolvimento de técnicas não-invasivas de monitoramento do diâmetro pupilar ao longo do tempo.

Os movimentos pupilares são descritos como: (i) a acomodação, que provoca a mudança do foco pupilar, (ii) o reflexo pupilar à luz (Pupil Light Reflex - PLR) que consiste na adaptação pupilar à quantidade de luz incidente e (iii) o hippus, uma oscilação de baixa amplitude permanente que resulta da convergência gerada pela acomodação e PLR [4].

Determinadas mudanças no diâmetro pupilar podem estar correlacionados a patologias ou determinados estados psicológicos. Soluções abertas ou de baixo custo disponíveis para coleta e análise de dados dessa natureza são, entretanto, praticamente inexistentes.

A reação pupilar à luz (PLR) representa uma das mais interessantes características do olho. Durante esse processo, dois músculos opositores (esfíncter pupilar e dilatador da

pupila) são ativados de forma a efetuar a miose (contração pupilar) e midríase (dilatação pupilar). A variação do tamanho pupilar em resposta a um estímulo luminoso depende, desse modo, do equilíbrio entre os sistemas nervosos simpático (SNS) e parassimpático (PNS) [12]. A PLR serve ainda ao exame de estruturas periféricas do sistema nervoso autônomo ou para avaliar o funcionamento de neurotransmissores no sistema nervoso central.

Os métodos farmacológicos de teste da reatividade pupilar existentes são limitados e seus resultados devem ser considerados com precaução, segundo Fotiou [12]. As deficiências de tais métodos levaram ao desenvolvimento de técnicas como a aplicada neste trabalho, onde a reação à variação luminosa é registrada por uma câmera infravermelho. O processo de análise das características pupilares envolve a utilização de filtros e técnicas de localização e mensuração pupilar em cada quadro da sequência em vídeo gerada no processo.

Os primeiros métodos pupilométricos não farmacológicos foram sugeridos ainda na década de 70 e 80, e consistiam em câmeras de TV comuns que capturavam ciclos pupilares sob iluminação infravermelho [12]. Possuíam custo proibitivo e apresentavam baixa precisão em relação aos sistemas existentes atualmente. O sistema sugerido neste trabalho emprega dispositivos de baixo custo e possibilita a execução automática da análise pupilométrica, tarefa árdua e lenta, previamente efetuada manualmente.

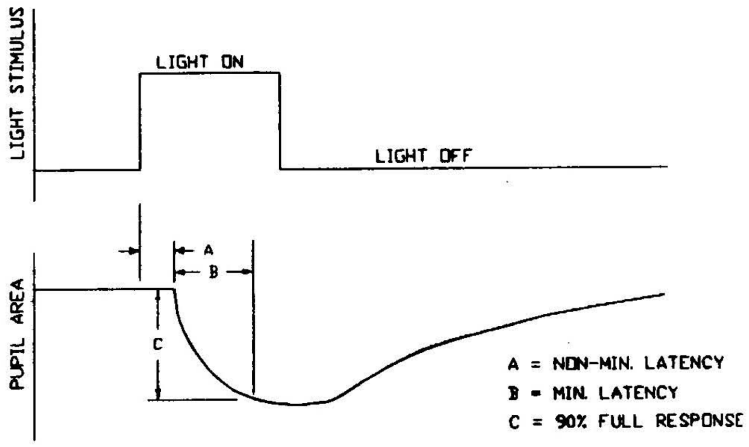


Figura 2.2: *Exemplo de resposta pupilar a um estímulo luminoso.*

A Figura 2.2 , retirada de Link [13], representa o comportamento pupilar ao se aplicar um estímulo luminoso na forma de degrau. Características como latência mínima e as

medidas ilustradas são úteis em aplicações diagnósticas.

A análise pupilométrica resume-se, portanto, no monitoramento pupilar em infravermelho ao se aplicar diversos estímulos luminosos à outra pupila do mesmo indivíduo. Há comprovadamente um reflexo síncrono entre as pupilas dos dois olhos em indivíduos normais [13] - chamado reflexo consensual - e sua falta pode indicar mau funcionamento do sistema visual humano. Como será descrito na subseção seguinte, há evidências suficientes que atestam a relevância da análise pupilométrica em procedimentos médicos.

2.3 Diagnóstico através da Pupilometria

O teste do reflexo pupilar à luz (PLR) já foi utilizado em trabalhos anteriores na investigação de condições como alcoolismo [14], uso de drogas [15, 16], síndrome de Down [17], depressão [18, 19], mal de Alzheimer[18, 20], mal de Parkinson [20, 21], insuficiência cardíaca [22], déficit de atenção, diabetes [23], AIDS [24], autismo [25], entre outros.

O ciclo pupilar pode ser obtido através de estímulos luminosos aplicados diretamente ao olho e consiste no movimento de contração pupilar seguido de expansão e retorno ao estado inicial [4].

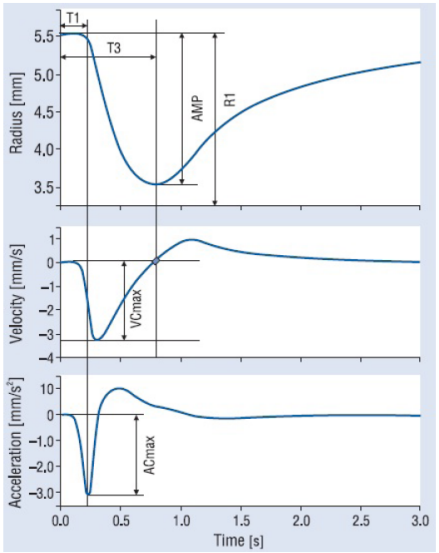


Figura 2.3: *Exemplo de parâmetros medidos em um ciclo pupilar.*

A Figura 2.3 , retirada de [4] representa o ciclo pupilar e alguns dos parâmetros mensurados que podem apresentar interesse diagnóstico, onde têm-se: latência (T1), máxima

velocidade de constrição (VCmax), tempo para a máxima constrição (T3), amplitude (AMP), raio inicial (R1), raio mínimo (R2) e máxima aceleração de constrição (ACmax). Amplitude máxima, latência, velocidade de dilatação e contração, tamanho máximo e mínimo são ainda outros parâmetros considerados.

Trabalhos como [12] procuram estabelecer um procedimento padronizado para o exame da PLR. Este sugere um experimento onde um único flash de duração 30 μs é administrado e um segundo teste consistindo na aplicação de 25 flashes com frequência de 1 Hz é efetuado, seguido da análise da resposta pupilar associada. Nesse caso a curta duração do estímulo luminoso é considerada importante ao procedimento. Diversos parâmetros são registrados, entre eles a razão entre o diâmetro da pupila e íris, a amplitude e latência do menor tamanho pupilar, latência do platô.

Keivanidou por sua vez [22] aplica cinco flashes de luz com resposta retangular, duração de 20 ms e intervalo de 30 s entre eles. Após aferir parâmetros como o raio pupilar após acomodação no escuro, latência da constrição, raio pupilar mínimo, velocidade e aceleração de constrição entre outros foi estabelecida relação entre estes e o diagnóstico de insuficiência cardíaca.

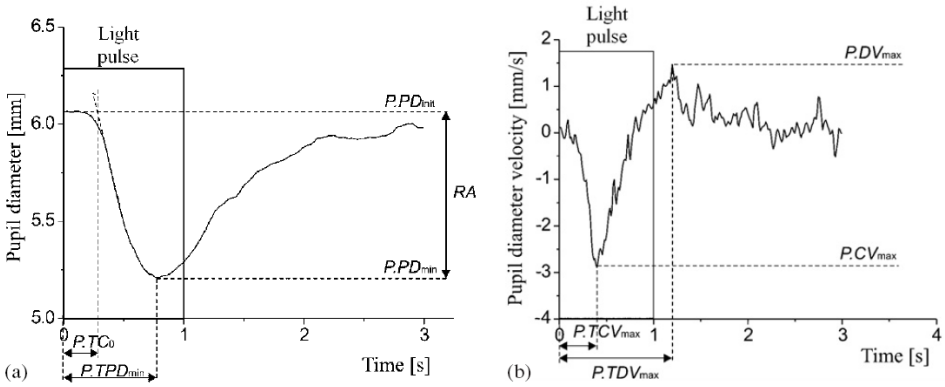


Figura 2.4: *(a) Resposta pupilar a um único pulso luminoso e (b) sua derivada [1].*

De acordo com a maior parte da literatura relacionada, a análise do PLR deve ser efetuada utilizando-se um único pulso luminoso em forma de degrau ou aplicando-se uma sequência periódica de pulsos. Exemplos de respostas pupilares típicas a esses estímulos são exibidas nas Figuras 2.4 e 2.5, retiradas de [1].

Diversos modelos de pupilômetros estão disponíveis no mercado e sua utilidade em diversos procedimentos é evidente. Tais dispositivos são, em sua maioria, soluções proprietárias de alto custo que empregam hardware de alta precisão e são destinados a aplicações

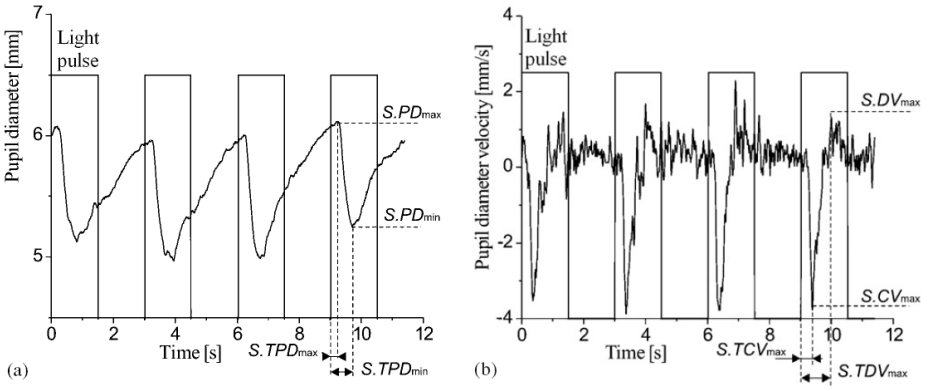


Figura 2.5: *(a) Resposta pupilar a uma sequência de pulsos luminosos e (b) sua derivada.*

específicas. O acesso de consumidores finais a esses equipamentos é geralmente limitado devido a seu preço e disponibilidade.

2.4 Trabalhos Relacionados

Diversas pesquisas voltadas ao desenvolvimento de sistemas pupilométricos foram realizadas nos últimos anos. A maioria deles envolve a utilização de computadores para a execução do algoritmo de localização pupilar, associado ao emprego de sistemas microcontrolados para controle luminoso. Algumas soluções propostas estão listadas a seguir.

Ferrari et al. [26] desenvolveu uma ferramenta capaz de diagnosticar a neuropatia diabética a partir da análise pupilométrica. Seu dispositivo consiste em uma câmera CCD responsável pela captura das imagens oculares em infravermelho, um sistema de sincronização baseado no microcontrolador PIC16F873 e um microcomputador com Pentium IV® responsável por processar as imagens, como ilustrado pela Figura 2.6. A captura das imagens e seu processamento são efetuados separadamente nesse sistema.

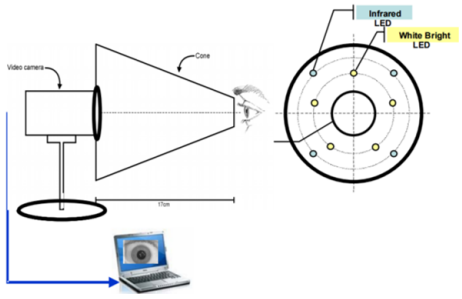


Figura 2.6: *Sistema pupilométrico desenvolvido por Ferrari et al. [26].*

Bernabei et al. [27], por sua vez, implementou um sistema portátil (ilustrado na Figura 2.7) constituído de 3 circuitos principais voltados à captura de imagens e sincronismo dos LEDs responsáveis pelo estímulo luminoso – controle efetuado pelo dispositivo Arduino - e software desenvolvido em LabView® e Matlab® responsável pela interface gráfica e controle dos parâmetros.

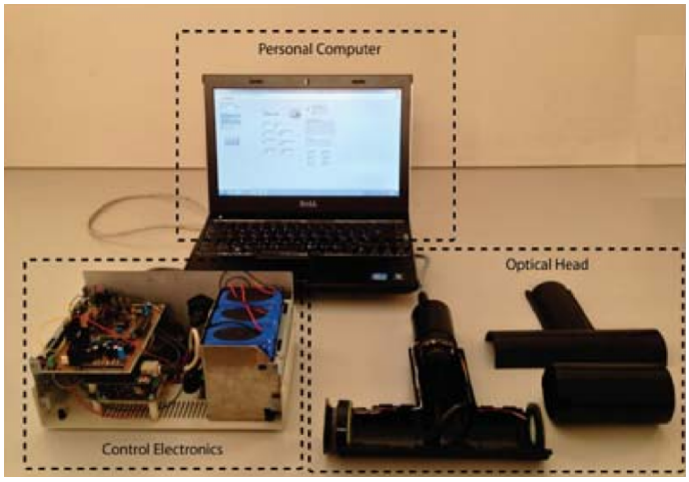


Figura 2.7: *Sistema portátil apresentado por Bernabei et al. [27].*

O alto custo das ferramentas de software e hardware de ponta empregados e a utilização de múltiplos circuitos associados torna o sistema proposto em [27] grande e sujeito a ruídos.

Um sistema baseado em Labview® foi desenvolvido por de Souza [28] para efetuar a pupilometria dinâmica e detecção de piscadas utilizando uma câmera com conexão *FireWire*. Um microcomputador é utilizado em conjunto com uma fonte controlada de iluminação, elementos que podem ser vistos na Figura 2.8. Percebe-se novamente a utilização do software proprietário LabView®, que pode resultar no alto custo da aplicação.

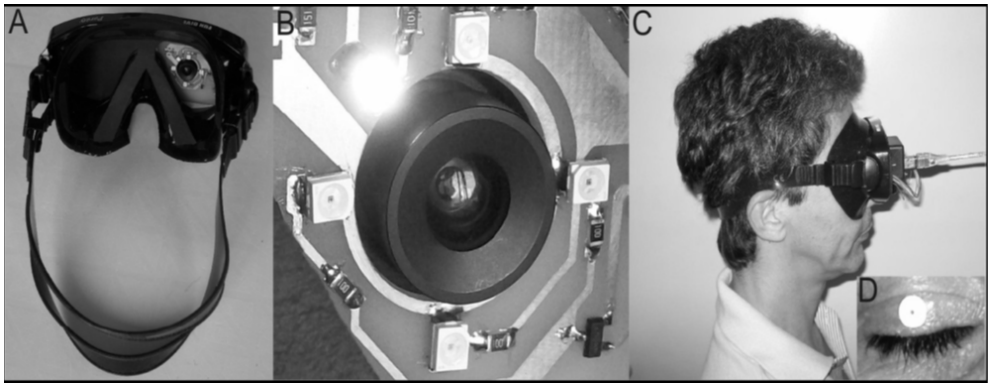


Figura 2.8: *Protótipo desenvolvido por de Souza [28].*

Hovagimian [29] procurou desenvolver uma solução onde um software desenvolvido em Matlab® estima o diâmetro pupilar a partir de imagens capturadas por uma câmera infravermelho exibida na Figura 2.9. Necessita, assim como os sistemas anteriormente descritos, da utilização de um microcomputador.

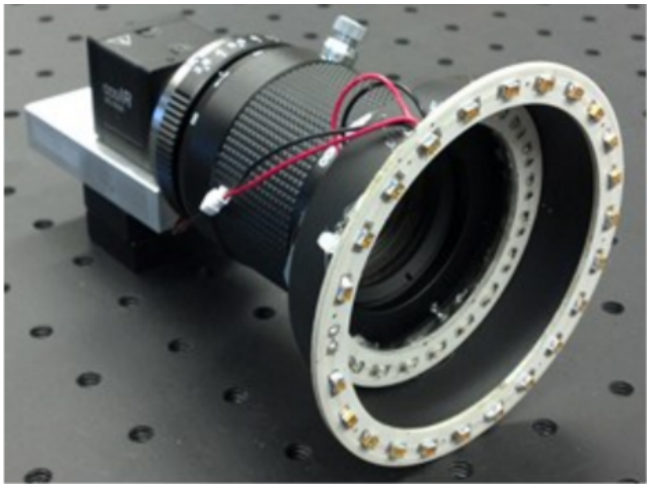
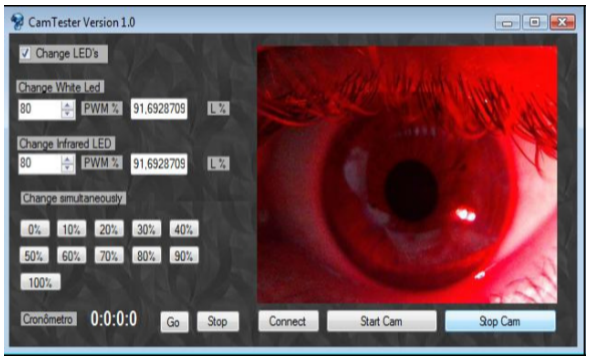


Figura 2.9: *Solução implementada por Hovagimian [29].*

Bernadelli [4] desenvolveu um software em Matlab® capaz de encontrar o diâmetro pupilar a partir de imagens capturadas em um sistema contendo o microcontrolador MSP430 da Texas instruments® [30]. As diversas etapas de tratamento e localização pupilar efetuados na sequência de imagens são efetuados pela rotina desenvolvida em Matlab®. Os elementos do sistema são ilustrados pela Figura 2.10.

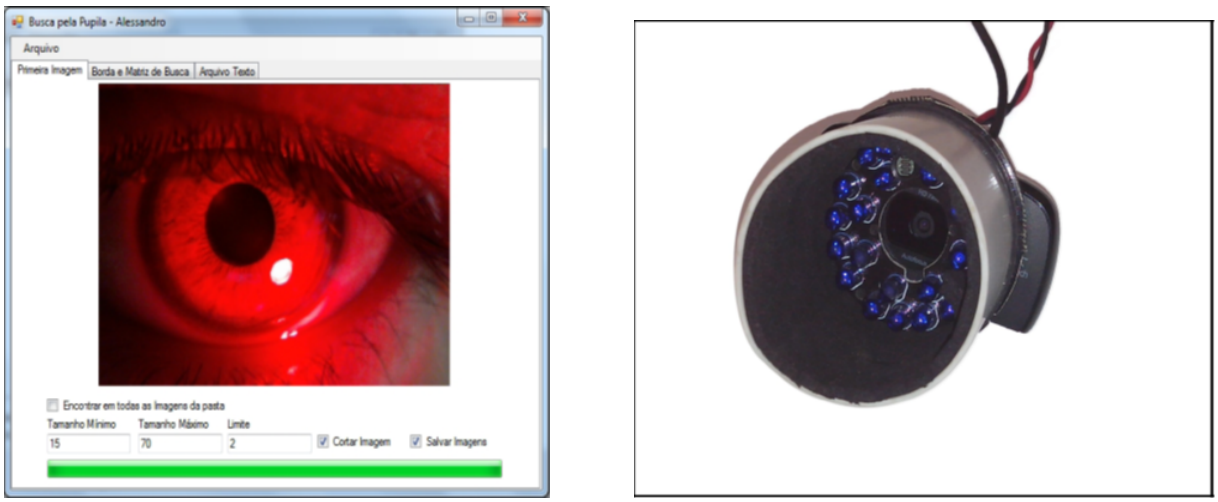


(a) *Interface gráfica de controle dos parâmetros*

(b) *Microcontrolador MSP430 da Texas instruments® utilizado*

Figura 2.10: *Sistema de rastreamento pupilar desenvolvido por Bernadelli [4].*

Uma proposta semelhante foi desenvolvida por Gontijo [2], contendo um software de rastreamento pupilar desenvolvido na linguagem de programação C#®. Possui interface gráfica para ajuste dos parâmetros de localização pupilar e controle do sistema microcontrolado responsável pela iluminação dinâmica utilizada na captura das imagens correspondentes ao ciclo pupilar, como observado na Figura 2.11.



(a) *Interface de software desenvolvido para rastreamento pupilar*

(b) *Protótipo de captura de imagens oculares em infravermelho*

Figura 2.11: *Solução projetada por Gontijo [2] para captura do ciclo pupilométrico.*

Percebe-se que a maioria dos trabalhos relacionados empregam ferramentas proprietárias e requerem a utilização de computadores para efetuar a análise pupilométrica posterior à captura das imagens. O desenvolvimento de uma solução capaz de efetuar todas as etapas da análise pupilométrica do PLR em um único dispositivo pode possibilitar a criação de um equipamento portátil que facilite a realização de procedimentos médicos e diagnósticos que requerem informações pupilométricas derivadas do reflexo pupilar à luz (PLR), constituindo o objetivo principal desta dissertação.

2.5 Considerações Finais

Este Capítulo apresenta características do sistema visual e seu principal órgão, o olho. Descreve os movimentos efetuados pela pupila e íris, assim como sua associação a determinados estados fisiológicos e psicológicos.

O ciclo pupilar e suas características foram representados, assim como os métodos adequados para sua obtenção através do reflexo pupilar à luz – PLR.

Capítulo 3

Processamento de Imagens Digitais

Segundo Gonzalez [31], a visão é sem dúvida o mais desenvolvido sentido humano e as imagens possuem, naturalmente, o papel mais importante na percepção humana. No entanto, o sistema visual humano apresenta limitações como a estreita faixa visível do espectro eletromagnético. Determinadas tarefas que envolvem reconhecimento de padrões ou análises extensivas de imagens representam tarefas dispendiosas e às vezes impossíveis à capacidade perceptiva humana.

Máquinas de captura e análise de imagens, por outro lado, são capazes de manipular imagens provenientes de qualquer faixa do espectro eletromagnético, como ultrassom, microscopia eletrônica, ondas de rádio e imagens geradas por computador [31]. O processamento digital de imagens possibilita a análise desse tipo de informação digital e apresenta, portanto, aplicações em praticamente todos os ramos da ciência e indústria.

Esta seção descreve as características de um sistema genérico de processamento digital de imagens similar ao proposto nesta dissertação, apresentando as principais definições conceituais necessárias à compreensão do mesmo.

3.1 Etapas de um Sistema de Processamento Digital de imagens

Um sistema de processamento digital de imagens manipula imagens adquiridas através de uma câmera eletrônica, similarmente como o cérebro processa as imagens geradas pelos olhos. Possui a capacidade de analisar qualquer forma de dados - não somente imagens provenientes de câmeras, mas também de sensores em virtualmente qualquer faixa do espectro eletromagnético.

A informação proveniente de câmeras ou sensores está na forma digital e é geralmente multidimensional. A captura de imagens à medida que variam com o tempo produz os dados de vídeo, igualmente importantes em inúmeras aplicações na ciência [5]. Os dados digitais de uma imagem correspondem a matrizes multidimensionais que contêm valores numéricos correspondentes aos níveis de intensidade luminosa de cada elemento da imagem, chamado de pixel.

A obtenção de informações úteis a partir de imagens digitais se torna possível através de uma sequência de passos, ilustrada na Figura 3.1 retirada de [2]. As etapas de um sistema genérico de processamento de imagens consistem em aquisição, pré-processamento, segmentação e reconhecimento. São dependentes de uma base de conhecimento que permite a extração de informações relevantes a partir da representação numérica de uma imagem [2].

As subseções seguintes abrangem as etapas de processamento necessárias à aplicação proposta neste trabalho, compreendendo as operações de aquisição, suavização, detecção de bordas, segmentação e reconhecimento.

3.1.1 Aquisição

Antes que o processamento de imagem ou vídeo seja iniciado, é preciso que a imagem seja capturada por uma câmera e convertida em uma matriz de valores numéricos. Esse é o processo conhecido como aquisição de uma imagem [32].

A aquisição de imagens deve ser efetuada de acordo com a aplicação desejada, de modo a satisfazer suas necessidades. Características como espectro, resolução, tempo de

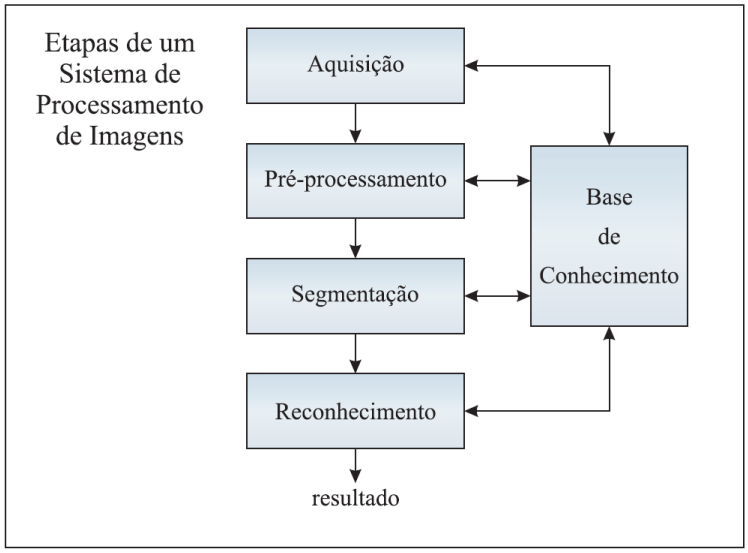


Figura 3.1: *Etapas de um sistema de processamento digital de imagens [2].*

exposição entre outras afetam diretamente a eficácia do processo posterior de análise de uma imagem [2].

A etapa de aquisição de imagens digitais envolve os processos de amostragem e quantização, resultando na criação de matrizes contendo as informações numéricas de nível de intensidade para cada pixel.

3.1.2 Suavização

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

Os filtros de suavização são também conhecidos como filtros passa baixa, uma vez que reduzem as variações nos níveis de cinza ao atenuar as altas frequências, que correspondem aos detalhes de uma imagem.

Os métodos de filtragem aqui utilizados atuam no domínio espacial, ou seja, operam 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.

A operação efetuada 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* [33]. Uma máscara genérica de tamanho 3 x 3 está representada na Figura 3.2:

$$\begin{vmatrix} w_0 & w_1 & w_2 \\ w_3 & w_4 & w_5 \\ w_6 & w_7 & w_8 \end{vmatrix}$$

Figura 3.2: *Máscara de dimensão 3 x 3 e seus coeficientes.*

O tamanho da matriz é escolhido arbitrariamente e é utilizado para se descrever uma máscara. Um filtro de dimensão 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 colocando-se a máscara centrada nele. Os pixels da vizinhança 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 repetido para todos os pixels da imagem a ser processada e corresponde ao processo de convolução da matriz pela imagem a ser transformada [34].

O processo de convolução de uma máscara de dimensão 3 x 3 por uma imagem é ilustrado pela Figura 3.3, onde pode-se observar a substituição do pixel central pelo novo valor calculado através da máscara e a remoção dos pixels da borda na imagem original – desconsiderados por não possuírem vizinhança completa para o cálculo do novo valor de intensidade. Ao atingir 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.

Para calcular o valor de um pixel na posição x, y em uma nova imagem $\mathbf{N}$, a máscara descrita na Figura 3.2 opera em uma imagem original $\mathbf{O}$ de acordo com a seguinte equação:

$$\begin{aligned} N_{x,y} = w_o \times O_{x-1,y-1} + w_1 \times O_{x,y-1} + w_2 \times O_{x+1,y-1} + \\ w_3 \times O_{x-1,y} + w_4 \times O_{x,y} + w_5 \times O_{x+1,y} + \\ w_6 \times O_{x-1,y+1} + w_7 \times O_{x,y+1} + w_8 \times O_{x+1,y+1} \\ \forall x, y \in 2, N-1 \end{aligned} \tag{3.1}$$

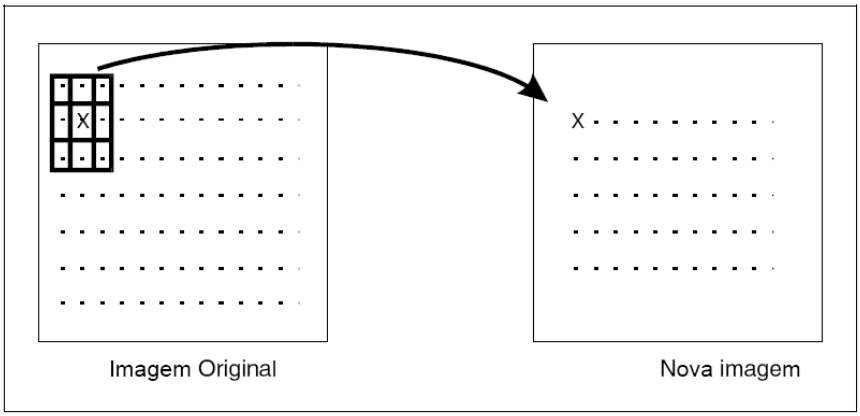


Figura 3.3: *Processo de convolução da máscara pela imagem no processo de filtragem.*

A seguir são apresentados os principais tipos de filtros de suavização presentes na literatura, largamente utilizados no processamento de imagens digitais em diversas aplicações.

Filtro de Média

O filtro de média consiste em uma máscara de filtragem cujos coeficientes possuem todos o mesmo valor. Após a multiplicação dos coeficientes pelo valor de intensidade dos pixels vizinhos, o resultado é dividido pelo tamanho da vizinhança. Este processo corresponde à substituição do pixel central original pela média de intensidade dos pixels vizinhos.

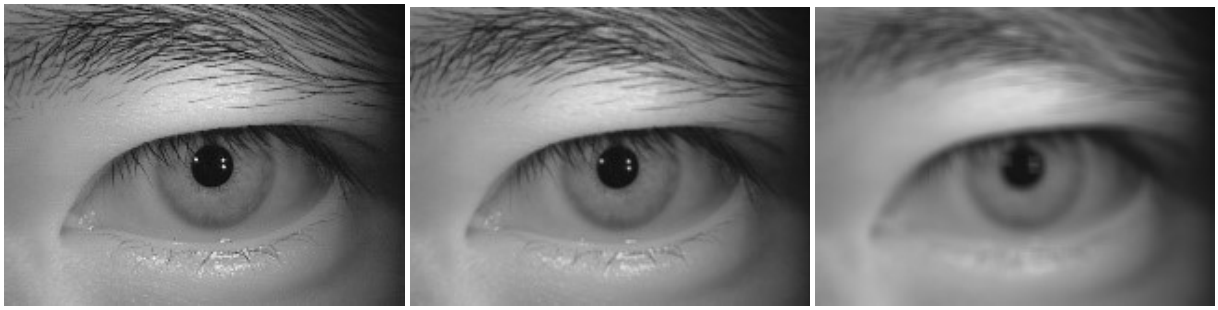
$$\begin{vmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{vmatrix}$$

Figura 3.4: *Máscara correspondente a um filtro de média de dimensão 3x3.*

Este tipo de filtragem é útil na redução de ruído em imagens, porém, reduz também detalhes como informações de borda [34].

Ao se aumentar o tamanho da janela que percorre a imagem, diminui-se banda passante do filtro e amplia-se a influência dos pixels vizinhos no pixel central, o que resulta em um

maior efeito de ”desfocalização”, retendo-se somente as maiores estruturas contidas na imagem. Na Figura 3.5 pode-se observar os resultados obtidos ao se utilizar o filtro de média com diferentes tamanhos de máscara em uma imagem.



(a) *Imagem original* (b) *filtro 5x5* (c) *filtro 17x17*

Figura 3.5: *Filtro de média.*

Maiores janelas utilizadas na filtragem também resultam em maior custo computacional relacionado ao processo, visto que um maior número de coeficientes do *kernel* ocasionam um maior número de operações a serem efetuadas. O aumento no número de operações necessárias se torna mais significativo sobretudo para imagens de maior dimensão.

Filtro Gaussiano

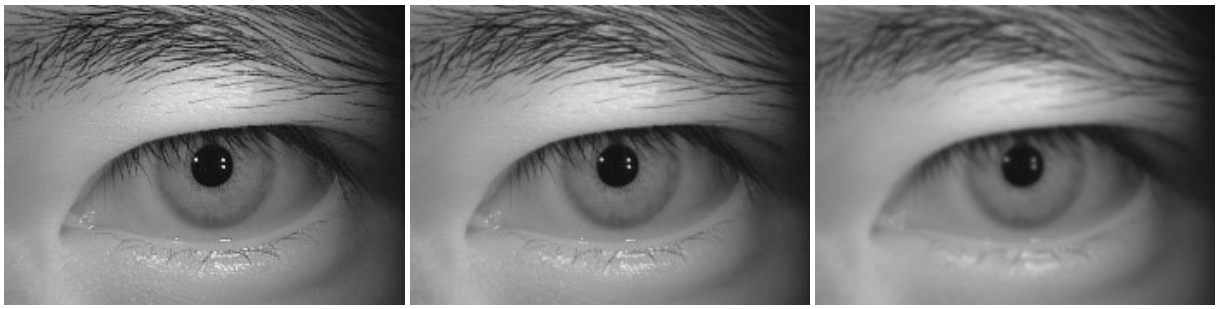
Em certas situações é desejável dar mais importância aos pixels mais próximos na vizinhança de um pixel. Com essa finalidade pode-se calcular uma média ponderada na qual pixels próximos recebem pesos maiores do que os afastados. Isso pode ser feito usando um padrão de pesos que obedeça uma função gaussiana, definida no ponto com coordenadas x, y e controlada pela variância σ de acordo com a equação abaixo:

$$g(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{\left(x^2+y^2\right)}{2\sigma^2}} \quad (3.2)$$

Neste caso, o peso associado a um pixel é proporcional à sua distância do pixel central. O valor de σ (sigma) controla a largura da função gaussiana resultante e, consequentemente, a banda passante do filtro. O valor resultante do pixel central é, portanto, a média ponderada dos pixels da vizinhança.

A função gaussiana possui a forma de um sino e é simétrica, o que a torna uma ferramenta de filtragem particularmente eficaz. É um filtro separável, isto é, a máscara bidimensional de filtragem pode ser decomposta em dois filtros de uma dimensão, resultando em um filtro geralmente mais rápido de ser operado devido ao menor número de multiplicações necessárias no cálculo computacional.

Na Figura 3.6(a) pode-se observar uma imagem ocular, objeto do sistema de rastreamento pupilar, e nas figuras 3.6(b) e 3.6(c) encontra-se o produto de duas filtragens pelo método gaussiano com máscaras de tamanho 5 e 17, respectivamente. Um determinado tamanho de máscara possui valor correspondente de σ assim como a escolha de um valor de σ resulta na determinação do tamanho de máscara que corresponde ao mesmo nível de suavização na filtragem espacial.



(a) *Imagem original* (b) *filtro 5x5* (c) *filtro 17x17*

Figura 3.6: *Filtro gaussiano.*

As características do filtro gaussiano o levaram a ser considerado ótimo para a maioria das situações onde a suavização de imagens é necessária. A retenção de mais características da imagem enquanto remove ruídos o torna preferível em relação ao filtro de média.

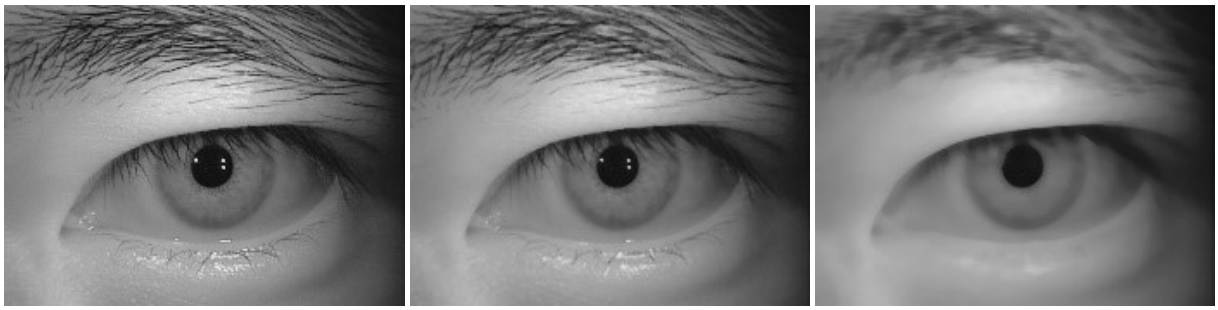
Filtro de Mediana

Os filtros de mediana, por sua vez, substituem o elemento central da máscara pelo valor da mediana das intensidades vizinhas. Ordena, portanto, a intensidade dos pixels dentro da máscara em ordem crescente ou decrescente e alocam ao pixel da posição central da máscara o valor da intensidade do pixel correspondente à posição intermediária do intervalo ordenado [35].

Este tipo de filtro não-linear permite a passagem de grande parte dos detalhes que

possuem altas frequências, sendo todavia, efetivo na remoção de ruídos. Efetua a suavização de uma imagem preservando características de contorno, processo chamado de difusão anisotrópica [34]. Essa é uma das vantagens do operador de mediana sobre o operador gaussiano.

Ao se aumentar o coeficiente de suavização, as regiões com nível de cinza uniforme são bastante suavizadas mantendo-se, entretanto, as bordas evidentes, como mostra a Figura 3.7. Apesar da alta desfocalização dos elementos da imagem, percebe-se claramente o contorno da circunferência pupilar. Carneiro et al. [36] demonstra a eficácia do filtro de mediana em relação ao filtro gaussiano ao se efetuar a segmentação de imagens oculares.



(a) *Imagem original* (b) *filtro 5x5* (c) *filtro 17x17*

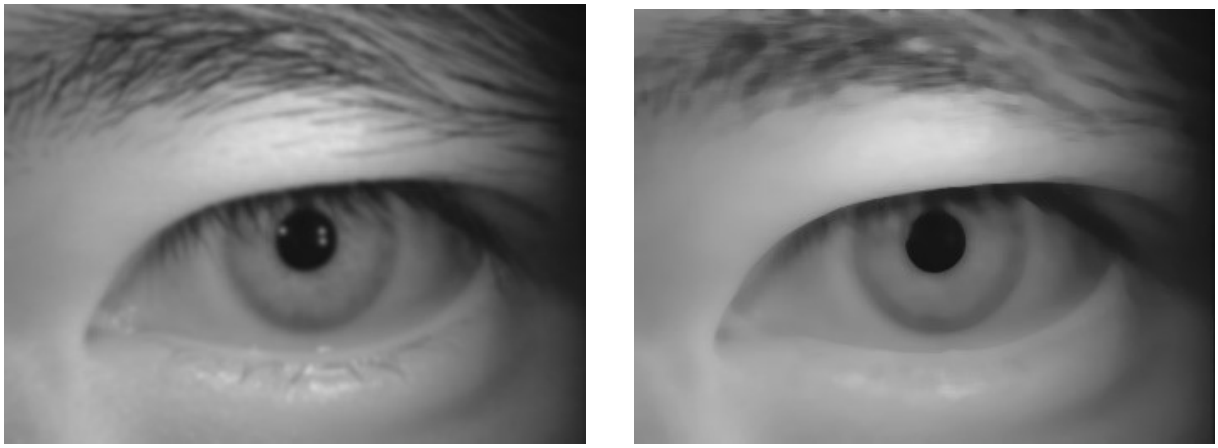
Figura 3.7: *Filtro de mediana.*

Observa-se na Figura 3.8a diferença entre os resultados após a aplicação dos filtros gaussiano e de mediana. Este realça as bordas da pupila podendo ser favorável à localização da mesma, objetivo do algoritmo aqui apresentado.

A desvantagem desse tipo de filtro está no dano causado nas linhas finas e curvas agudas. Pode não ser tão eficaz na redução de ruídos com características gaussianas [34]. Além disso, o processo é computacionalmente mais complexo que a filtragem gaussiana, sendo consequentemente mais lento.

Filtro Bilateral

A filtragem bilateral consiste em uma técnica de suavização de imagens que preserva contornos importantes em uma imagem. Foi desenvolvida a partir de trabalhos sobre filtros gaussianos não-lineares tendo sido utilizada, segundo Paris et al. [37], em contextos diversos no processamento de imagens, como na remoção de ruído, edição de textura e



(a) *Filtro gaussiano* (b) *Filtro de mediana*

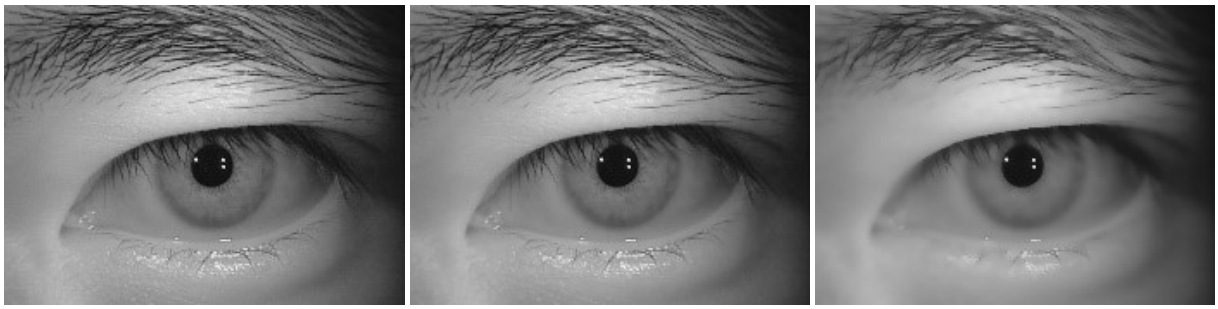
Figura 3.8: *Comparação entre os filtros com máscara de tamanho 17x17.*

iluminação, gerenciamento de tons, estimação de fluxo óptico, entre outros.

Entre suas principais características estão sua simples formulação na qual cada pixel é substituído pela média ponderada de seus vizinhos, sua dependência de somente dois parâmetros que indicam as características a serem preservadas e a capacidade de ser utilizado de maneira não-iterativa.

Assim como os demais filtros aqui apresentados, o filtro bilateral também consiste em uma média ponderada de pixels. Se distingue, entretanto, por considerar também a variação de intensidades para preservar contornos. Dois parâmetros, σ_s e σ_r, definem a intensidade de filtragem de uma imagem.

Na Figura 3.9 pode-se verificar o resultado da filtragem da mesma imagem utilizada para ilustrar os demais filtros, utilizando-se o filtro bilateral com vizinhanças de tamanho 5 x 5 e 17 x 17.



(a) *Imagem original* (b) *filtro 5x5* (c) *filtro 17x17*

Figura 3.9: *Filtro bilateral.*

No processo de filtragem cada vizinho é multiplicado por um peso por uma componente espacial que penaliza pixels distantes e uma componente de variação que penaliza pixels com intensidade diferente. A combinação faz com que somente pixels próximos e similares contribuam para resultado final. Este filtro pode ser utilizado iterativamente para se conseguir imagens do tipo '*cartoon*'. O filtro bilateral apresenta diversas aplicações, e o fundamento da combinação espacial e de intensidade foi explorado em diversas abordagens e aplicações específicas [38]. O método apresenta, no entanto, alto custo computacional, necessitando de técnicas numéricas na aceleração do algoritmo, sobretudo em imagens de maiores dimensões. Pode-se ver pela Figura 3.10, a diferença de resultado entre os métodos de suavização apresentados, com mesmo tamanho de máscara aplicados em uma mesma imagem. A comparação quantitativa de resultados entre tais métodos é apresentada em subseção posterior.

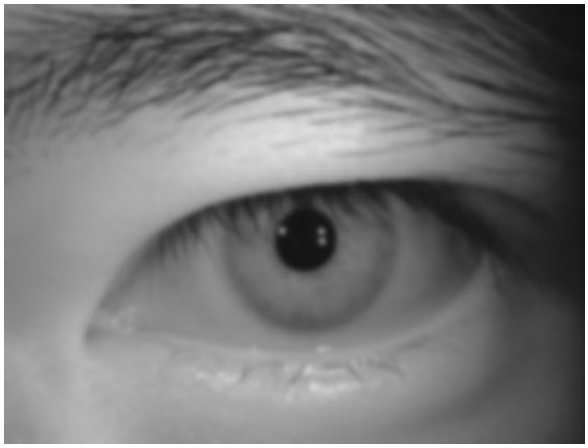
A adequada escolha do filtro de suavização pode ser crucial na eficácia de sistemas de reconhecimento, sobretudo em processos onde seja necessária a detecção de bordas na imagem. A detecção de bordas, apresentada na próxima subseção, é geralmente precedida de suavização da imagem para que se elimine os contornos excessivos que não apresentam relevância ao processo de reconhecimento. A Figura 3.11 ilustra a diferença entre as bordas detectadas em uma imagem após suavização pelos filtros de média, mediana, gaussiana e bilateral.

3.1.3 Detecção de Bordas

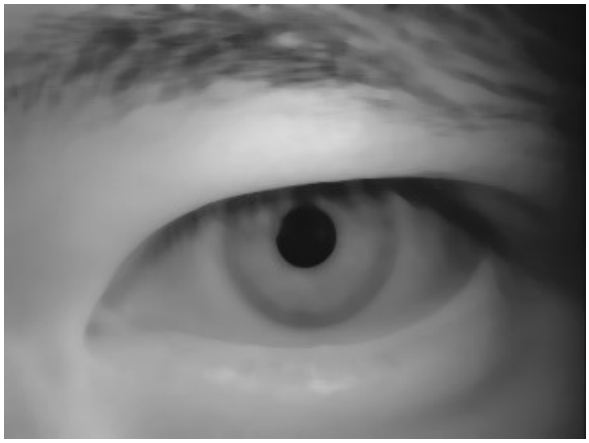
As bordas em uma imagem de interesse caracterizam os contornos nela presentes, apresentando importância fundamental na segmentação e detecção de objetos e padrões. Os pontos que correspondem a bordas em uma imagem possuem variações abruptas de níveis de cinza. Eles correspondem a pontos de transição entre diferentes objetos, de cordo com De Queiroz et al. [39]. Os métodos mais utilizados para detecção de borda são os operadores de Robert, Sobel, Prewitt e Canny, como descreve Dawson-Howe [35]. Após a passagem dos operadores, realiza-se a etapa de limiarização que resulta em uma imagem que contém somente as bordas identificadas pelo método utilizado.

O detector de bordas largamente utilizado em aplicações similares e escolhido neste trabalho é o operador de Canny [40]. Este detector apresenta boa imunidade ao ruído

(a) *Filtro de média*



(b) *Filtro gaussiano*



(c) *Filtro de mediana*

(d) *Filtro bilateral*

Figura 3.10: *Comparação entre os filtros com máscara de tamanho 17x17.*

ao mesmo tempo em que detecta verdadeiros pontos de borda com erro mínimo [34]. O operador de Canny é baseado em três otimizações na detecção de bordas, de acordo com Dawson-Howe [35]:

- 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 produz as vantagens deste detector em relação a outros existentes. Esse detector necessita que a imagem seja previamente suavizada, antes que a direção e valores do

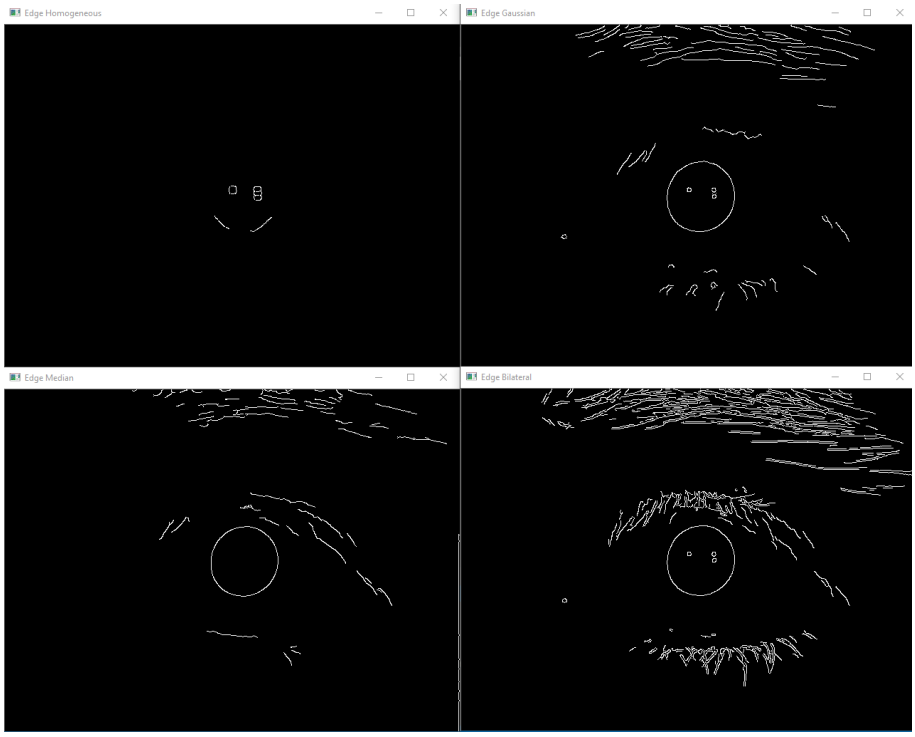


Figura 3.11: *: Detecção de bordas a partir de imagens suavizadas pelos filtros (a) de média, (b) gaussiano, (c) de mediana e (d) bilateral.*

gradiente sejam calculados. O filtro gaussiano é geralmente utilizado para esse fim, porém outros filtros de suavização podem ser preferíveis, dependendo da aplicação prevista.

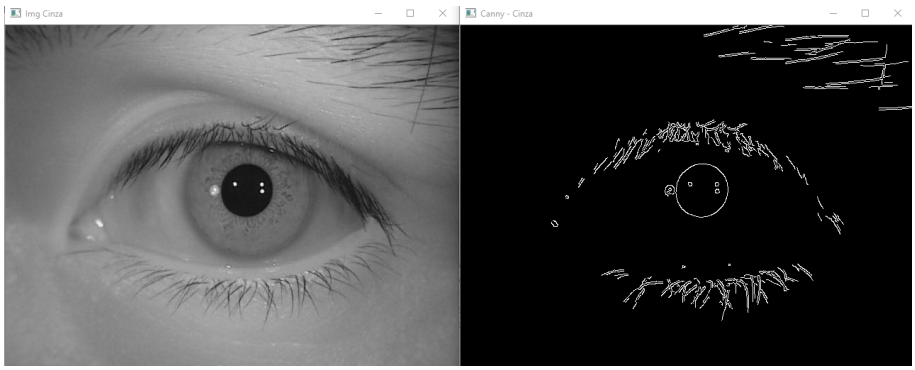


Figura 3.12: *Detecção de bordas através do operador de Canny.*

A Figura 3.12 ilustra a detecção de bordas em uma imagem, sem a aplicação prévia de um filtro de suavização.

3.1.4 Segmentação

Processos de análise de imagens requerem a extração de características, medidas e informações de forma automática ou semi-automática. Informações relevantes geralmente estão localizadas em uma parte específica da imagem que é, portanto, subdividida de forma a otimizar a área de análise em partes ou objetos específicos, assim como descreve De Queiroz [39].

As técnicas de segmentação permitem a identificação de dissimilaridades entre regiões ou objetos de uma imagem, de forma a efetuar a discriminação dos mesmos entre si e em relação ao fundo da imagem (*background*).

Usualmente os algoritmos de segmentação de imagens monocromáticas se baseiam na identificação de descontinuidades e similaridades dos níveis de cinza. A detecção de bordas é elemento geralmente presente nesses métodos. Pode-se empregar ainda outros artifícios como o adotado posteriormente neste trabalho, que pode utilizar informações 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.

Para a presente situação, 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 eficácia do algoritmo, como será abordado na seção de resultados.

3.1.5 Reconhecimento

Uma vez segmentada, a imagem passa à próxima e última etapa de processamento: o reconhecimento de objetos ou regiões na cena. O reconhecimento de padrões é elemento essencial na visão computadorizada e processamento de imagem, como salienta Dawson-Howe [35]. O reconhecimento está presente em aplicações que vão da medicina diagnóstica e biometria à classificação de documentos, sensoriamento remoto, entre outros.

Nesta etapa, pode-se perceber a necessidade dos tratamentos anteriormente efetuados na imagem, sobretudo o processo de segmentação. Se o objeto a ser detectado não está devidamente incluído em uma região de interesse efetiva, não se obterá êxito no reconhecimento ou haverá prejuízos no desempenho do algoritmo [2].

Alguns atributos de um objeto multidimensional que definem padrões são a área, volume, perímetro, superfície, entre outros. Esses podem ser mensurados através da

contagem de pixels. Similarmente, a forma de um objeto pode ser caracterizada a partir se suas bordas. A seleção e extração dos parâmetros corretos representam o problema principal no reconhecimento de padrões.

O objeto alvo das imagens no presente sistema é a pupila. Pretende-se localizá-la a partir do reconhecimento de forma, aproximando o contorno pupilar através de uma circunferência. As ferramentas mais utilizadas capazes de realizar essa tarefa são a Transformada circular de Hough e o operador *Integro-diferencial* de Daugman, além das ferramentas de *template matching* [2].

Melhores resultados em tempo de processamento sem comprometimento da precisão são comprovadamente obtidos através do método da transformada circular de Hough [2]. Este será, por consequência, utilizado neste trabalho.

3.2 Transformada de Hough

A transformada de Hough é utilizada para detectar formas geométricas como linhas, círculos e elipses em imagens digitais. Provavelmente uma das técnicas mais utilizadas na visão computacional [41], apresenta número de citações que ultrapassa largamente operadores clássicos como Sobel e o detector de borda de Canny, como apontado por Hart [42].

Paul Hough depositou, em 1962, uma patente introduzindo um método eficiente de detecção de linhas em imagens binarizadas [43]. A grande ideia explorada pelo cientista consistia na transformação de padrões distribuídos espacialmente na imagem em características compactas dentro de um espaço paramétrico. De acordo com Antolovic [44], se uma forma específica está presente na imagem, o mapeamento de todos os seus pontos no espaço paramétrico fica distribuído em torno dos parâmetros que correspondem à forma.

Dessa maneira, problemas de detecção global no espaço de uma imagem são convertidos em tarefas mais simples de detecção de picos no espaço de parâmetros. O método original de Hough, usualmente designado *Standard Hough Transform* (SHT), descrevia a detecção de linhas. Com o passar dos anos surgiram diversas variações e abordagens para identificação de outras formas analíticas e mesmo irregulares [45].

A Transformada Circular de Hough (CHT) serve à identificação de circunferências

[46]. A Transformada de Hough e suas variantes dependem da conversão prévia das imagens para suas versões binarizadas ou em níveis de cinza. A imagem em níveis de cinza passa pela detecção de bordas preliminar e então é submetida ao método de votação no espaço paramétrico. A CHT, no entanto, se baseia na equação de uma circunferência, apresentando três parâmetros. Tais características resultam em uma maior complexidade computacional em sua operação, demandando mais tempo de processamento e armazenamento em memória, como apontado por Hassanein et al. [45].

A utilização da Transformada de Hough apresenta algumas vantagens. Primeiramente, a transformada considera cada ponto de borda independentemente, o que torna o processamento paralelo dos pontos possível e possibilita aplicações em tempo real [45]. Além disso, o método consegue operar em formas ruidosas ou parcialmente deformadas/degradadas devido à estratégia de votação. A transformada é capaz ainda de detectar múltiplas ocorrências das formas buscadas, uma vez que cada ocorrência é representada em célula específica no espaço paramétrico memória, como salienta Hassanein [45].

Desvantagens existem, entretanto, ao se utilizar a transformada de Hough. Alto custo computacional e capacidade de armazenamento são exigidos. Quanto maior a dimensão onde se deseja operar, maior o número de cálculos necessários. A alta eficiência do algoritmo depende de técnicas de redução na dimensão do acumulador conseguidas através de informações já conhecidas a respeito das formas procuradas.

3.2.1 Representação Paramétrica

A transformada de Hough pode ser descrita sinteticamente como uma transformação de um ponto no plano x,y para o espaço paramétrico. Este é definido de acordo com a forma do objeto de interesse [46].

Uma linha reta passando pelos pontos (x_1, y_1) e (x_2, y_2), por exemplo, pode ser descrita no plano x, y pela Equação 3.3.

$$y = ax + b \tag{3.3}$$

Essa é a representação de uma linha reta no sistema de coordenadas cartesianas, sendo a e b os parâmetros dessa linha. Tal conversão paramétrica é o principal conceito

da transformada de Hough. Esta, no entanto, utiliza uma representação alternativa, uma vez que linhas perpendiculares ao eixo x apresentariam um coeficiente a que tende ao infinito. Os parâmetros efetivamente utilizados na representação de linhas retas pela transformada de Hough são o ângulo θ e o comprimento ρ.

$$\rho = x \sin\theta + y \cos\theta \tag{3.4}$$

A Equação 3.4 descreve a representação de uma linha reta em um espaço paramétrico que possui tamanho finito, dependente da resolução definida para θ. A distância ρ da origem do eixo de coordenadas à reta assume tamanho máximo de duas vezes o comprimento diagonal da imagem, segundo Gonzalez et al. [31].

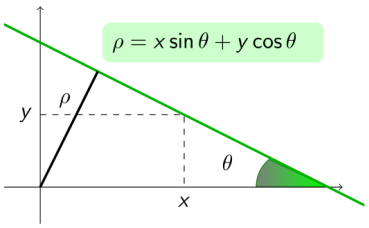


Figura 3.13: *Representação do espaço paramétrico de Hough para linhas.*

Quando a forma de interesse é uma circunferência, utiliza-se diretamente no espaço de Hough – que agora define a chamada transformada circular de Hough – os parâmetros utilizados na equação de um círculo, onde r representa o raio da circunferência, a e b representam o centro do círculo nos eixos x e y, respectivamente. A Equação 3.5 corresponde à representação de uma circunferência.

$$r^2 = (x - a)^2 + (y - b)^2 \tag{3.5}$$

A representação paramétrica de uma circunferência obtida a partir da Equação 3.5 é, portanto:

$$x = a + r \cos\theta \tag{3.6}$$

$$y = b + r \sin\theta \tag{3.7}$$

O espaço paramétrico para linhas é bidimensional e o espaço para representação de círculos é tridimensional. À medida que o número de parâmetros necessários para se representar uma forma aumenta, amplia-se também a dimensionalidade do espaço de Hough correspondente.

3.2.2 Acumulador

O processo de detecção de formas pela transformada de Hough é efetuado através da soma de um acumulador. Também chamado de votação, esse método opera em imagens após a detecção de bordas.

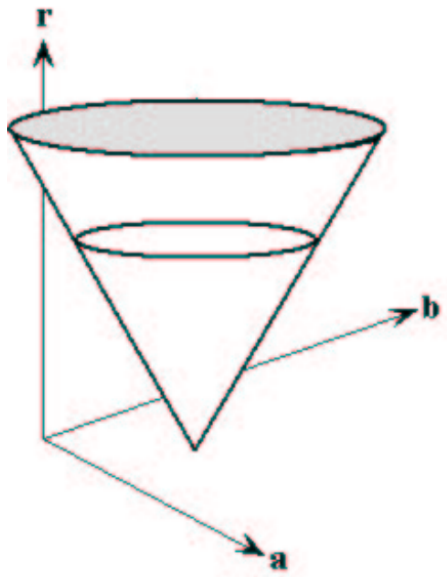


Figura 3.14: *Espaço paramétrico utilizado na transformada circular de Hough.*

Para cada ponto de borda da imagem, desenha-se um círculo no espaço paramétrico com centro no ponto correspondente e raio igual ao procurado. Dessa forma, o eixo x do espaço de Hough corresponde ao parâmetro a, o eixo y corresponde ao parâmetro b e o eixo z representa os valores de raio dentro de uma faixa especificada, como ilustrado na Figura 3.14. Nas coordenadas que correspondem ao perímetro do círculo desenhado, incrementa-se o valor na matriz do acumulador (matriz que possui mesma dimensão do espaço paramétrico).

O processo de votação é ilustrado pela Figura 3.15, onde pode-se observar que cada ponto de borda do círculo gera uma circunferência no espaço paramétrico. As circunfe-

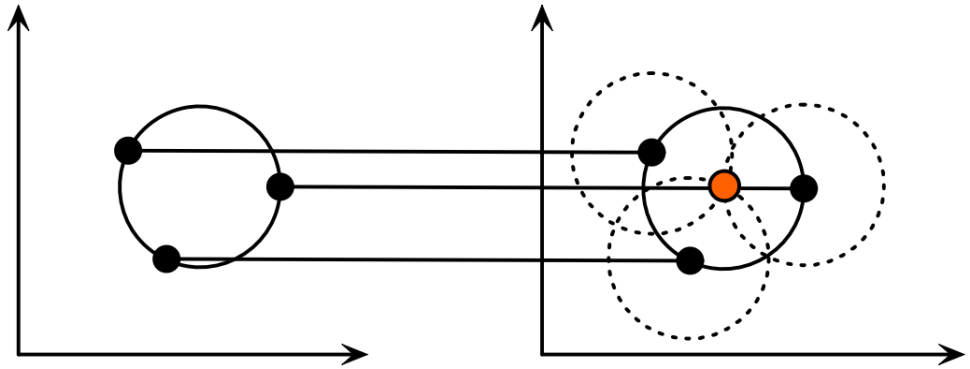
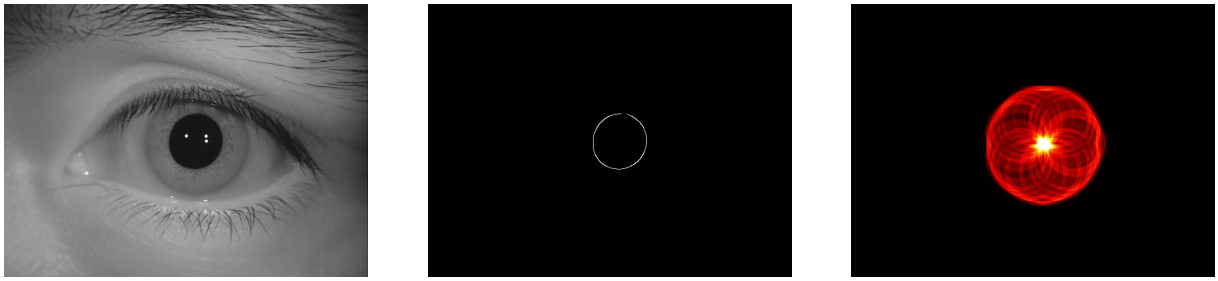


Figura 3.15: *Processo de votação efetuado pela Transformada Circular de Hough.*

rências se cruzam no ponto (a,b) que representa o centro no espaço geométrico.



(a) *Imagem original* (b) *Detecção de bordas* (c) *Espaço de Hough*

Figura 3.16: *Exemplo do espaço de Hough correspondente à imagem de bordas de um olho humano.*

Após efetuar-se 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, ou seja, os pontos com maior valor de acumulador, correspondem aos centros de circunferências localizadas na imagem, como descreve Pedersen [46]. O processo descrito está ilustrado na Figura 3.16, onde pode-se observar o espaço de Hough resultante após a votação efetuada para todos os pixels de borda vistos na Figura 3.16(b).

3.2.3 Localização de Circunferências

A matriz correspondente ao acumulador é tridimensional na transformada circular de Hough. Ao efetuar-se uma busca por circunferências com diferentes valores de raio pode-se facilmente obter matrizes demasiadamente grandes. Os eixos x e y dependem do tamanho

da imagem, e o eixo z da quantidade de valores de raio das circunferências buscadas.

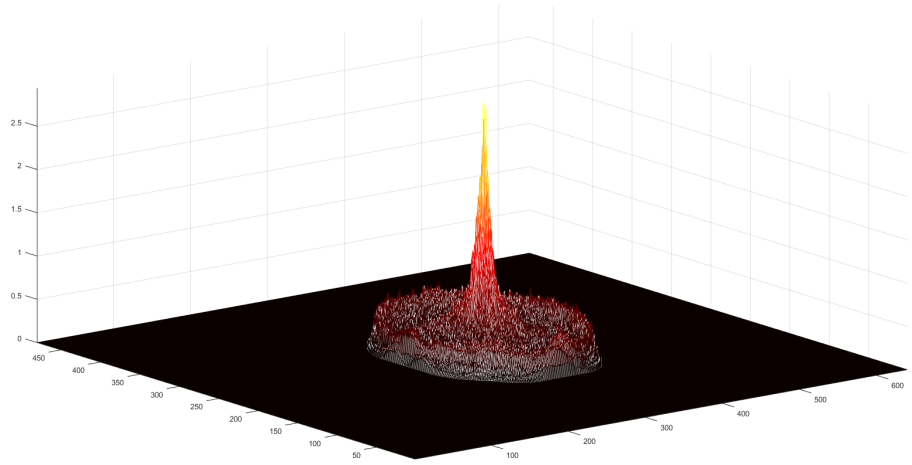
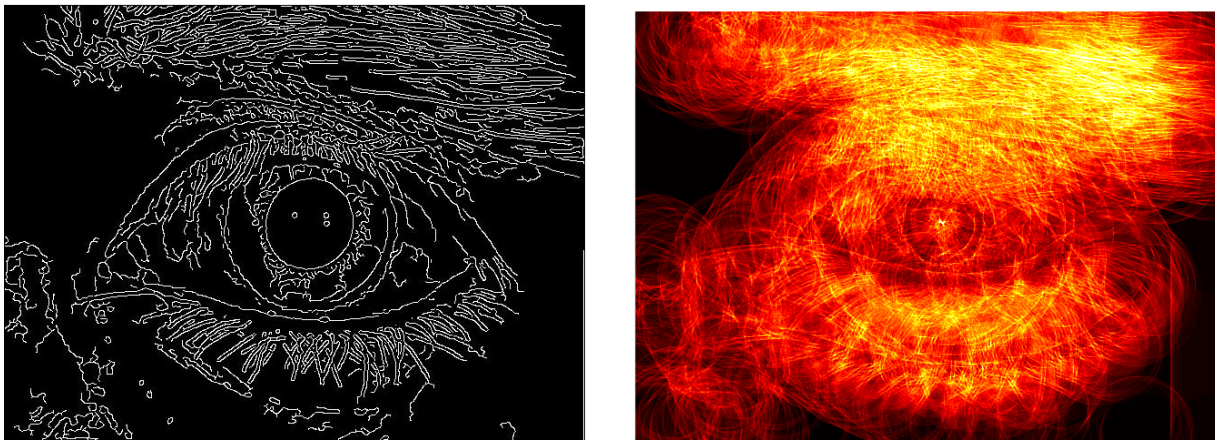


Figura 3.17: *Superfície correspondente ao acumulador no plano a,b com raio r= 51 para a imagem da Figura 3.16 .*

A partir dos dados do acumulador gerado pela transformada de Hough, é possível encontrar os parâmetros círculos existentes na imagem. Esse processo pode ser complexo caso não se tenha informações prévias sobre os círculos procurados. É efetuado procurando-se pelos picos existentes no acumulador para cada plano a, b correspondente a um valor de raio específico [46]. Se o valor acumulado no pico corresponde ao número de pixels de borda de um círculo de raio específico, as coordenadas desse pico provavelmente correspondem aos parâmetros do centro do círculo.

Na Figura 3.17 pode-se observar claramente a presença de um pico na superfície do acumulador que corresponde à localização do centro da circunferência. O centro de um círculo pode ser representado por um pico no acumulador com valor menor ao número de pixels de borda. Isso significa que um círculo pode estar incompleto ou ter forma de elipse. Caso seja difícil localizar picos específicos, pode-se ainda suavizar a superfície do acumulador.

Ao não se efetuar as etapas precedentes à votação do acumulador adequadamente, dificulta-se a correta detecção de circunferências. Na Figura 3.18, pode-se observar a detecção de bordas de uma imagem não suavizada, o que gera a representação no espaço de Hough também ilustrada na imagem. A superfície do acumulador correspondente pode ser vista na Figura 3.19. Percebe-se que o pico que representa a circunferência pupilar está presente em meio a muitos outros valores de pico que podem ocasionar a falsa detecção da circunferência procurada.



(a) *Imagem de bordas* (b) *Espaço de Hough*

Figura 3.18: *Detecção de bordas ineficaz e espaço de Hough correspondende.*

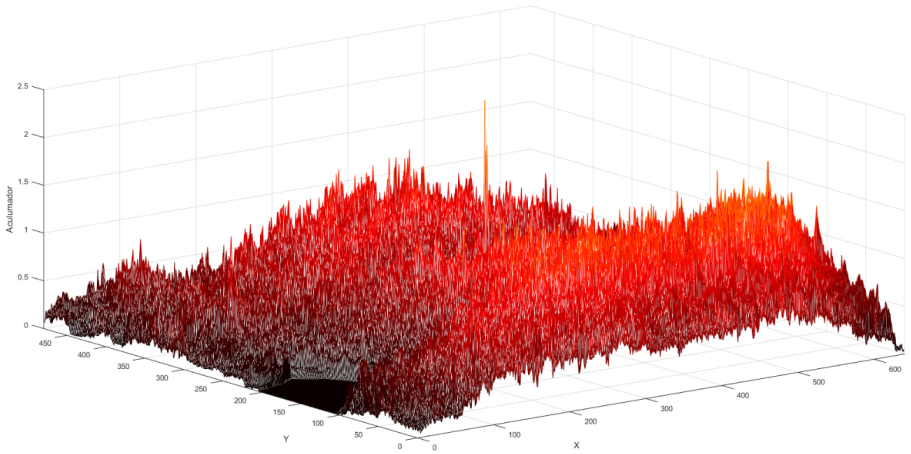


Figura 3.19: *Acumulador de Hough no plano a,b com raio=51 encontrado na imagem da Figura 3.18.*

O método de Hough se mostra, portanto, adequado à aplicação desenvolvida neste trabalho. Como visto na Figura 3.20, a circunferência pupilar é corretamente localizada, a depender da correta implementação das etapas de pré-processamento que são determinantes para a eficácia na aplicação da transformada.

3.3 Bancos de Imagens Oculares

Este trabalho efetua testes comparativos de precisão e tempo de processamento utilizando bancos de imagens oculares confiáveis que foram aplicados em trabalhos anteriores relacionados [2, 4, 36]. As bases de dados são descritas nas subseções seguintes.

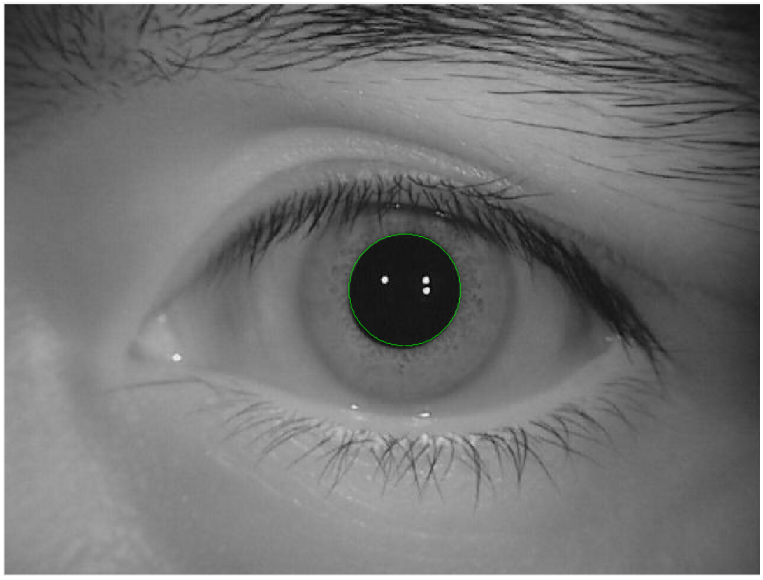


Figura 3.20: *Sucesso na detecção da circunferência pupilar através da transformada de Hough.*

3.3.1 Banco de Imagens CASIA-Iris

Este banco consiste em um conjunto de imagens de íris adquiridas em arquivos JPEG em 8 bits de níveis de cinza, coletadas sob iluminação infravermelho. O subconjunto de imagens utilizado neste trabalho é chamado CASIA-Iris-Lamp [47], coletado utilizando o sensor portátil produzido pela OKI e representado na Figura 3.21.

Figura 3.21: *Dispositivo de captura utilizado na coleta de imagens para o banco CASIA-Iris-Lamp.*

Uma iluminação é ligada e desligada próximo ao olho do indivíduo de forma a in-

troduzir variações intra classe. Esta base é bastante útil no estudo do comportamento não-linear da íris em relação à iluminação e representação característica de íris, uma vez que as imagens foram obtidas sob condições variáveis de iluminação.

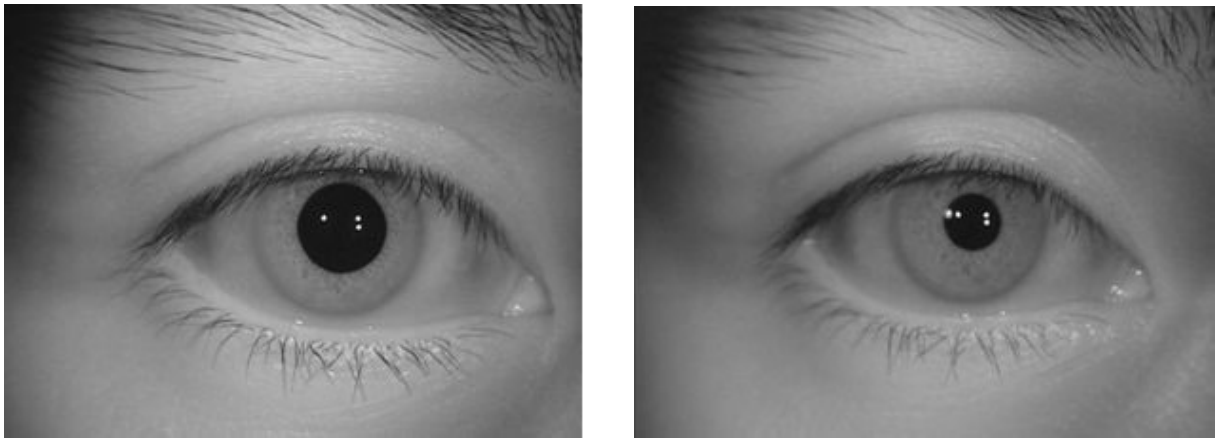


Figura 3.22: *Exemplos de imagens do banco CASIA-Iris-Lamp*

As 16.212 imagens desta base de dados foram coletadas em ambiente fechado. Foram utilizados 411 indivíduos, em sua maioria estudantes da instituição CASIA. As imagens possuem resolução de 640x480. A maioria dos exemplos presentes neste trabalho utilizam imagens desta base de dados. Este foi também utilizado em trabalhos de referência como Gontijo [2].

Se reconhece, portanto, a utilização de imagens coletadas pela *Chinese Academy of Sciences Institute of Automation* (CASIA), e presentes em seu banco de dados disponível para fins educacionais e de pesquisa. Todos os direitos são reservados.

3.3.2 Banco de Imagens LAVI Iris DB2 (USP)

Esta base de dados foi criada pelo *'Laboratory of Computer Vision'* (LAVI) da Universidade de São Paulo e coordenado pelo professor Dr. Adilson Gonzaga. Contém 212 vídeos, de 53 indivíduos diferentes. São 4 vídeos por indivíduo que apresentam a resposta reflexiva pupilar a um estímulo luminoso do tipo degrau. Os vídeos foram gerados estimulando-se um olho com pulsos luminosos de forma a provocar a contração e dilatação pupilar e simultaneamente capturando-se imagens do outro olho iluminado por LEDs infravermelho [48].

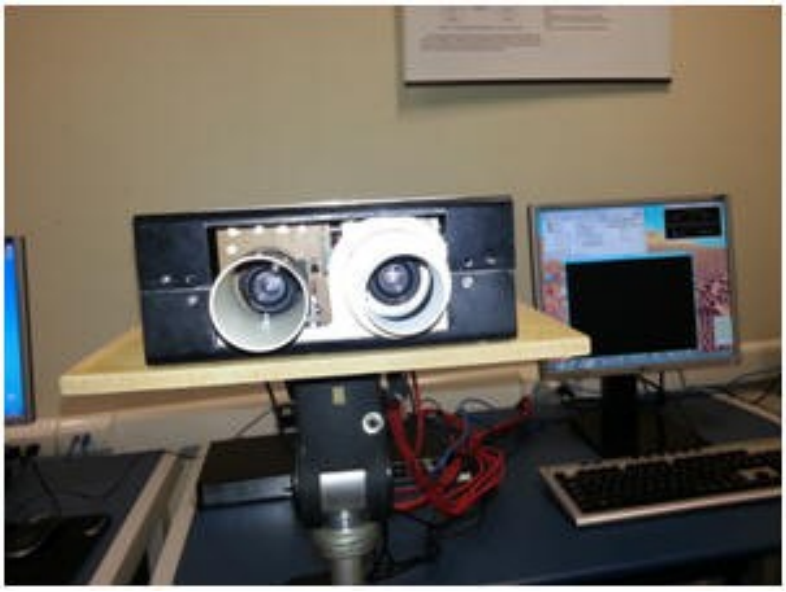


Figura 3.23: *Dispositivo de captura de imagens oculares.*

O dispositivo de gravação pode ser visto na Figura 3.23. Cada sequência de vídeo possui duração de 1 minuto e 6 segundos, a uma taxa de quadros de 15 fps, resultando em um total de 1000 quadros. O estímulo luminoso é efetuado duas vezes em cada sequência, produzindo respostas de contração e dilatação pupilar associadas. A Figura 3.24 apresenta alguns exemplos de imagens desta base de dados.

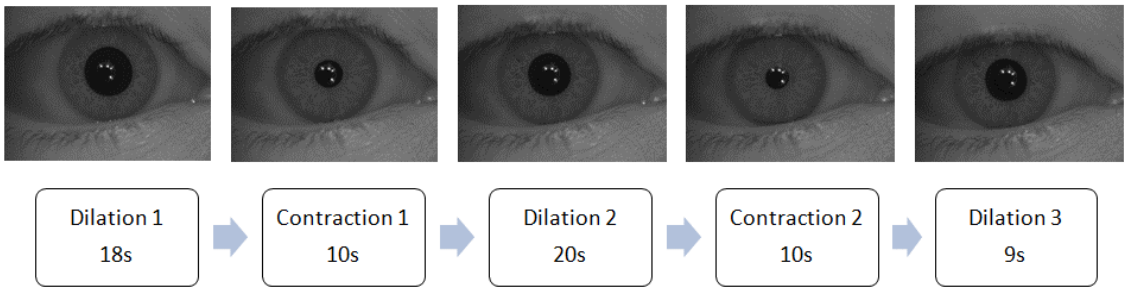


Figura 3.24: *Intervalos de gravação dos vídeos com iluminação dinâmica.*

O protótipo de captura foi construído utilizando duas câmeras multi-espectrais para registrar as imagens em cor visível e em infravermelho, na resolução de 1024x768. As sequências de vídeo em infravermelho próximo foram sincronizadas com o pulso de luz visível gerado por um LED branco.

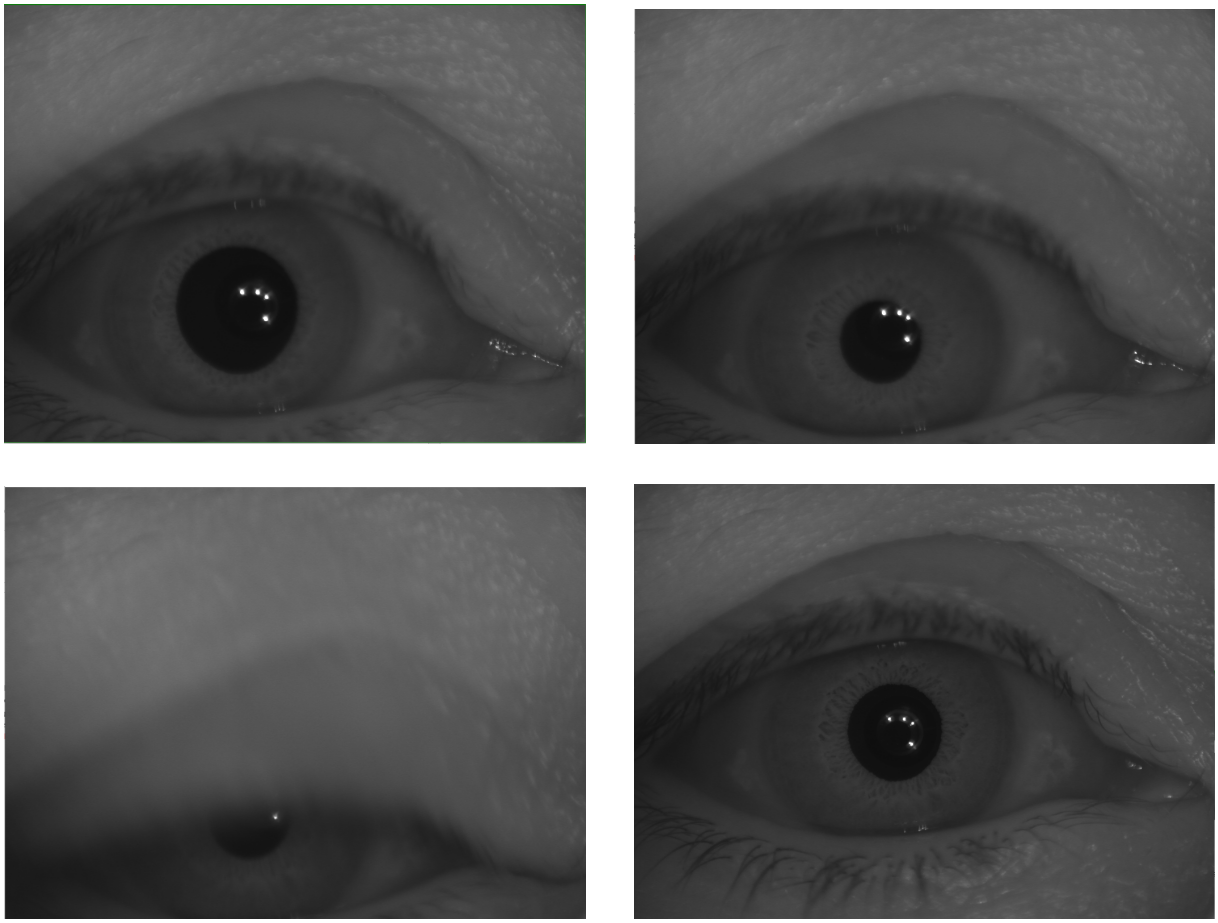


Figura 3.25: *Exemplos de imagens do banco de dados Iris DB2.*

3.4 Visão Computacional

A automação moderna necessita do auxílio de sensores artificiais de modo a proporcionar a utilização dos cinco sentidos. A visão é, talvez, o mais importante dos sentidos, tornando a visão computacional uma ferramenta indispensável a qualquer tarefa de reconhecimento.

Os avanços em tecnologia da computação, sensores, processamento de imagens e reconhecimento de padrões resultaram em melhores e mais baratas soluções de inspeção visual e ferramentas automáticas que possuem capacidade de visão [49]. O grande desafio consiste na simulação da visão humana, capturando e compreendendo uma imagem eletronicamente [50].

Técnicas de aperfeiçoamento de sistemas de processamento digital de imagens envolvem o desenvolvimento de algoritmos mais eficientes de processamento. Outra abordagem possível consiste na escolha de linguagens de programação mais adequadas à aplicação

desejada. Melhor desempenho pode ser adquirido através da utilização de hardware específico e mais poderoso ou ainda da exploração do paralelismo e o gerenciamento eficiente de memória.

Este trabalho busca efetuar otimizações através da escolha ótima de hardware, linguagem de programação e algoritmos efetivos na aferição automática de características pupilares.

Existem atualmente diversas ferramentas de software voltadas a aplicações de visão computacional. Alguns exemplos são as ferramentas: OpenCV, DevIL, CImg, Simd e CxImage, entre muitas outras. Consistem em bibliotecas e algoritmos voltados à manipulação de imagens (imagens estáticas e em vídeo), desde a captura ao reconhecimento de padrões e descrição.

Dentre as ferramentas existentes, este trabalho utiliza a biblioteca OpenCV, devido a suas características de algoritmo, extensa documentação e utilização, e custo zero – é uma biblioteca de código aberto.

3.4.1 OpenCV

OpenCV é uma biblioteca de uso comercial e acadêmico livre e foi desenvolvida a partir do ano 2000, visando o auxílio no desenvolvimento de algoritmos de visão computacional multiplataforma. É escrita na linguagem C++ e sua interface primária é em C++, apesar de possuir suportes a outras linguagens. Funciona com as interfaces C++, C, Python e java, sendo suportado pelas plataformas Windows, Linux, Mac OS, iOS e Android. Possui funcionalidades de processamento multi-núcleo, aceleração de hardware e foco em aplicações em tempo real.

A biblioteca OpenCV pode utilizar a aceleração em hardware em sistemas compatíveis com OpenCL. Possui mais de 350 algoritmos de visão computacional como filtros de imagem, calibração de câmera, reconhecimento de objetos, análise estrutural e outros.

Capítulo 4

Algoritmo Proposto

O algoritmo proposto neste trabalho deve ser capaz de efetuar o rastreamento de características pupilares humanas correspondentes à resposta a uma iluminação variável aplicada.

O desenvolvimento desta rotina consiste inicialmente na definição dos requisitos de software, correspondentes às funcionalidades necessárias à aplicação [51]. Neste caso, os requisitos dependem diretamente de como deve ser efetuada a localização pupilar, quais os dados de entrada e qual a saída esperada do sistema. A etapa de definição de requisitos é caracterizada na seção 4.1.

Parte essencial na concepção do software compreende a análise minuciosa de sistemas afins já desenvolvidos com o intuito de identificar possíveis funcionalidades adicionais, lacunas de desenvolvimento e otimizações a serem ainda realizadas. Subsequentemente são definidas as características de projeto e iniciada a codificação, implementando-se as funcionalidades anteriormente levantadas. Esta etapa está detalhadamente descrita nas seções 4.2 e 4.3 a seguir.

Ulteriormente são efetuados os testes de programa (seção 4.6), onde verifica-se suas funcionalidades e identifica-se *bugs* e defeitos. Checa-se o funcionamento adequado do programa e em caso de defeitos os mesmos são sistematicamente corrigidos.

Esta seção apresenta em detalhes as etapas de desenvolvimento e o software resultante. Salienta-se, entretanto, que o processo de desenvolvimento de software é contínuo significando a necessidade de constante manutenção e atualização de um bom software.

4.1 Definição de Requisitos

A principal funcionalidade do software desenvolvido se resume na detecção das características pupilares em uma imagem. Dessa forma, o requisito mais importante é a análise automática de uma imagem do olho humano de forma a extrair o diâmetro e posição pupilares correspondentes.

Complementarmente, o software deve ser capaz de efetuar o mesmo processo de detecção pupilar a partir de arquivos de vídeo, idealmente em tempo real, de forma a possibilitar o processamento de um stream de vídeo proveniente de uma câmera instantaneamente.

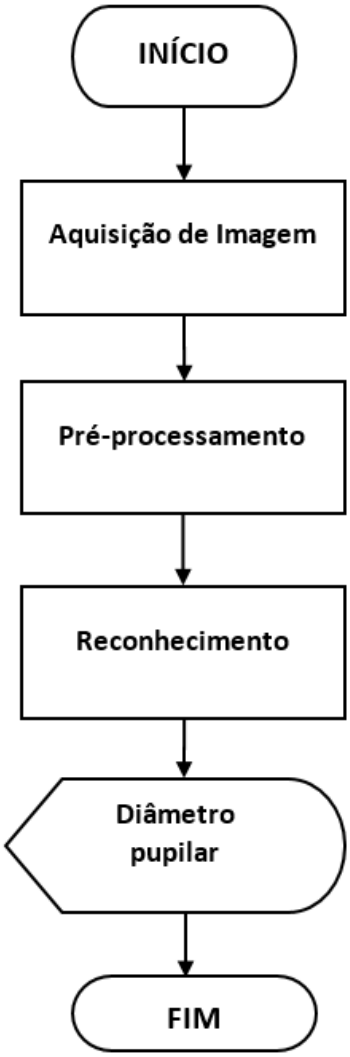


Figura 4.1: *Diagrama de blocos de um algoritmo para extração do diâmetro pupilar.*

Por último, será adicionado ao software a capacidade de capturar uma sequência de vídeo de uma câmera, efetuar o pré-processamento e extrair as medidas pupilares instantaneamente, ou seja, em tempo real. A otimização de software feita com base em trabalhos existentes [4, 2] é fundamental para que se possa atingir desempenho de algoritmo suficiente para o processamento em tempo real.

4.2 Pré-Processamento

A Transformada circular de Hough é utilizada em grande parte da literatura existente relacionada à extração de características oculares. Consiste em uma técnica robusta e eficiente para detecção de circunferências em imagens, incluindo-se as circunferências correspondentes à borda pupilar na presente situação.

A efetividade do algoritmo de rastreamento pupilar, no entanto, é fortemente dependente da qualidade da imagem analisada e das técnicas de pré-processamento aplicadas na correção e acentuação de suas características. Ao se ignorar a etapa de pré-processamento da imagem na presente aplicação, ocorre a localização de múltiplas circunferências em uma imagem, além da correspondente à pupila, como observa-se na Figura abaixo.

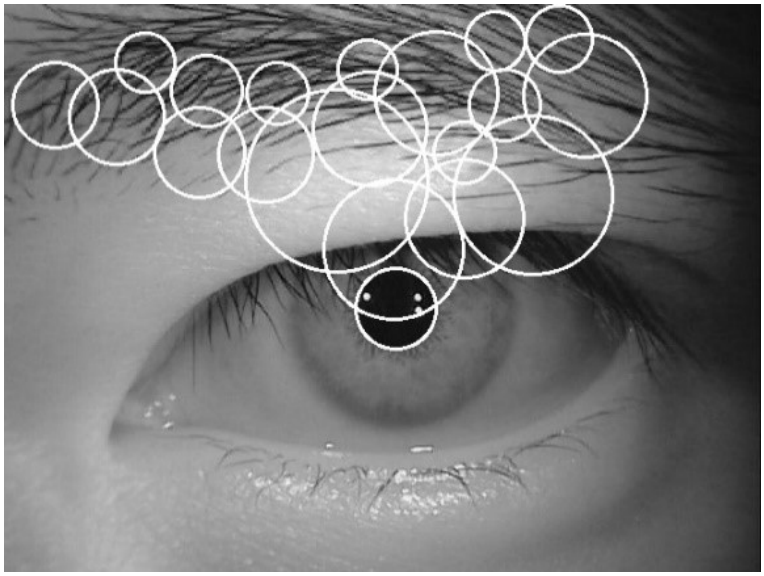


Figura 4.2: *Detecção de circunferências em imagem sem pré-processamento.*

O pré-processamento da imagem adquirida é fundamental para o sucesso do reconhecimento de forma proposto neste algoritmo. Os processos de tratamento da imagem diferem

de acordo com o objetivo do sistema de processamento, a natureza das imagens obtidas e a qualidade das mesmas. A presença de ruído ou distorções na imagem requer a utilização de filtros corretores, transformações e por vezes, restaurações.

A detecção de contornos e reconhecimento de objetos é efetuada geralmente em imagens em níveis de cinza. A primeira operação efetuada, portanto, consiste na conversão da imagem em cores para uma imagem em níveis de cinza. Em seguida deve-se condicionar a imagem de forma a torná-la ótima para o reconhecimento de características.

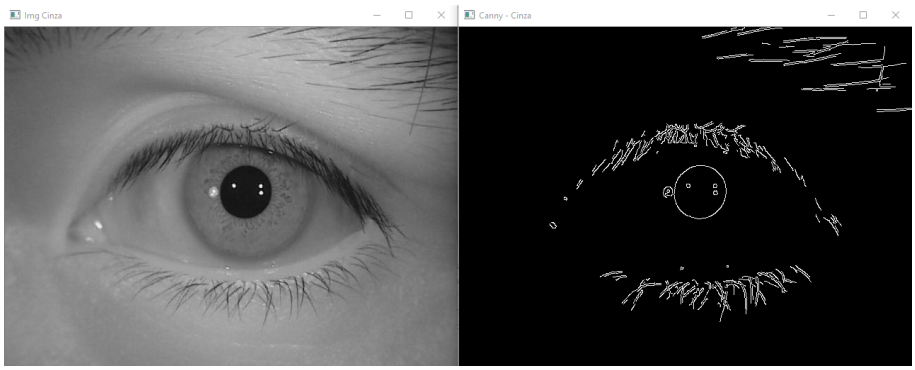


Figura 4.3: *Imagem Capturada e sua detecção de bordas.*

Bernadelli [4] e Gontijo [2] demostram a eficácia do pré-processamento composto por equalização do histograma da imagem – eficaz no aumento do contraste de imagens – seguido de sua suavização através de filtros e então detecção de bordas pelo método de Canny.

Foram efetuados testes com o banco de imagem de referência desta dissertação e dos trabalhos acima mencionados [47] de forma a avaliar a eficácia de cada uma das etapas de pré-processamento comumente utilizadas. Na Figura 4.3 pode-se ver a imagem de referência e sua detecção de bordas através do método de Canny, sem as etapas de pré-processamento (equalização de histograma e suavização). Percebe-se a presença de bordas que não são interessantes ao propósito deste sistema, como cílios e sobrancelhas.

Se efetuamos a equalização do histograma antes de efetuar a detecção de contornos, obtemos o resultado observado na Figura 4.4 . Neste caso, constata-se a presença de mais contornos irrelevantes à detecção pupilar. O contorno pupilar, objeto que se deseja detectar, acaba sendo degradado. A equalização de histograma será, portanto, desconsiderada neste trabalho, visto que apresenta efeito destrutivo na imagem-objeto.

Avaliando-se a suavização da imagem, representada na Figura 4.5 , pode-se observar

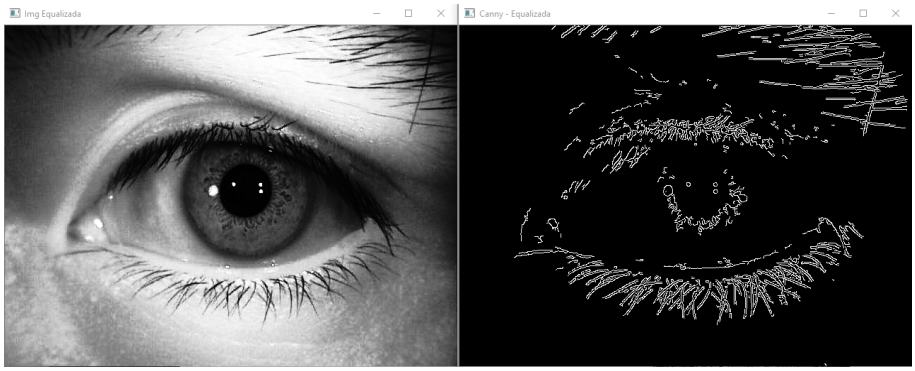


Figura 4.4: *Imagem com histograma equalizado e sua detecção de bordas.*

que quase todos os contornos não relacionados à pupila são eliminados. Os filtros passa-baixa, que efetuam a suavização, atuam de forma a 'borrar' os contornos e assim a detecção de contornos passa a desconsiderá-los. O ajuste do filtro de suavização, entretanto, é crítico para que não se elimine detalhes importantes da imagem que possam comprometer o reconhecimento.

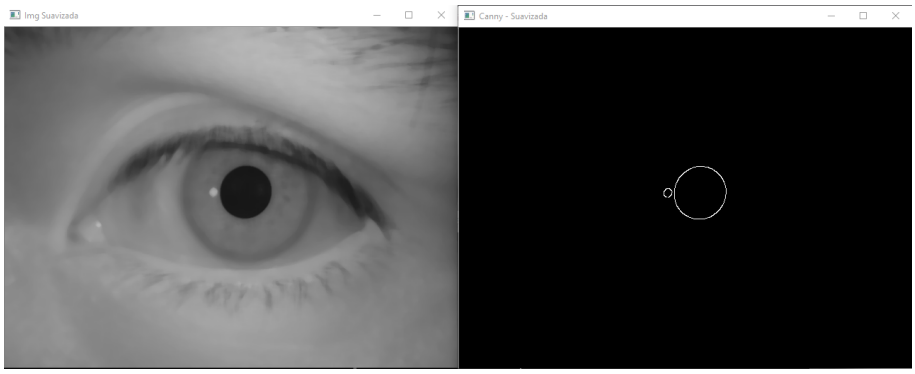


Figura 4.5: *Imagem suavizada (pelo filtro de mediana) e sua detecção de bordas.*

Como último teste de eficácia no pré-processamento, demostra-se o resultado obtido (Figura 4.6) ao se efetuar a equalização do histograma seguida da suavização, como efetuado nos trabalhos de referência. Observa-se a completa eliminação do contorno pupilar, o que torna essa alternativa de tratamento da imagem inadequada, considerando-se as características de captura do banco de dados utilizado e os parâmetros utilizados na detecção pupilar.

À vista disso, conclui-se que o pré-processamento ótimo para o rastreamento pupilar em imagens do banco CASIA [47] compreende a suavização da imagem, precedida da conversão para escala de cinza. A suavização demostrada nas figuras acima foi efetuada pelo filtro de mediana com tamanho de máscara 11, e a detecção de bordas efetuada pelo

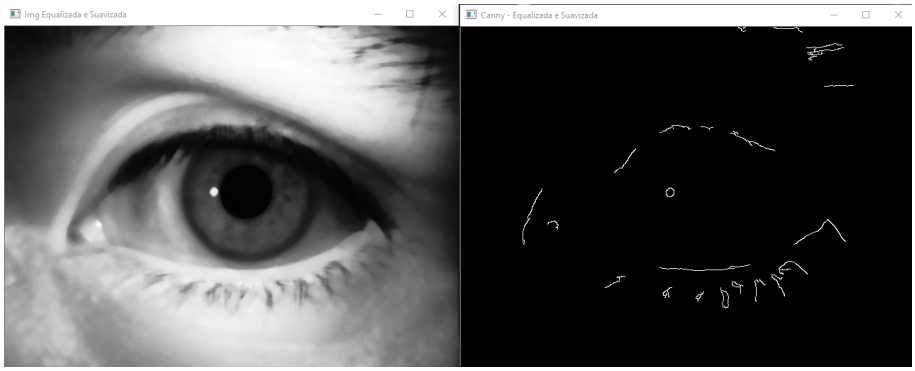


Figura 4.6: *Imagem após equalização de histograma e suavização e sua detecção de bordas.*

método de Canny por histere [40], utilizando-se os limiares inferior e superior de 125 e 225, respectivamente.

A suavização, no entanto, pode ser efetuada por diferentes filtros passa-baixa e utilizando-se diferentes níveis de suavização. O ajuste dessas características tem influência considerável na taxa de acerto do sistema, demandando extensivos testes comparativos de forma a se determinar a melhor combinação de parâmetros para o banco de imagens utilizado [47].

A próxima seção compreende os testes de eficácia do processo de suavização, necessários para se definir o tipo de filtro de suavização mais adequado à aplicação e o nível de suavização que gera melhores resultados. Essa combinação de parâmetros é crucial para a maior taxa de acertos do sistema e pode representar grande contribuição aos trabalhos existentes.

4.3 Otimização da Taxa de Localização Pupilar

O primeiro quesito a ser explorado na tentativa de aperfeiçoamento do algoritmo corresponde à sua precisão. Nesta seção procura-se determinar maneiras de melhorar a taxa de acertos na busca pela circunferência pupilar. Para essa tarefa, foram avaliadas as etapas presentes nos algoritmos já desenvolvidos.

Os métodos de correção comumente utilizados em algoritmos com objetivos semelhantes a este compreendem basicamente a suavização da imagem através de filtros. A filtragem é uma tarefa fundamental no processamento de sinal e imagem, servindo à extração seletiva de certos aspectos em uma imagem que são importantes a determinada

aplicação.

As operações de filtragem permitem a remoção de ruído em imagens, extração de características visuais interessantes, re-amostragem, entre outros [31]. Os filtros aqui considerados são os filtros de suavização ou filtros passa-baixa, e correspondem ao filtro de média, o filtro gaussiano, o filtro de mediana e o filtro bilateral.

O aperfeiçoamento da precisão do algoritmo realiza, portanto, comparações de desempenho no rastreamento pupilar utilizando diferentes filtros de suavização e diferentes níveis de suavização, de modo a estabelecer relações entre a acurácia resultante das diferentes técnicas de pré-processamento utilizadas.

O teste comparativo presente nesta seção, consiste na análise do número de circunferências pupilares efetivamente encontradas em função do nível de suavização - definida pela dimensão da máscara de convolução utilizada na suavização, ou seja, do tamanho da vizinhança do pixel central. Efetuar essa avaliação nos diferentes tipos de filtros de suavização (filtro de média, filtro gaussiano, filtro de mediana e filtro bilateral) possibilitará a futura análise comparativa entre eles para que se possa escolher o mais adequado.

A efetividade do método de localização pupilar depende diretamente do nível de suavização do filtro na etapa de pré-processamento que, por sua vez, é definida pelo tamanho da vizinhança em torno do pixel central a ser substituído no processo de filtragem. Logo, espera-se com esse teste, determinar o tamanho de máscara que apresenta maior rendimento para cada um dos filtros de suavização considerados.

Todos os testes efetuados foram realizados em imagens digitalizadas do olho humano obtidas do banco de dados CASIA [47]. O banco de dados utilizado é composto de 16.212 imagens em resolução 640x480, obtidas aplicando-se iluminação de forma a causar movimentação da pupila e íris em 411 indivíduos. O tamanho de máscara ótimo identificado será utilizado posteriormente na avaliação comparativa entre os diferentes filtros de suavização.

4.3.1 Filtro de Média

Ao se variar o nível de suavização deste filtro, obteve-se os resultados dispostos na Figura 4.7:

Percebe-se que os tamanhos de vizinhança que geram boa taxa de localização de circun-

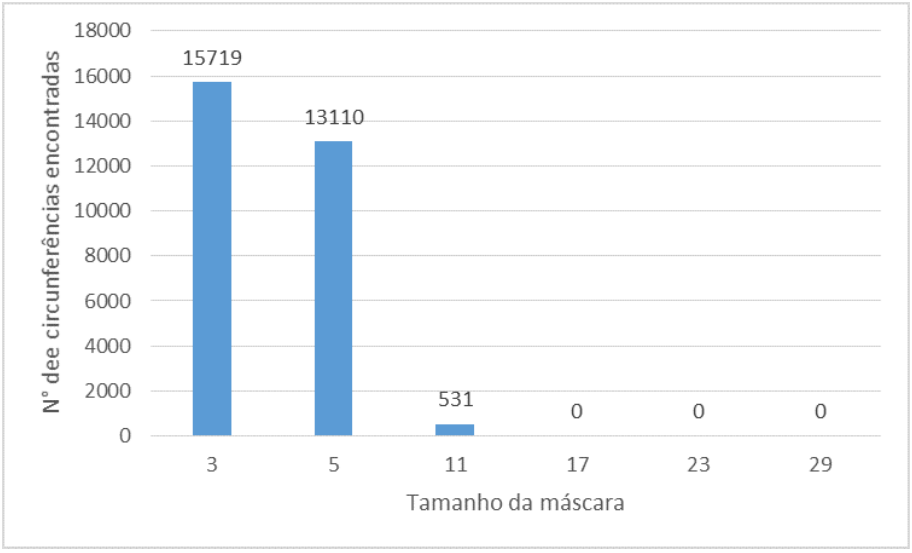


Figura 4.7: *Número de imagens onde foram encontradas circunferências utilizando-se o filtro de média.*

ferências são somente 3 e 5. Essas configurações resultam, respectivamente, na localização de circunferências em 15.719 (96,95%) e 13.110 (80,86%) imagens.

Entretanto, tais matrizes correspondem a um baixo nível de suavização, resultando na localização de falsas circunferências que justificam as altas taxas de circunferências encontradas.

Para dimensões de máscara com tamanho superior o rendimento foi praticamente nulo, pois os detalhes de borda são removidos, como se pode observar na Figura 4.8. Isso demostra a ineficácia deste filtro para a aplicação proposta, uma vez que baixos coeficientes de suavização resultam em falsas circunferências encontradas e coeficientes elevados suprimem as bordas pupilares necessárias à aplicação.

4.3.2 Filtro Gaussiano

O filtro gaussiano, utilizado na grande maioria das aplicações envolvendo suavização de imagens, se mostra mais efetivo na supressão de altas frequências sem comprometer o contorno pupilar. Máscaras de tamanho 3 e 5 resultam, como na suavização pelo filtro de média, na localização de muitas falsas circunferências. Uma vizinhança de tamanho 11, entretanto, gera rendimento de 70,43% correspondentes a 11.418 imagens cuja circunferência pupilar foi encontrada como ilustrado no gráfico da Figura 4.9:

Níveis de suavização bastantes superiores (que a suavização gerada por tamanho de

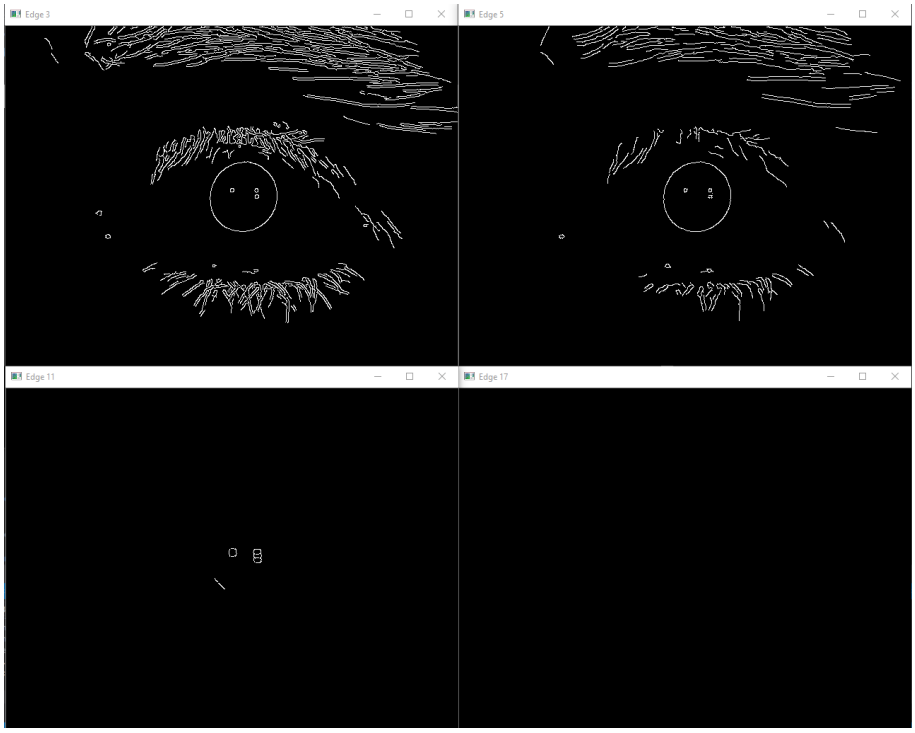


Figura 4.8: *Detecção de bordas em imagens suavizadas pelo filtro de média com tamanho de máscara (a) 3, (b) 5, (c) 11 e (d) 17.*

máscara 11) não geram bons resultados, assim como para o filtro de média, ainda que mantenham grande parte do contorno da pupila.

A Figura 4.10 ilustra a detecção de bordas em imagem suavizadas pelo filtro gaussiano com tamanho de máscara de 3, 5, 11 e 17.

4.3.3 Filtro de Mediana

Efetuando-se o mesmo teste com o filtro de mediana, obtêm-se os resultados do gráfico da Figura 4.11. O rendimento deste se mostra bastante superior aos filtros anteriormente avaliados, mesmo com a aplicação de altos níveis de suavização.

Recomenda-se a utilização de vizinhanças com tamanho superior a 5, de forma a evitar a localização de circunferências que não correspondem à pupila, assim como nas situações anteriores. Para vizinhanças com dimensão 11, 17 e 23 foram encontradas circunferências em 13.190 (81,36%), 12.500 (77,1%) e 10.863 (67%) imagens respectivamente.

A utilização deste filtro com máscara de tamanho 11x11 representa a configuração ótima para o sistema de localização pupilar aplicado a imagens do banco CASIA [47], mantendo alta taxa de circunferências pupilares efetivas encontradas ao mesmo tempo

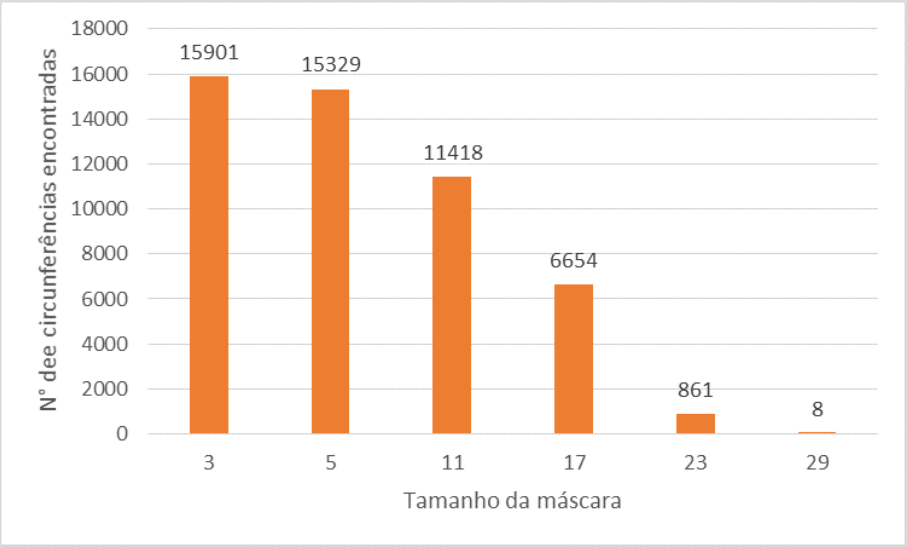


Figura 4.9: *Número de imagens onde foram encontradas circunferências utilizando-se o filtro gaussiano.*

que remove praticamente todas as localizações de falsas circunferências.

Através da Figura 4.12, onde está representada a detecção de bordas de imagens suavizadas pelo filtro de mediana com diferentes níveis de suavização, percebe-se que o contorno pupilar permanece intacto mesmo após a acentuada suavização.

4.3.4 Filtro Bilateral

A utilização do filtro bilateral apresenta rendimento superior ao filtro de média, ainda que seja menos efetivo que os filtros gaussiano e mediana.

Tamanhos de máscara 11 e 17 resultam em circunferências encontradas em 7.212 (44,5%) e 5.467 (33,7%) imagens respectivamente, e mesmo com níveis de suavização superiores a presença de múltiplas circunferências persiste.

A utilização do filtro bilateral não é recomendada, portanto, uma vez que mantém características de alta frequência da imagem e prejudica a localização da circunferência alvo do sistema, correspondente à borda pupilar.

Os resultados encontrados para todos os filtros utilizando-se diferentes tamanhos de máscara demonstram que baixos níveis de suavização (máscaras de tamanho inferior a 5) são ineficazes para rastreamento das corretas características pupilares. Vizinhanças de tamanho superior a 17 também são desaconselhadas, visto que causam a remoção de características da imagem determinantes ao rastreamento pupilar nas imagens de referência

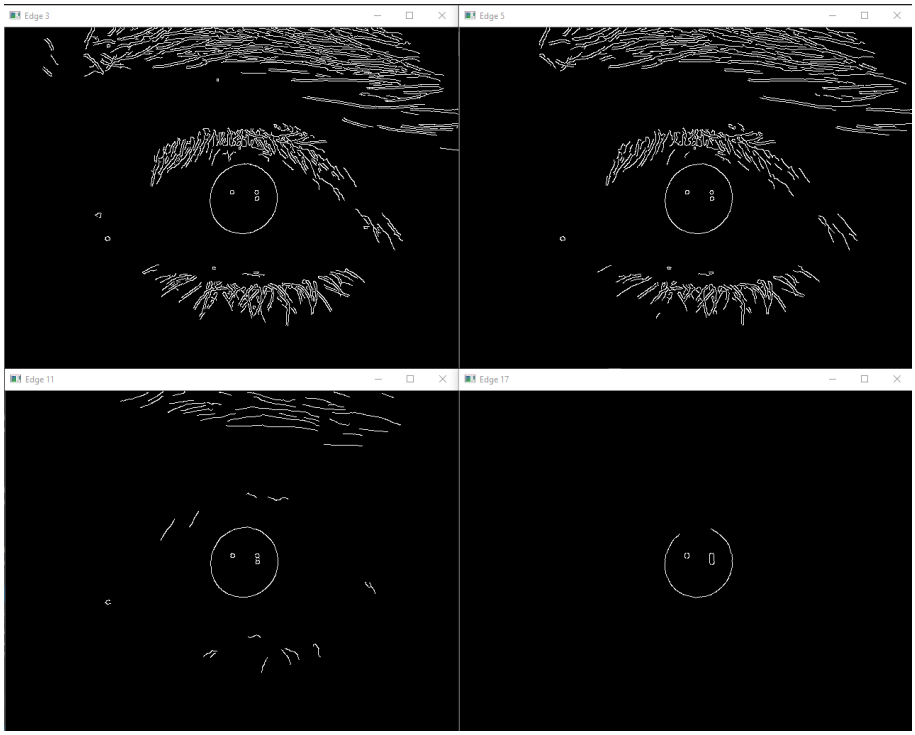


Figura 4.10: *Detecção de bordas em imagens suavizadas pelo filtro gaussiano com tamanho de máscara (a) 3, (b) 5, (c) 11 e (d) 17.*

[47].

Constata-se pela Figura 4.15, que a desfocalização excessiva da imagem faz com que nenhuma circunferência seja localizada, a depender do método de suavização utilizado. Somente a suavização pelo filtro de mediana resultou na correta localização pupilar nesse caso. Elevados coeficientes de suavização resultam ainda em um maior tempo de processamento devido à maior dimensão das matrizes de convolução que por sua vez aumentam consideravelmente o número de operações matemáticas necessárias à filtragem.

A Figura 4.15 representa o resultado da detecção pupilar através do método de Hough de identificação de circunferências para imagens profundamente suavizadas pelos filtros de média, gaussiano, de mediana e bilateral.

Logo, pode-se inferir que máscaras de tamanho igual ou próximo a 11 resultam em bom rendimento, representando bom equilíbrio entre permissão de detalhes de borda necessários ao rastreamento pupilar e suavização das altas frequências responsáveis pela localização de circunferências inadequadas.

Esta seção evidencia a relação entre o tamanho de máscara utilizado no processo de filtragem e a taxa de detecção de circunferências pupilares em imagens provenientes do

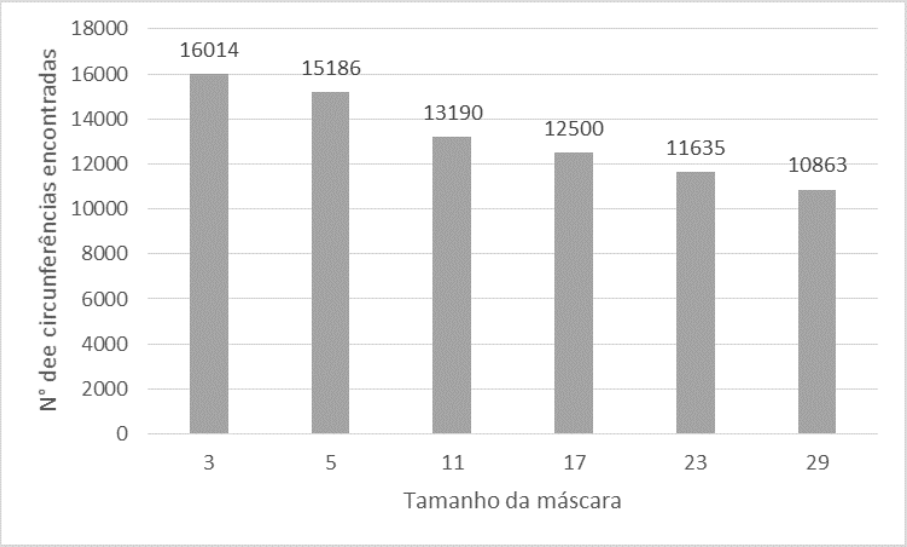


Figura 4.11: *Número de imagens onde foram encontradas circunferências utilizando-se o filtro de mediana.*

banco de imagens CASIA [47]. Salienta-se que os parâmetros adequados são dependentes de qualidades das imagens utilizadas, dependentes das condições de captura das imagens, contraste, resolução e focalização. O segundo banco de dados utilizado nesta dissertação [48], por exemplo, requer valores acentuadamente diferentes de limiares para detecção de bordas e coeficiente de suavização.

Na subseção 4.6.1 descreveremos os resultados comparativos entre os diferentes tipos de filtros, de forma a avaliar quantitativamente a influência da escolha adequada do filtro suavizador na eficácia do sistema.

4.4 Otimização do Tempo de Processamento do Software

Nesta etapa do aperfeiçoamento do software foram identificadas as melhorias que apresentam potencial de diminuição de tempo de processamento no algoritmo final.

O sistema de localização pupilar apresentado em [4] foi desenvolvido em linguagem Matlab®. A rotina possui uma etapa de pré-processamento que compreende a conversão da imagem de entrada para escala de cinza, equalização de histograma, suavização da imagem, detecção de bordas e tratamento das mesmas, seguido da etapa de localização

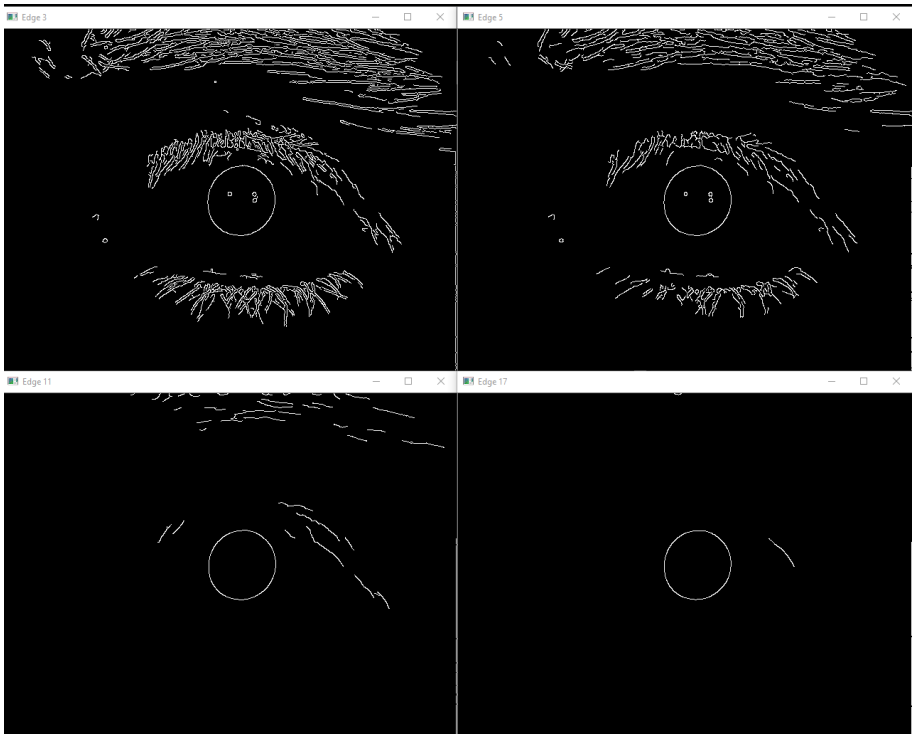


Figura 4.12: *Detecção de bordas em imagens suavizadas pelo filtro de mediana com tamanho de máscara (a) 3, (b) 5, (c) 11 e (d) 17.*

pupilar efetuada pela transformada circular de Hough, previamente descrita na seção 3.2.

Outra abordagem existente [2] realiza processamento similar e apresenta ganho considerável em performance em relação a [4]. O programa foi desenvolvido em linguagem C#® e dispõe de etapas suplementares no pré-processamento. Foi desenvolvida a busca eficiente de valores de raio pupilar e a segmentação da imagem de acordo com a sequência de vídeo. Os aperfeiçoamentos apresentados resultaram em um melhor desempenho em tempo de processamento e taxa de acerto [2].

A primeira característica básica do presente sistema definida a partir das constatações expostas foi a linguagem de programação a ser utilizada. A linguagem de programação C++ foi escolhida, uma vez que consiste em uma linguagem orientada a objetos para desenvolvimento de programas de uso geral. Possibilita a utilização de centenas de bibliotecas voltadas ao desenvolvimento de aplicações em áreas específicas, incluindo visão computacional e reconhecimento de padrões que são o foco deste trabalho.

A linguagem C++ é uma linguagem bastante flexível podendo ser utilizada na grande maioria dos tipos de hardware existentes. É uma linguagem compilada, ou seja, gera programas em código de máquina, o que resulta em um ganho considerável em desem-

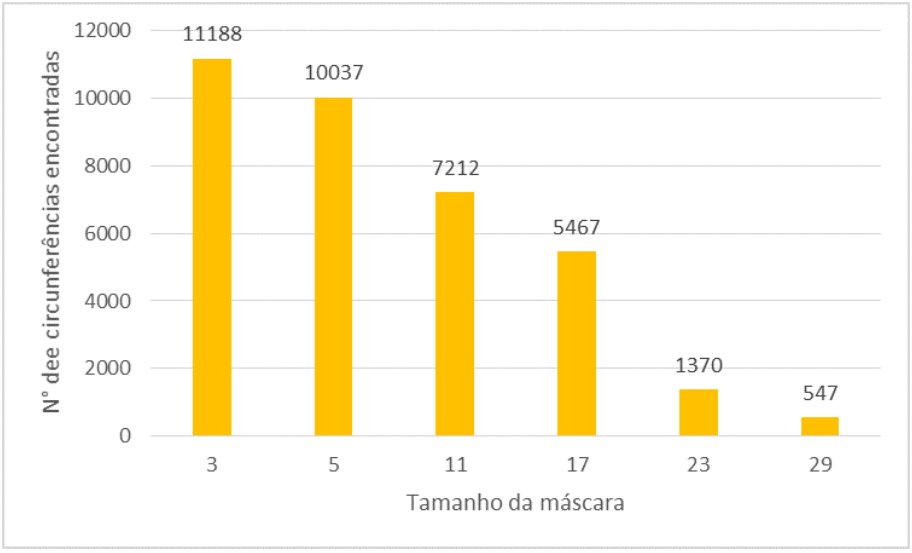


Figura 4.13: *Número de imagens onde foram encontradas circunferências utilizando-se o filtro bilateral.*

penho sobre linguagens interpretadas como Python e Matlab®. Suas características de linguagem de baixo nível a tornam adequada para aplicações em dispositivos embarcados.

Outro aperfeiçoamento do algoritmo sugerido por [2] está relacionado à definição acurada da região de interesse na imagem a ser processada. Como apresentado na seção 3.2 a transformada circular de Hough efetua uma varredura em toda a imagem a fim de localizar circunferências com raios dentro de uma faixa de valores pré-estabelecidos. Ao se delimitar rigorosamente a área de busca da transformada de Hough (região de interesse), ignorando partes desnecessárias da imagem suspeita-se que haja um possível ganho no desempenho do sistema projetado.

Será explorado ainda neste trabalho a delimitação da faixa de valores de raio a serem utilizados na busca de circunferências pela transformada circular de Hough. Sabe-se que a pupila não efetua variações súbitas no diâmetro em uma sequência de imagens capturadas [4], considerando-se uma velocidade de captura em torno de 30 frames por segundo. Logo, podemos estimar uma faixa de valores prováveis de diâmetro de um frame a partir da medida identificada no quadro anterior, diminuindo a faixa de busca e consequentemente o tempo de processamento do algoritmo.

O desenvolvimento do software na linguagem C++ juntamente com biblioteca OpenCV e as técnicas de delimitação da região de interesse e busca de circunferências das imagens processadas são as principais otimizações utilizadas, a partir das quais procura-se obter

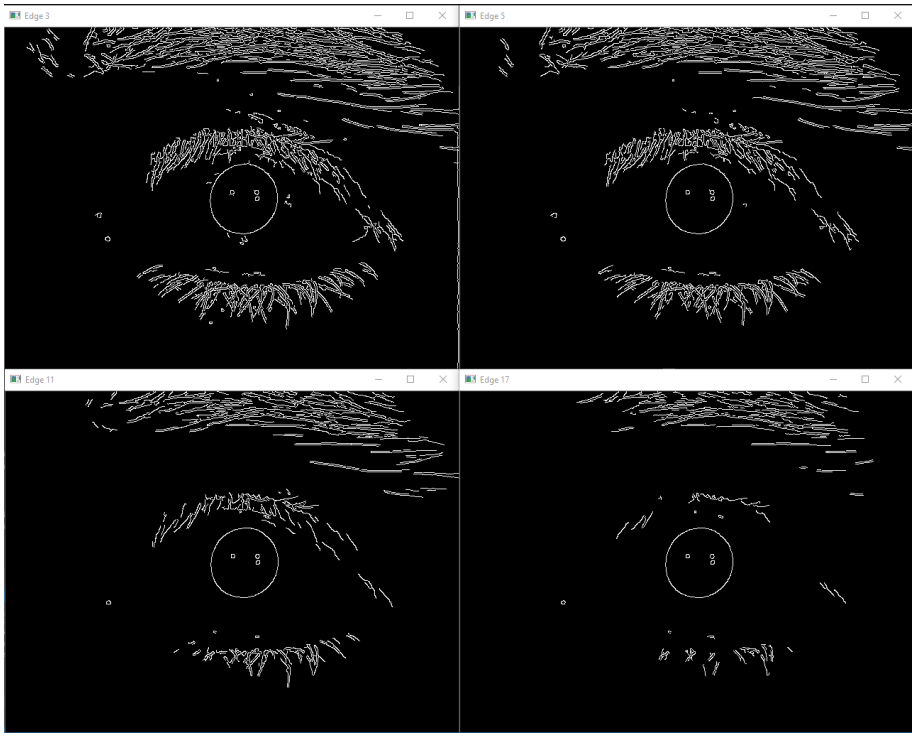


Figura 4.14: *Detecção de bordas em imagens suavizadas pelo filtro bilateral com tamanho de máscara (a) 3, (b) 5, (c) 11 e (d) 17.*

um ganho em tempo de processamento que seja determinante na aplicação em tempo real do sistema.

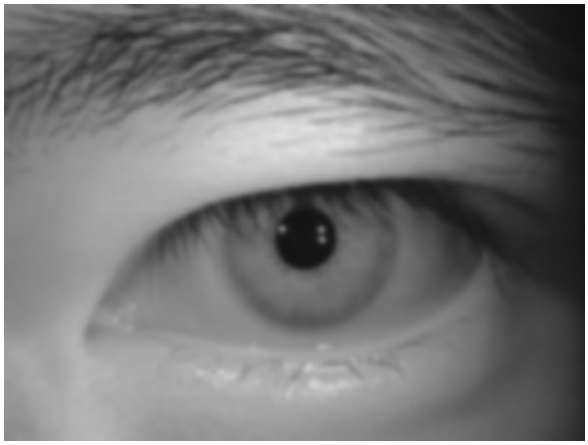
4.5 Descrição do Software Resultante

A partir das possibilidades de aperfeiçoamento do algoritmo acima descritas, buscou-se estruturar o programa para ser efetivo na tarefa de localização pupilar e possibilitar análises comparativas relativas a sistemas similares.

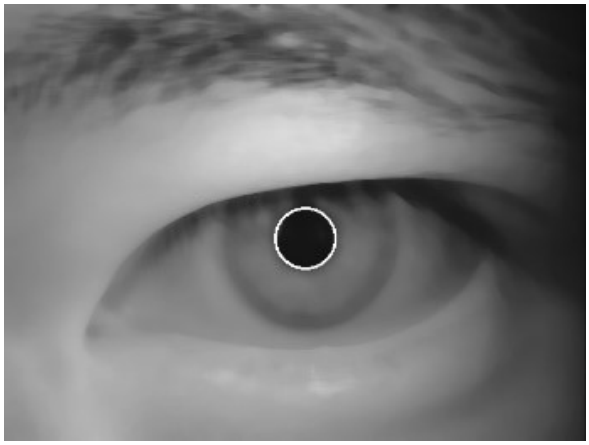
O algoritmo de localização da circunferência pupilar desenvolvido possui etapas de aquisição das imagens, pré-processamento (suavização das imagens), detecção de bordas e localização pupilar (através da transformada de Hough). Foi desenvolvido na linguagem de programação C++, utilizando a biblioteca de código aberto OpenCV para processamento de imagens.

Como pode ser visto no código do programa, contido no apêndice, há funções implementadas na biblioteca OpenCV para as operações necessárias [52]. A conversão de cores de imagens é efetuada através do comando *'cvtColor'*, a suavização pelo filtro de

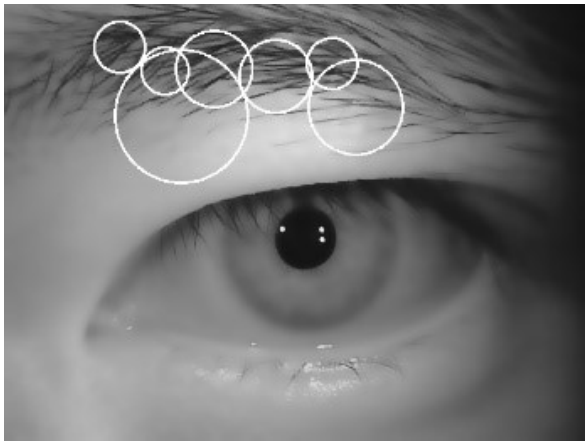
(a) *Filtro de média*



(b) *Filtro gaussiano*



(c) *Filtro de mediana*



(d) *Filtro bilateral*

Figura 4.15: *Detecção de Hough em imagens suavizadas com máscara de tamanho 17x17.*

mediana é realizada pelo comando *'medianBlur'* e a localização de circunferências pelo método de Hough [43] operada pelo comando *'HoughCircles'*, comando este que efetua automaticamente a detecção de bordas pelo método de Canny. Os comandos acima citados requerem, naturalmente, os devidos parâmetros de forma a proceder a conformação da imagem e localização pupilar.

4.6 Testes e Resultados

Foram efetuados diferentes testes a fim de aferir a validade e eficácia das técnicas implementadas em software. O primeiro teste busca constatar quais os ganhos obtidos na precisão de localização pupilar. Em seguida são descritos dois testes relacionados ao aperfeiçoamento do algoritmo em tempo de processamento.

4.6.1 Teste – Taxa de Localização Pupilar

A análise conjunta de todos os filtros aqui considerados foi efetuada utilizando-se máscara de tamanho 11x11, comprovadamente mais eficiente que as demais dimensões de vizinhança. Foram obtidos os seguintes números de imagens com circunferências adequadas localizadas: 531, 11.418, 13.190 e 7.212 para os filtros de média, gaussiano, de mediana e bilateral, respectivamente. O resultado está disposto na Figura 4.16:

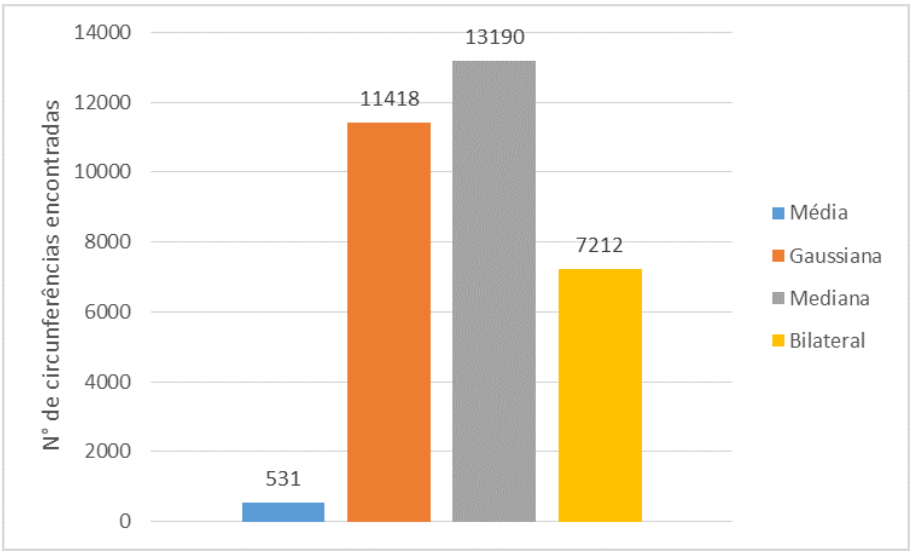


Figura 4.16: *Número de imagens onde foram encontradas circunferências utilizando-se máscara 3x3.*

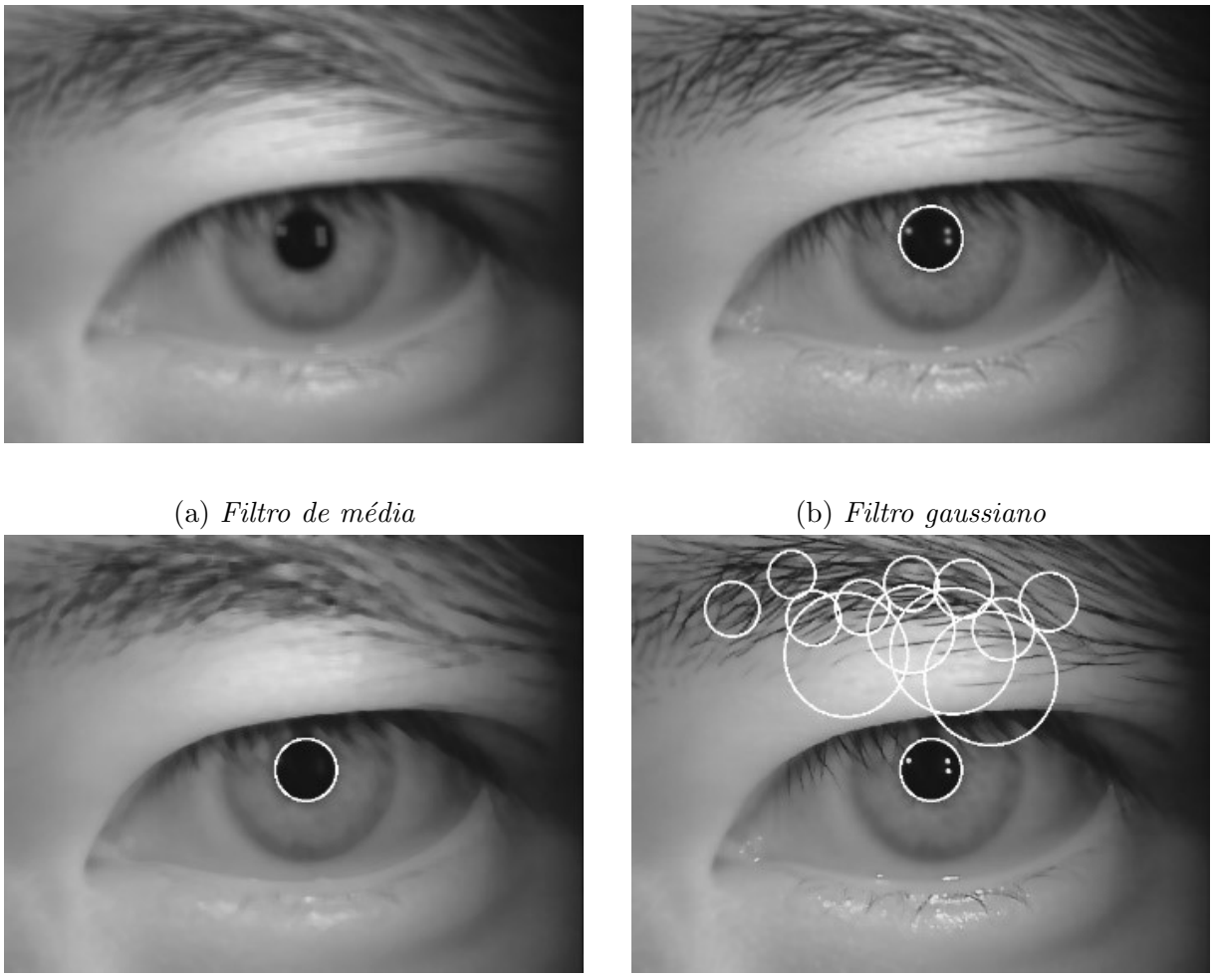
O filtro de mediana claramente apresenta melhor resultado em relação aos outros filtros de suavização. A melhora na efetividade corresponde a 67% em relação ao filtro de média e a 11% em relação ao filtro gaussiano.

Constata-se que a escolha do tipo de filtro suavizador é determinante na acurácia e efetividade de sistemas de reconhecimento de formas e contornos em imagens. Os resultados obtidos apresentam diferença média de 11% na taxa de identificação de circunferências

entre o filtro gaussiano, utilizado na grande maioria dos trabalhos relacionados, e o filtro de mediana, aqui sugerido.

Adicionalmente, o sistema desenvolvido efetua um ajuste preciso no nível de suavização utilizado pelo filtro selecionado, de forma a obter o maior rendimento possível do sistema. Esse parâmetro se mostra ainda mais determinante na obtenção de bons resultados de identificação pupilar. Como descrito na seção, uma máscara de suavização de dimensão 11x11 se mostra a mais adequada para a presente aplicação.

A Figura 4.17 representa a localização pupilar resultante após a suavização pelos filtros de média, gaussiano, de mediana e bilateral, todos com tamanho de máscara 11.



(a) *Filtro de média* (b) *Filtro gaussiano*

(c) *Filtro de mediana* (d) *Filtro bilateral*

Figura 4.17: *Detecção de Hough em imagens suavizadas com máscara de tamanho 11x11.*

O sistema desenvolvido obtém, suplementarmente, ganhos de performance ainda superiores em relação a sistemas semelhantes, devido à escolha de parâmetros ótimos nas

etapas posteriores à filtragem.

4.6.2 Teste – Tempo de Processamento

Com o propósito de aferir o desempenho dos aperfeiçoamentos sugeridos em tempo de processamento, mediu-se o tempo médio gasto para processar sequências de imagens utilizando o algoritmo desenvolvido sem otimizações seguido dos algoritmos otimizados.

Foram utilizadas diferentes sequências, cada uma contendo 20 imagens, retiradas do banco de imagens CASIA [47] para as quais mediu-se o tempo de processamento resultantes dos algoritmos. As imagens possuem resolução de 640x480 pixels, e estão em escala de cinza. Estes testes foram realizados em um computador com processador Intel® Core i5-4300U com 2 núcleos e frequência padrão de 1,9GHz, 4GB de memória RAM, utilizando arquitetura 64 bits e sistema Operacional Windows 10®.

O algoritmo inicial desenvolvido (não inclui os aperfeiçoamentos sugeridos anteriormente) consiste na busca de circunferências com raio entre 25 e 80 pixels varrendo-se toda extensão a imagem analisada. Obteve-se, após sucessivas aplicações do algoritmo em diferentes sequências de imagens, um tempo de processamento médio de 700 ms. Isso corresponderia a uma taxa de quadros de aproximadamente 28,57 fps, visto que cada sequência contém 20 imagens.

A Figura 4.18 representa duas janelas do programa em operação, tendo localizado a circunferência pupilar e retornado o raio e posição do centro, para duas imagens diferentes.

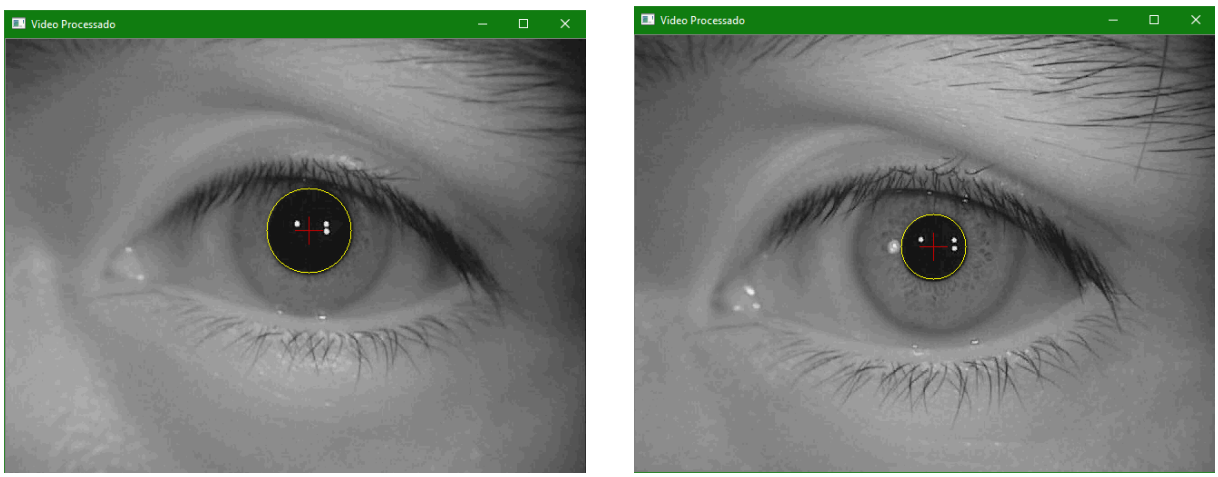


Figura 4.18: *Execução do algoritmo sem busca otimizada de raio pupilar.*

O primeiro algoritmo aperfeiçoado efetua a delimitação na busca do raio na imagem alvo, de forma a rastrear raios com até 10 pixels de diferença em relação ao quadro anterior em uma sequência de imagens. Essa estratégia resultou em um tempo médio de processamento de 672 ms, equivalentes uma taxa de quadros de 29,76 fps. Percentualmente obtém-se um ganho de 4% em relação ao algoritmo inicial, que não apresenta este artifício.

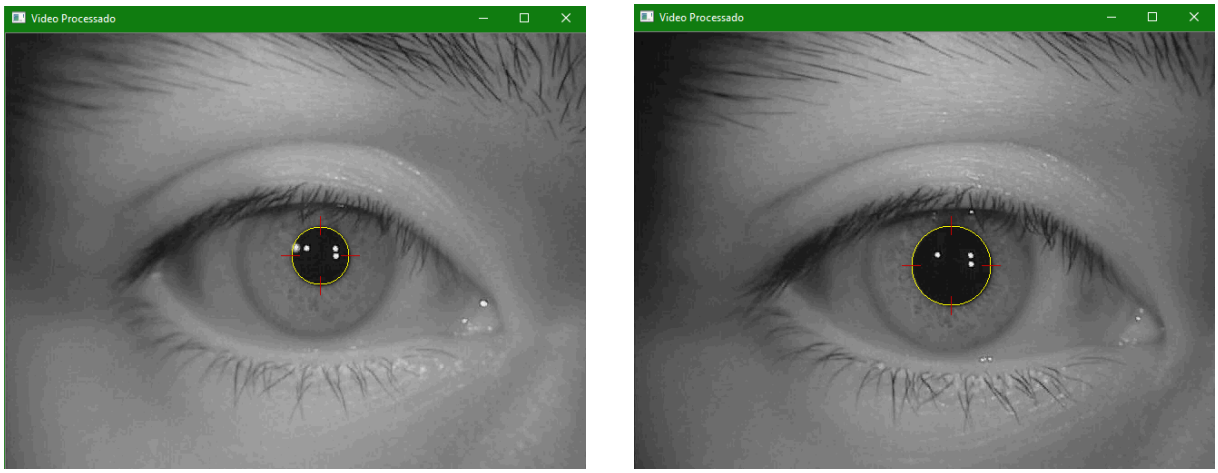


Figura 4.19: *Execução do algoritmo exibindo o raio encontrado em um frame e os limites superior e inferior de busca para o próximo frame.*

Pela Figura 4.19 pode-se perceber que a limitação dos valores de busca do raio não prejudicou a localização pupilar. A janela do programa operando duas imagens diferentes está ilustrada.

A segunda tática de aperfeiçoamento aqui sugerida consiste no corte da imagem a ser analisada de acordo com o raio eventualmente encontrado no frame anterior. O frame inicial, assim como o frame cujo anterior não resultou em raio localizado, é cortado em torno do centro da imagem. Os *frames* que sucedem localizações bem-sucedidas de circunferências são cortados em torno do centro encontrado no frame anterior, de modo a otimizar a estimação da posição pupilar de cada frame. As imagens cortadas possuem resolução de 250x250 pixels e são centradas no centro pupilar estimado.

Atingiu-se um tempo de processamento médio de 406 ms para cada sequência de imagens, equivalente a uma taxa de quadros de 49,26 fps. Isso representa 58% do tempo de processamento do algoritmo inicial (sem melhorias), ou seja, alcança-se um ganho de 42% em tempo de processamento.

Finalmente, sugere-se uma combinação dos dois aperfeiçoamentos acima mencionados,

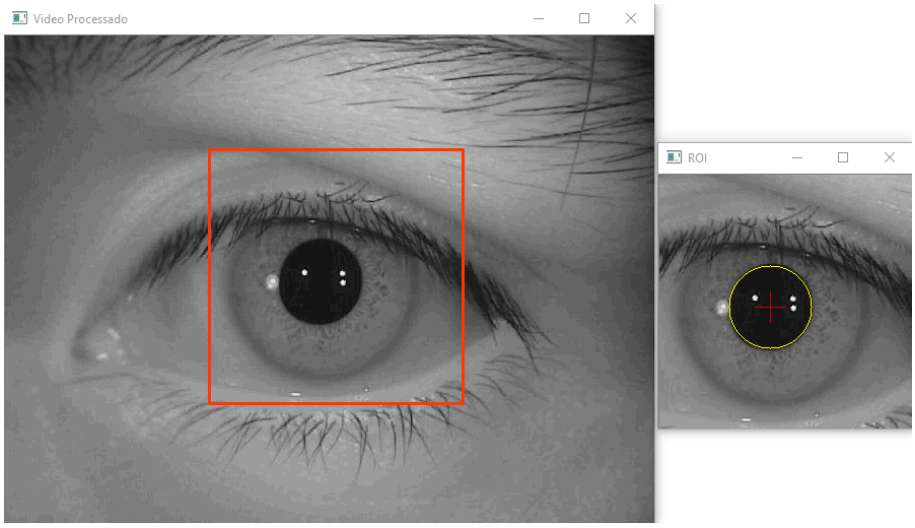


Figura 4.20: *Imagem inteira e região de corte em torno do centróide encontrado.*

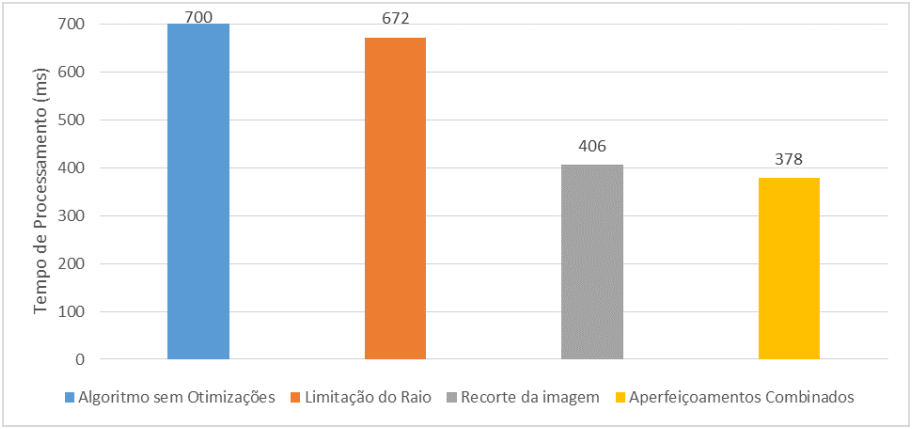


Figura 4.21: *Tempo médio de processamento para os algoritmos.*

limitando-se a região de interesse onde efetua-se a varredura e limitando-se a faixa de raios buscados pela Transformada de Hough. Esta abordagem híbrida resulta em um tempo de processamento médio de 378 ms, o que equivale a uma taxa de quadros de 52,9 fps. O ganho resultante equivale, portanto, a 46% no tempo de processamento em relação ao algoritmo originalmente desenvolvido.

A Figura 4.21 representa o tempo médio de processamento para uma sequência de 20 imagens, obtido pelos algoritmos desenvolvidos. A Figura 4.22 contém as mesmas informações em representação percentual, apontando as vantagens obtidas pelas táticas de aperfeiçoamento.

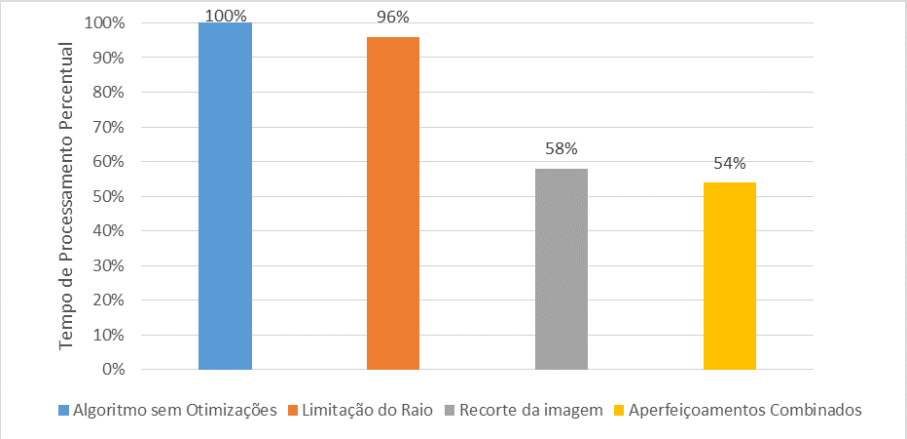


Figura 4.22: *Tempo médio percentual de processamento para os algoritmos.*

4.6.3 Teste – Redimensionamento da Imagem

Os aperfeiçoamentos implementados geraram comprovadamente resultados vantajosos ao sistema de rastreamento pupilar. A aplicação de um algoritmo de rastreamento, computacionalmente intensivo, em um dispositivo embarcado genérico sem capacidade de processamento muito elevada requer a utilização de todos os métodos possíveis de redução do tempo de processamento.

A redução crítica no tempo de processamento se deu principalmente devido ao recorte da imagem, sugerido por [2],diminuindo a área de varredura do método de localização de circunferências. Tal circunstância instiga a investigação de uma outra hipótese: O simples redimensionamento da imagem a ser processada seria vantajoso? A redução do tamanho da imagem antes de aplicá-la ao sistema causa prejuízos à detecção pupilar? Tais perguntas podem ser respondidas considerando-se o teste descrito nesta subseção.

O conjunto total de imagens constantes no banco de dados CASIA [47] – 16.212 imagens – foi submetido ao software de localização pupilar aqui desenvolvido após diferentes níveis de redimensionamento. A resolução original das imagens é de 640x480 pixels. Foi efetuado o redimensionamento resultando em imagens com metade, um terço e um quarto da resolução original, correspondendo a dimensões de 320x240, 213x160 e 160x120 respectivamente. A Figura 4.23 acima representa a imagem original e sua versão reduzida para a resolução 160x120.

Observando-se a Figura 4.24, pode-se constatar que ao se reduzir o tamanho da imagem pela metade obtemos um tempo de processamento 55% menor. Observa-se, ademais, que reduções superiores na dimensão das imagens não gera ganho suplementar em tempo de

(a) *Resolução 640x480* (b) *Resolução 160x120*

Figura 4.23: *Imagem original e versão redimensionada.*

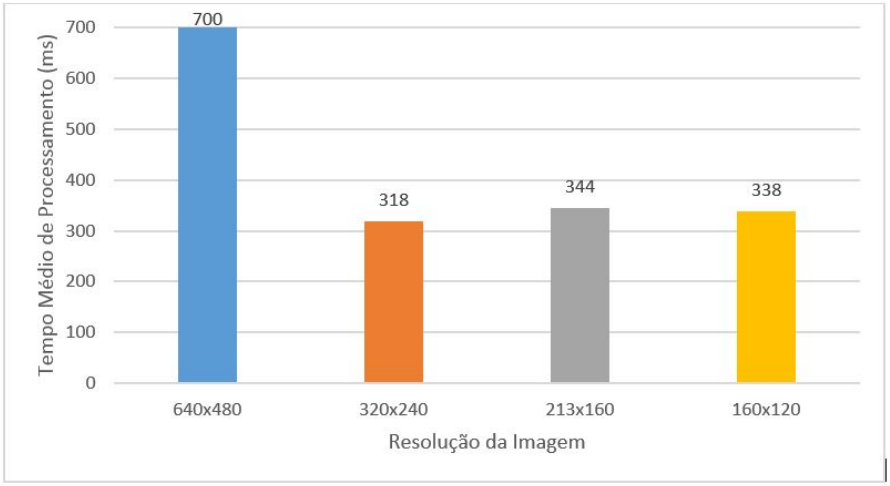


Figura 4.24: *Tempo médio de processamento obtido para imagens de diferentes resoluções.*

processamento mas causam ligeiro aumento nesse tempo.

Ao se reduzir excessivamente o tamanho da imagem, provoca-se a localização de múltiplas falsas circunferências e se torna necessário o aumento no nível de suavização precedente ao rastreamento pupilar. O tempo de processamento é, consequentemente, prejudicado dado que o processo de votação da transformada de Hough se torna ineficiente, como demostrado na Figura 3.19. Conclui-se que somente uma redução no tamanho de uma imagem pela metade seria vantajoso do ponto de vista do tempo de processamento resultante, para o banco de dados de referência [47].

Observa-se na Figura 4.25 os valores resultantes de tempo médio de processamento percentualmente. As reduções de dimensão por dois, três e quatro geraram ganho percen-

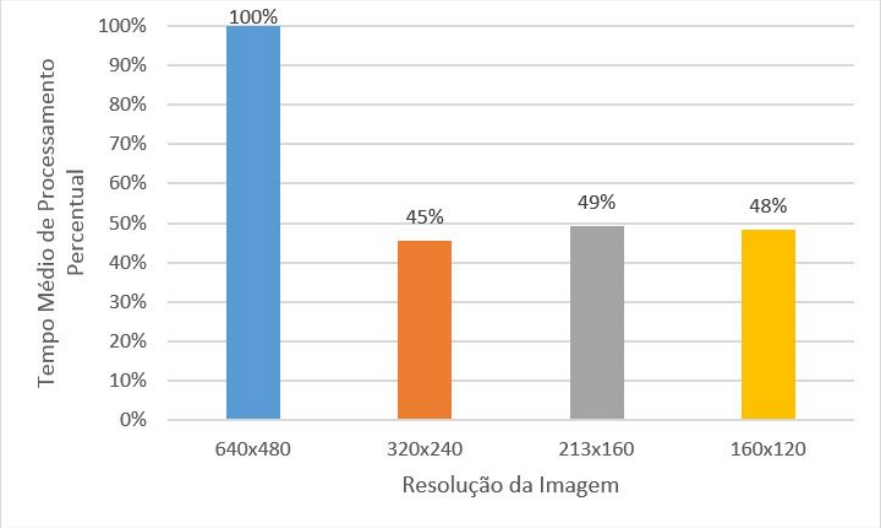


Figura 4.25: *Tempo médio percentual de processamento obtido para imagens de diferentes resoluções.*

tual de 55%, 51% e 48% respectivamente, em relação ao processamento de imagens com tamanho original.

Deve-se avaliar, no entanto, os efeitos do redimensionamento na localização da circunferência pupilar nas imagens. Ao se submeter os conjuntos de imagens redimensionadas ao programa de localização pupilar, obteve-se os resultados quantitativos descritos no gráfico da Figura 4.26.

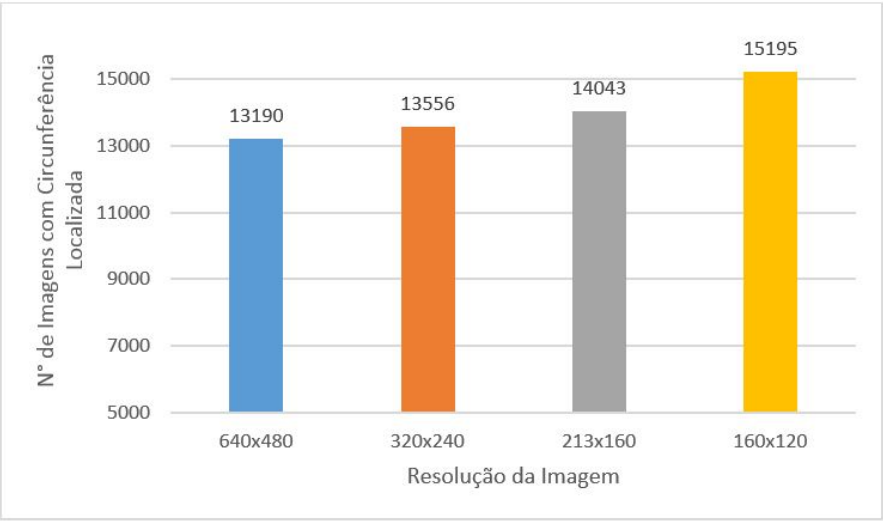
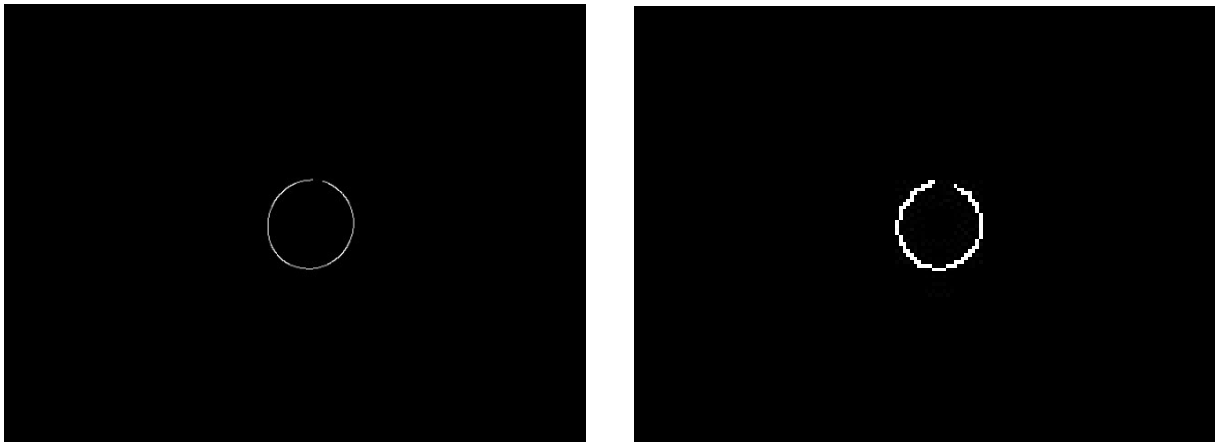


Figura 4.26: *Número de imagens com circunferências encontradas em diferentes resoluções.*

A redução da dimensão das imagens resulta em um maior número de imagens onde

foram localizadas circunferências. Essa medida não expressa qualitativamente, no entanto, a eficácia da localização pupilar. Quanto menor a resolução das imagens, mais sujeito está o sistema à detecção de falsas circunferências. A detecção de bordas da imagem sem redimensionamento e da imagem reduzida em 4 vezes está representada na Figura 4.27.



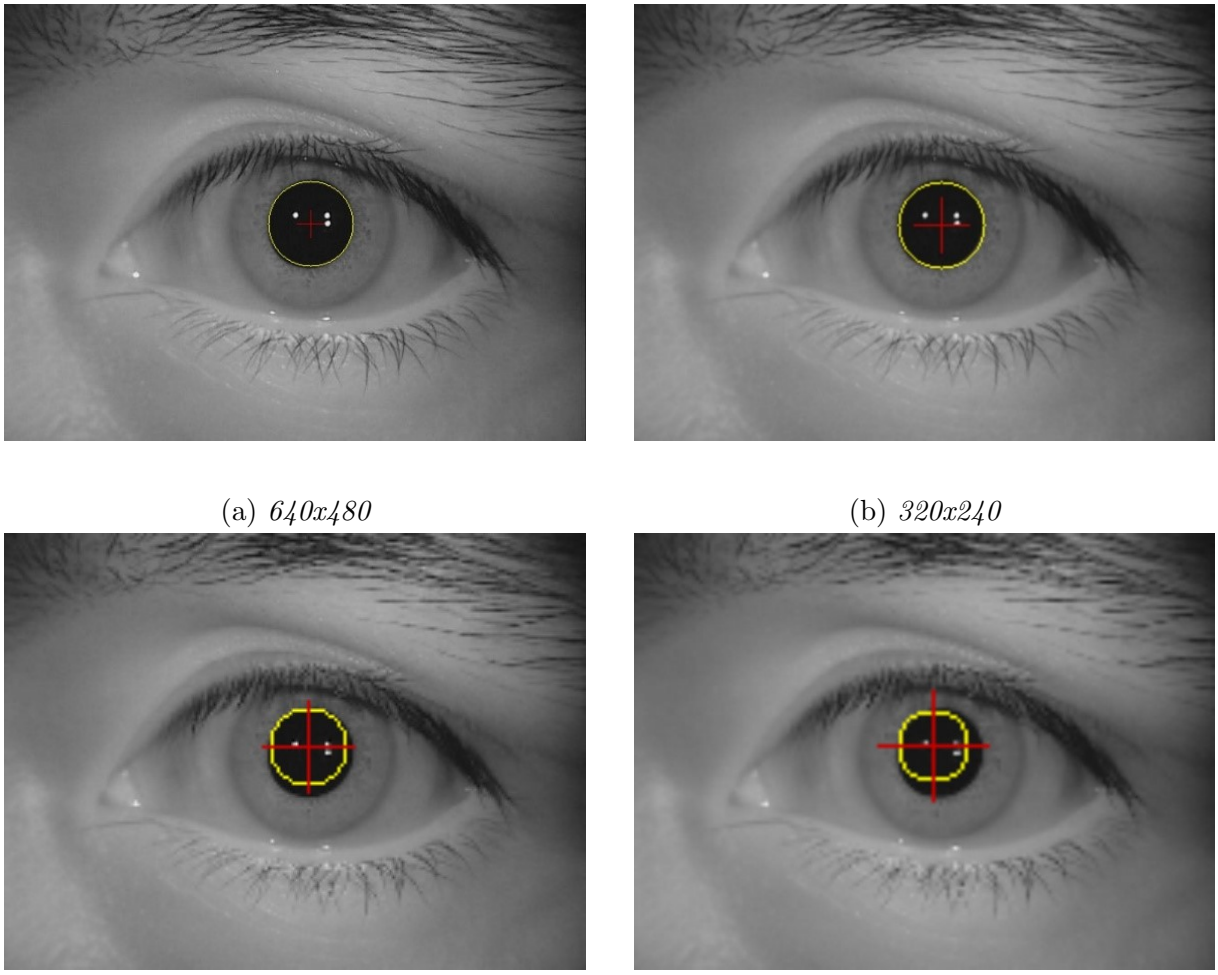
(a) *Resolução 640x480* (b) *Resolução 160x120*

Figura 4.27: *Detecção de bordas de imagem original e de imagem redimensionada.*

À medida que se diminui a resolução das imagens, perde-se precisão no rastreamento pupilar, visto que o redimensionamento ocasiona a supressão de detalhes, necessários ao reconhecimento. Ocorre o aparecimento de mais falsas circunferências que gera a necessidade de aumento no coeficiente de suavização que, por sua vez, também acarreta na redução de detalhes característicos da imagem e aumento do tempo de processamento. Na figura 4.27 nota-se a diferença de detalhes entre os dois contornos encontrados.

Conclui-se, portanto, que grandes reduções no tamanho de uma imagem prejudicam a detecção de circunferências devido à presença de falsos positivos e aumento de variações abruptas de nível de cinza, não passíveis de correção por filtros suavizadores de ordem superior. As circunferências pupilares corretas ocasionalmente encontradas em imagens com grande redução de tamanho são também deterioradas e sujeitas ao efeito granular causado pela baixa resolução da imagem.

A imagem 4.28 contém a detecção pupilar através do método de Hough para a imagem em tamanho original e suas versões reduzidas. Percebe-se claramente que as imagens redimensionadas por fator 3 e 4 geram detecções pupilares imprecisas. Tais configurações não permitiriam uma análise precisa do raio pupilar efetivo, diminuindo a resolução dos



(a) *640x480* (b) *320x240*

(c) *213x160* (d) *160x120*

Figura 4.28: *Circunferências localizadas nas imagens com diferentes resoluções.*

valores de raio possíveis.

O redimensionamento da imagem pela metade é, entretanto, vantajoso. Produz ótimos resultados no ganho em processamento ao passo em que não causa a detecção de muitas falsas circunferências. Combinações entre o redimensionamento da imagem e as técnicas de aperfeiçoamento anteriormente sugeridas podem ser capazes de gerar resultados ainda mais promissores na aplicação do algoritmo em dispositivos com capacidade regular de processamento.

4.7 Considerações Finais

Este capítulo descreveu em detalhes as características do software desenvolvido. Pretendeu-se efetuar aperfeiçoamentos em todas as etapas do sistema, desde a captura das imagens oculares ao reconhecimento e apresentação da resposta pupilar obtida.

Os testes explicitaram a importância crucial da boa definição dos parâmetros utilizados no processamento das imagens. Esses parâmetros (otimizados nesta dissertação para aplicação em imagens do banco CASIA) são fortemente variáveis e dependem do banco de dados ou dispositivo de captura utilizado.

Os resultados apresentados demostram o sucesso na otimização da acurácia de localização pupilar, obtendo-se taxa de localização pupilar de 81% para as imagens do banco de dados CASIA [47]. O aperfeiçoamento relativo ao tempo de processamento resultou na obtenção de um sistema capaz de ser executado a uma taxa de quadros superior a 50 fps nos testes efetuados em um notebook.

Capítulo 5

Protótipo Desenvolvido

O aperfeiçoamento do software apresentado nesta dissertação permitiu a obtenção de uma execução de programa em tempo real com taxa de quadros superior a 50fps em um microcomputador para imagens do banco CASIA[47] à resolução 640x480.

A próxima etapa consiste, naturalmente, no desenvolvimento de um sistema embarcado capaz de obter medidas pupilares de indivíduos em tempo real, a fim de se observar a viabilidade de um sistema integrado – etapa de captura, controle luminoso e localização pupilar em um mesmo dispositivo – e a taxa de quadros obtida em um dispositivo genérico e de baixo custo. Esse processo deve ser realizado capturando-se imagens digitais do olho humano, processando-as e retornando o diâmetro e posição pupilar para cada quadro capturado, de forma possibilitar a análise da resposta pupilar à iluminação (PLR) [12].

Foi utilizado no desenvolvimento do protótipo o sistema Raspberry Pi 2, um dispositivo integrado SBC (*Single Board Computer*) de baixo custo desenvolvido inicialmente com o intuito promover o ensino da tecnologia da informação no Reino Unido. O sensor de imagens escolhido foi a câmera Pi NoIR, comercializada pela fundação Raspberry Pi [53] e dedicada ao dispositivo.

5.1 Raspberry Pi

Ao ser lançado, o Raspberry Pi tinha o propósito de auxiliar o ensino da ciência da computação em colégios. No entanto, o interesse de cientistas, estudantes e entusiastas

levou à utilização do dispositivo em uma infinidade de projetos e aplicações em diversas áreas da ciência e sociedade. Estima-se que 5 milhões de unidades foram vendidas até fevereiro de 2015 [54].

Foram lançados posteriormente diversos modelos, entre eles o modelo A, A+, B, B+. O modelo Pi 2 é o mais recente até o momento. Este é consideravelmente mais poderoso em relação a seu antecessor e possibilita sua utilização em aplicações atuais e mais sofisticadas como por exemplo difusão de mídia, 'internet das coisas' [55] e visão computacional.

O Raspbery Pi 2 consiste em um hardware composto pelo 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*), quatro portas USB.

Figura 5.1: *Raspberry Pi 2.*

Faz-se necessário, no presente projeto, capturar imagens oculares humanas e processá-las em tempo real, a fim de se conceber um dispositivo portátil de análise pupilométrica passível de ser utilizado em futuras aplicações diagnósticas. Necessita-se, portanto, utilizar uma câmera conectada ao Raspberry Pi 2.

O emprego da câmera dedicada Pi Camera, escolhida para este projeto, traz vantagens como maior velocidade de transmissão dados, visto que utiliza uma conexão proprietária CSI (Camera Serial Interface) na placa do Raspberry Pi. Através de testes realizados constatou-se que câmeras USB conectadas ao dispositivo apresentam desempenho consideravelmente inferior, sofrendo latência e atrasos críticos para aplicações de visão computacional, uma vez que a interface CSI é capaz de transmitir taxas de dados extremamente altas contendo exclusivamente dados dos pixels capturados.

A câmera adquirida para utilização neste projeto consiste em uma versão específica da Pi Camera, chamada Pi NoIR. É idêntica ao módulo Pi Camera, com a única diferença de não efetuar a filtragem da iluminação no espectro Infravermelho. Isso nos dá a habilidade de capturar imagens nítidas iluminadas com LEDs infravermelho, mesmo na presença de pouca ou nenhuma luz visível.

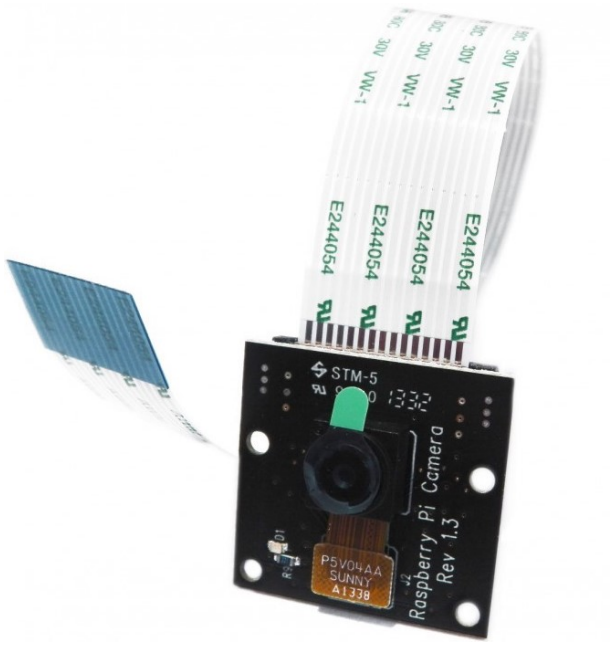


Figura 5.2: *Câmera Pi NoIR.*

Ao combinar o emprego da Pi NoIR com a utilização de LEDs infravermelho, pode-se obter imagens nítidas de um olho, ao mesmo tempo o outro olho é estimulado com luz visível, e cuja resposta será analisada pelo programa. As características da câmera utilizada incluem a captura de imagens até a resolução máxima de 2592x1944 pixels (aproximadamente 5 megapixels), vídeos à resolução máxima de 1920x1080 e taxa de quadros máxima de 90 quadros por segundo (taxa máxima de quadros, alcançada ao se capturar imagens em resolução VGA ou inferior.).

Para se desenvolver programas que utilizem a Pi Camera deve-se, no entanto, instalar a biblioteca RaspiCam [53]. Dessa forma obtêm-se total controle da câmera em programas que utilizam ou não a biblioteca OpenCV [56] de visão computacional. Os códigos que fazem uso das bibliotecas OpenCV e RaspiCam podem ser compilados normalmente utilizando comandos simples CMAKE [57] do ambiente linux/unix.

5.2 Configurando o ambiente no Raspberry Pi para desenvolvimento

A utilização do ambiente livre baseada em linux para desenvolvimento de programas requer a instalação de bibliotecas e pacotes, além de configuração de parâmetros de sistema. O sistema operacional Raspbian, biblioteca OpenCV, biblioteca RaspiCam, biblioteca mmal e biblioteca WiringPi foram utilizados no desenvolvimento do sistema pupilométrico funcional.

O Sistema Operacional Raspbian [58], utilizado no dispositivo em utilização, é uma distribuição Linux baseada na versão Debian. É otimizada para o hardware do Raspberry Pi. Essa distribuição proporciona desempenho consideravelmente superior para aplicações que demandam intenso processamento. O SO Raspbian pode ser baixado gratuitamente [58]. Outras distribuições linux podem ser utilizadas sem que se comprometa funcionalidades do sistema. Após a instalação do sistema operacional deve-se atualizar o sistema e os pacotes instalados, através dos comandos:

```
$ sudo apt-get update
$ sudo apt-get upgrade
$ sudo rpi-update
$ sudo reboot
```

Programas em linguagem C++ podem ser perfeitamente compilados e executados em plataformas Linux, uma vez instalados os compiladores e bibliotecas necessários ao desenvolvimento de um programa. O compilador CMAKE, utilizado no desenvolvimento deste projeto [57] é multiplataforma e permite aplicações que dependam de múltiplas bibliotecas. Esse compilador é bastante útil e simples, funcionando em conjunto com ambientes de compilação nativos como o make (para plataformas linux) e Microsoft Visual Studio®. Após a codificação do algoritmo, deve-se compilá-lo através do comando *'cmake .'* no diretório do projeto seguido do comando *'make'*, como demostrado abaixo. Um arquivo de configuração do projeto deve apontar as bibliotecas necessárias cujas funções são chamadas dentro da rotina.

```
$ cmake .
```

```
$ make
```

As funções específicas de processamento de imagens utilizadas neste programa estão contidas na biblioteca OpenCV. O emprego dessa biblioteca requer a instalação prévia de seus pacotes no computador onde são compilados os programas. Há versões da biblioteca para diversos sistemas operacionais [56], entre eles o Linux. Logo, é possível utilizar plenamente a biblioteca de visão computacional em um sistema baseado no Raspberry Pi. A versão 3.0 da biblioteca foi utilizada no desenvolvimento da presente aplicação. Para instalar a biblioteca OpenCV 3 seguido de algumas dependências utiliza-se os comandos abaixo:

```
$ sudo apt-get install build-essential git cmake pkg-config
$ sudo apt-get install libgtk2.0-dev
$ sudo apt-get install libatlas-base-dev gfortran
$ cd ~
$ wget -O opencv.zip https://github.com/Itseez/opencv/archive/
3.0.0.zip
$ unzip opencv.zip
$ wget -O opencv_contrib.zip https://github.com/Itseez/
opencv_contrib/archive/3.0.0.zip
$ unzip opencv_contrib.zip
$ cd ~/opencv-3.0.0/
$ mkdir build
$ cd build
$ cmake -D CMAKE_BUILD_TYPE=RELEASE \
        -D CMAKE_INSTALL_PREFIX=/usr/local \
        -D INSTALL_C_EXAMPLES=ON \
        -D INSTALL_PYTHON_EXAMPLES=ON \
        -D OPENCV_EXTRA_MODULES_PATH=~/
opencv_contrib-3.0.0/modules \
        -D BUILD_EXAMPLES=ON ..
$ make -j4
$ sudo make install
```

```
$ sudo ldconfig
```

Após executar os comandos acima, a biblioteca OpenCV 3 está pronto para uso dentro das rotinas em C++. A próxima biblioteca necessária consiste nas funções relacionadas à Pi Camera, câmera dedicada ao Raspberry Pi. Deve-se executar o comando abaixo e ativar a câmera no menu.

```
$ sudo raspi-config
```

Em seguida instala-se a biblioteca RaspiCam, contendo funções que permitem a utilização da Raspberry Camera na linguagem C++ em conjunto com OpenCV. Essa biblioteca possui limitações de desempenho, fazendo com que a taxa de quadros máxima obtida seja 30fps – bastante inferior à taxa de quadros nativa da câmera utilizada. A instalação da biblioteca RaspiCam requer o uso dos comandos abaixo:

```
$ git clone https://github.com/cedricve/raspicam .
$ cd raspicam
$ mkdir build
$ cd build
$ cmake ..
make
sudo make install
sudo ldconfig
```

O Raspberry Pi 2 possui portas de entrada e saída – GPIO (General Purpose Input Output) - que será utilizada no controle da iluminação incidente nos olhos durante a captura do ciclo pupilométrico (LEDs infravermelhos e brancos). Essas portas possuem configuração específica, incluindo saídas com nível de tensão 3,3V, 5V e GND. O esquema das portas de entrada e saída pode ser observado na Figura 5.3.

Para acessar a GPIO através de rotinas em linguagem C++, foi necessário instalar uma ferramenta de terceiros chamada Wiring Pi [59]. Essa biblioteca possibilita o acesso de registros de forma a executar o controle de circuitos externos e transferência de dados pelo software desenvolvido. A instalação dessa biblioteca é efetuada através dos comandos abaixo:

```
$ git clone git://git.drogon.net/wiringPi
```

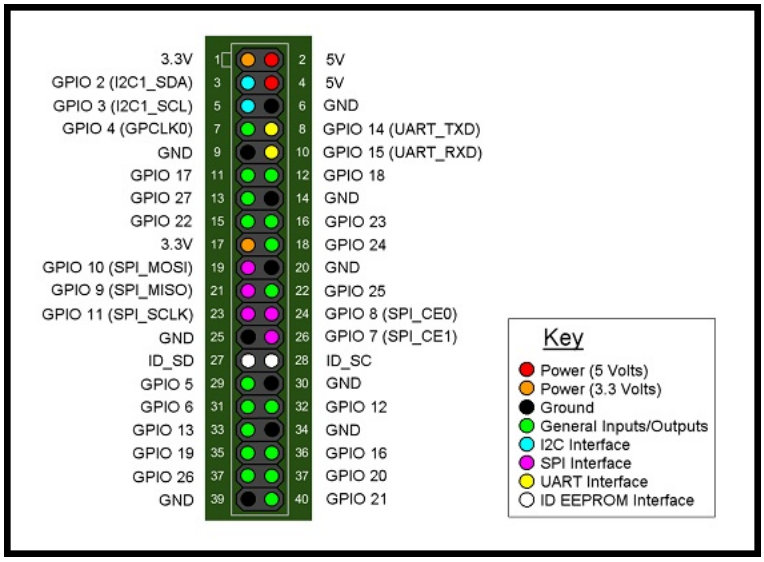


Figura 5.3: *Configuração das portas de entrada e saída no Raspberry Pi 2.*

```
$ cd wiringPi
$ git pull origin
$ cd wiringPi
$ ./build
```

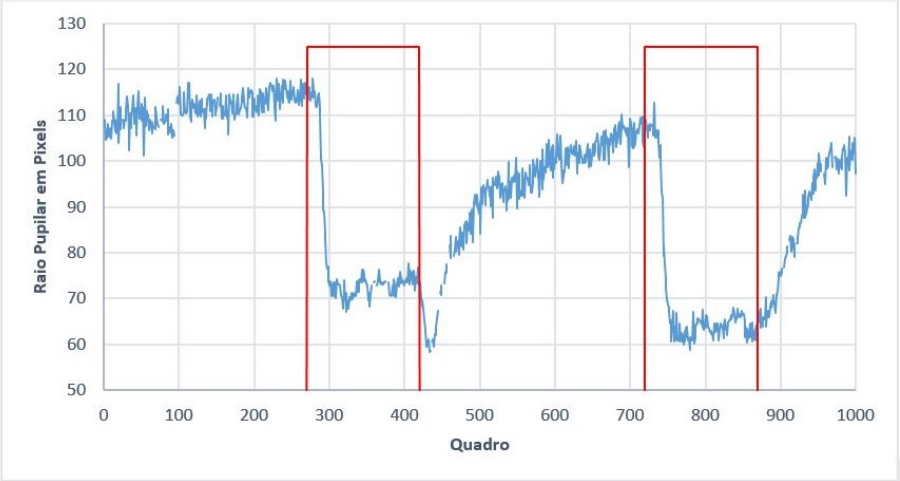
Uma vez configurados o ambiente de desenvolvimento e as bibliotecas necessárias aos algoritmos de visão computacional, foi possível efetuar a codificação e compilação dos mesmos.

5.3 Testes e Resultados

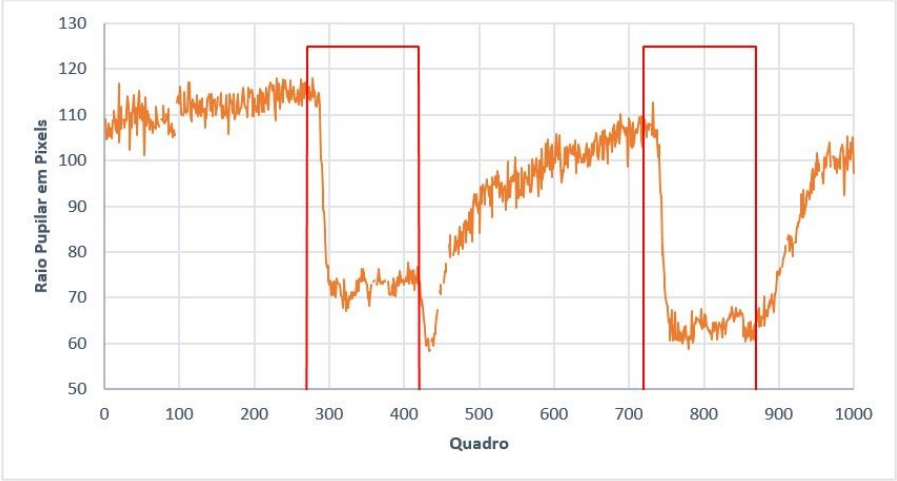
Após codificados e compilados no ambiente Linux, os arquivos executáveis dos programas desenvolvidos e compilados podem ser facilmente utilizados em outros dispositivos semelhantes. Vários programas foram desenvolvidos, explorando-se diferentes módulos e capacidades de hardware e software. Esse processo resultou em dois softwares embarcados no Raspberry Pi 2 com aplicações distintas, como descrito a seguir.

5.3.1 Primeiro Software – Análise Pupilar de Imagens do Banco de Dados

O primeiro software desenvolvido recebe como entrada um arquivo de vídeo contendo imagens oculares previamente capturadas – provenientes de bancos de dados existentes ou capturas previamente efetuadas. O software retorna as medidas do raio pupilar e posição de seu centro. Todas as versões dos algoritmos, como descritos nas seções 4.3 e 4.4 foram modificados e recompilados no ambiente Linux (arquitetura ARMv7) e funcionam perfeitamente no dispositivo embarcado, havendo evidentemente diferenças no tempo de processamento em relação à execução no ambiente Windows® (arquitetura x64).



(a) *Execução no microcomputador*



(b) *Execução no Raspberry Pi 2*

Figura 5.4: *Comparação entre a resposta pupilar obtida pelo software.*

A Figura 5.4 representa a comparação entre resposta pupilar obtida a partir de imagens

do banco de dados de imagens de PLR LAVI Iris DB2 [48], descrito na subseção 3.3.2 . O programa foi executado em um microcomputador e no Raspberry Pi 2 de forma a possibilitar a análise comparativa.

Percebe-se que a resposta obtida é idêntica para os dois sistemas, independentemente da arquitetura portátil, de baixo custo e poder de processamento inferior do sistema embarcado em relação a um PC.

A tabela 5.1 apresenta os resultados de posição (coordenadas x e y) e raio pupilar encontrados pelo software na versão para microcomputador e para Raspberry Pi. Observa-se que os resultados são idênticos, ou seja, o sistema embarcado possui a mesma eficácia na localização pupilar que o software para plataformas x86 ou x64.

Pode-se identificar na resposta pupilar apresentada a contração pupilar correspondente à ativação dos estímulos luminosos (representados em vermelho) em torno dos quadros de posições 275 e 730, seguidos da dilatação pupilar após o estímulo luminoso ser desligado. Conforme observado na Figura 3.24 as contrações pupilares sucedem exatamente os instantes 18 e 48 segundos, dada a taxa de quadros de 15fps utilizada.

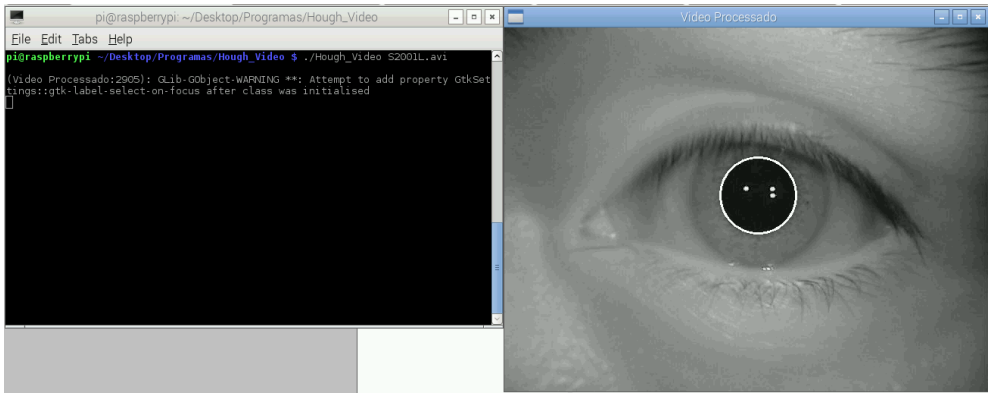


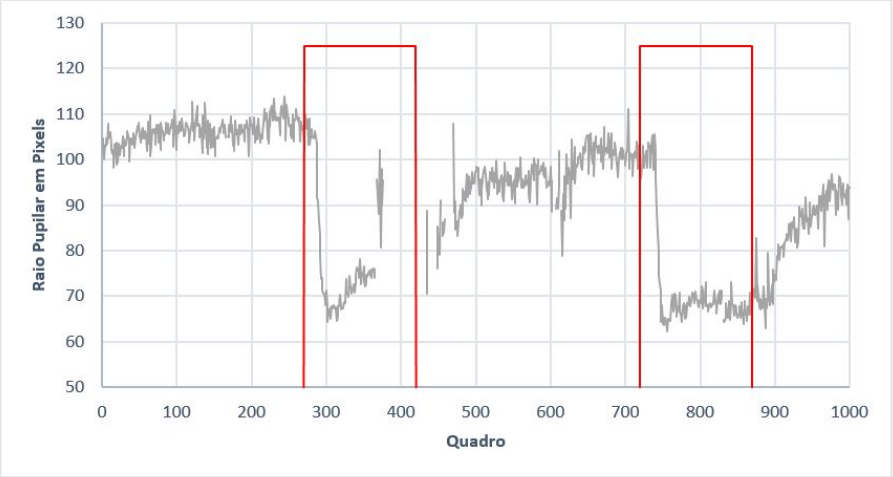
Figura 5.5: *Execução da análise pupilométrica a partir de sequências de vídeo em ambiente Linux no Raspberry PI 2.*

Com relação ao tempo de processamento, observa-se que o sistema embarcado leva 14,6 minutos para efetuar o processamento de uma sequência de vídeo de 1 minuto e seis segundos – 1000 quadros a 15fps – dada a resolução de 1024x768 e o microcomputador toma 0,95 min na execução da mesma tarefa, ou seja, aproximadamente 15 vezes mais rápido. Foi utilizado para este teste, entretanto, a versão do algoritmo não otimizada em tempo de processamento, descrito na seção 4.3 .

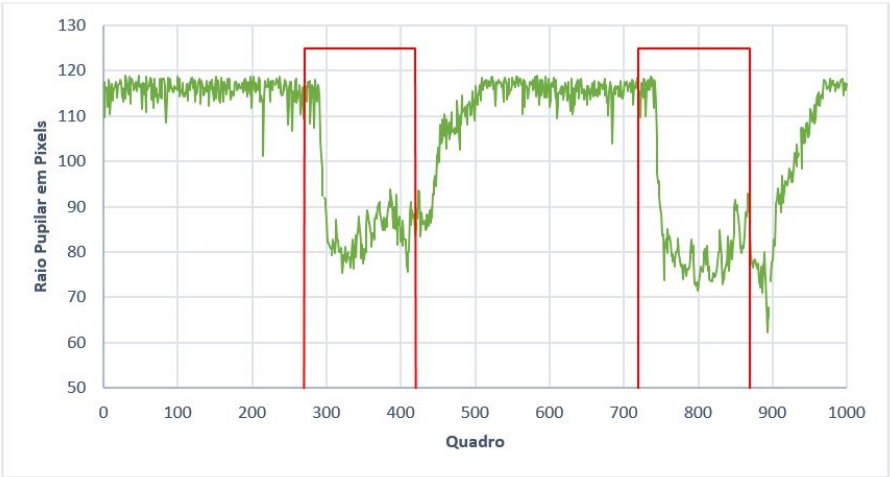
As figuras 5.6 e 5.7 representam as análises pupilométricas efetuadas pelo software

Tabela 5.1: Posição e raio encontrados no PC e no Raspberry Pi

Posição e raio pupilar encontrados					
Coordenada x		Coordenada y		Raio	
PC	Raspberry	PC	Raspberry	PC	Raspberry
442,5	442,5	532,5	532,5	109,08	109,08
433,5	433,5	538,5	538,5	108,777	108,777
427,5	427,5	532,5	532,5	104,721	104,721
433,5	433,5	541,5	541,5	105,14	105,14
436,5	436,5	541,5	541,5	108,058	108,058
436,5	436,5	541,5	541,5	106,745	106,745
439,5	439,5	547,5	547,5	105,33	105,33
439,5	439,5	544,5	544,5	107,231	107,231
436,5	436,5	541,5	541,5	105,643	105,643
436,5	436,5	544,5	544,5	104,912	104,912
439,5	439,5	541,5	541,5	108,805	108,805
436,5	436,5	544,5	544,5	109,108	109,108
439,5	439,5	538,5	538,5	110,908	110,908
430,5	430,5	541,5	541,5	108,307	108,307
433,5	433,5	541,5	541,5	106,388	106,388
436,5	436,5	544,5	544,5	108,979	108,979
436,5	436,5	538,5	538,5	105,965	105,965
436,5	436,5	544,5	544,5	107,51	107,51
448,5	448,5	544,5	544,5	116,818	116,818
448,5	448,5	520,5	520,5	104,041	104,041
448,5	448,5	529,5	529,5	108,998	108,998
442,5	442,5	523,5	523,5	106,942	106,942
445,5	445,5	529,5	529,5	111,788	111,788
x	x	x	x	x	x
430,5	430,5	499,5	499,5	108,915	108,915
430,5	430,5	502,5	502,5	107,706	107,706
427,5	427,5	502,5	502,5	107,147	107,147
433,5	433,5	505,5	505,5	109,007	109,007
433,5	433,5	505,5	505,5	108,759	108,759
430,5	430,5	511,5	511,5	111,079	111,079
445,5	445,5	511,5	511,5	113,985	113,985
439,5	439,5	514,5	514,5	111,949	111,949
433,5	433,5	511,5	511,5	105,7	105,7
433,5	433,5	520,5	520,5	102,296	102,296
436,5	436,5	514,5	514,5	110,429	110,429
433,5	433,5	511,5	511,5	107,203	107,203
430,5	430,5	514,5	514,5	109,51	109,51



(a) *Resposta pupilar (em cinza) a pulsos luminosos (em vermelho) na sequência em vídeo AAS_S2.avi.*



(b) *Resposta pupilar (em verde) a pulsos luminosos (em vermelho) na sequência em vídeo AFN_S2.avi.*

Figura 5.6: *Respostas pupilares de sequências de vídeo do banco Iris DB2.*

para mais duas sequências de vídeo do banco LAVI Iris DB2 [48]. O código fonte do programa está disposto no apêndice desta dissertação.

Na tabela 5.2 encontra-se as medidas extraídas do ciclo pupilométrico, a partir dos resultados gerados pelo software. AS sequências de vídeo são pertencentes ao banco de dados LAVI Iris DB2 [48]. Os parâmetros mensurados são:

- (Pa/Ia) é a razão entre raio da pupila e raio da íris antes do disparo do flash;
- τ_a é o tempo de latência entre o disparo do flash e o início da constrição pupilar (em num. de quadros);
- (Pb/Ib) é a razão entre raio da pupila e raio da íris para a maior constrição;
- τ_b é o tempo de latência para a maior constrição (em num. de quadros);
- τ_p é o tempo de latência de alcance do plateau (P=75% do valor inicial (em num. de quadros);
- AMP é a amplitude do movimento pupilar (em pixels);
- R_{min} é o raio pupilar mínimo (em pixels).

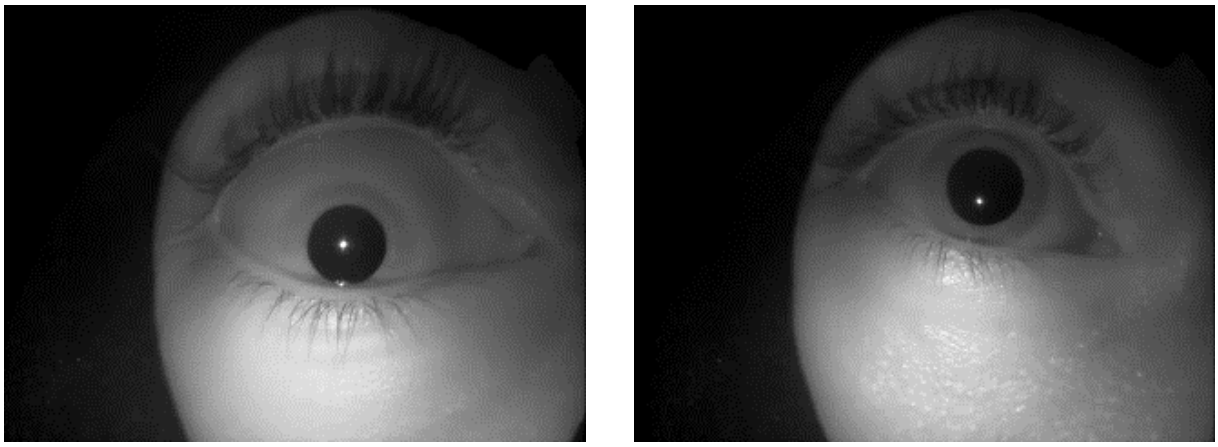
Tabela 5.2: Exemplos de medidas pupilométricas obtidas para vídeos do banco Iris DB2

Vídeo	Pa/Ia	Pb/Ib	Pb/Ib	τ_a	τ_b	τ_p	AMP	R_{min}
AAS_S1	0,5115	0,31787	0,31787	18	55	105	217,2	67,8
AAS_S2	0,5071	0,335	0,335	18	45	90	194,4	65,6
ADG_S1	0,5224	0,31762	0,31762	19	52	106	188,63	76,37
AFN_S2	0,5784	0,35619	0,35619	21	51	126	185,7	75,3

5.3.2 Segundo Software – Captura das Imagens e Análise Pupilar Simultânea

O segundo programa foi concebido para controlar a Pi Camera, de forma a capturar e processar em tempo real a sequência em vídeo, retornando posição e raio de possíveis circunferências pupilares presentes em cada frame do fluxo de entrada da câmera.

A imagem 5.7 ilustra a aplicação do software, que funciona de forma a localizar circunferências na imagem. Para que o protótipo seja adequado para captura de imagens oculares é necessário, entretanto, instalar a câmera juntamente com o Raspberry Pi em uma unidade contendo uma lente de forma a justar a distância focal entre câmera e olho.



(a) *Amostra 1;* (b) *Amostra 2;*

Figura 5.7: *Captura de imagem da câmera durante execução do software no dispositivo Raspberry Pi 2.*

O algoritmo resultante permite a aferição do ciclo pupilar em tempo real com taxa de quadros dependente da resolução escolhida. Isso representa grande vantagem, visto que se trata de um dispositivo portátil não dedicado ao processamento de vídeo e que consome potência média de 1 W. Obteve-se um funcionamento do programa com taxa de quadros em torno de 10 fps ao se utilizar a resolução 320x240.

O programa utiliza diversas bibliotecas de código aberto para controle dos periféricos. A biblioteca OpenCV [56] contém as operações de processamento de imagem e vídeo, enquanto a biblioteca WiringPi é necessária para se acessar as portas de entrada e saída do dispositivo a partir de uma aplicação compilada. A biblioteca RaspiCam [53] e mmal realizam a interface entre a câmera acoplada e os comandos OpenCV.

A utilização da biblioteca RaspiCam é obrigatória mas é prejudicial, no entanto, para a execução do algoritmo em tempo real. Ao se utilizar a biblioteca, a taxa de quadros fica limitada a 30 fps, e a resolução máxima é de 1280x960. Ao se aplicar o processamento necessário para a localização pupilar, obtêm-se o registro pupilométrico à taxa de quadros média de 10 fps para resolução de 320x240, ainda que a taxa de atualização exibida na interface do programa seja maior.

A limitação em tempo de processamento é causada em grande parte por limitações nas bibliotecas de código aberto utilizadas. Há, no entanto a possibilidade de aperfeiçoamento dessas ferramentas a depender de equipes de desenvolvimento e da comunidade que apóia o aperfeiçoamento de ferramentas livres.

A utilização de resoluções superiores a VGA causa redução na velocidade de execução do programa, uma vez que a quantidade de pixels a ser varrida pelo algoritmo aumenta. A unidade de processamento gráfico dedicada (GPU) efetua grande parte dos cálculos necessários e não sobrecarrega a unidade central de processamento (CPU).

As etapas de projeto até aqui concretizadas possibilitam o desenvolvimento do dispositivo de captura, que se resume no controle da iluminação de LEDs, necessária à análise do PLR (*Pupil Light Reflex*). Às portas de entrada e saída (GPIO) do Raspberry Pi 2 foram conectados LEDs brancos com intensidade luminosa controlada pelo algoritmo – a iluminação de um olho produz resposta no olho não estimulado do indivíduo – e LEDs infravermelho responsáveis pela iluminação do olho a ser filmado pelo dispositivo, uma vez que a faixa do espectro do infravermelho próximo (*Near Infrared* - NIR) não causa interferência ou estímulo visual nos humanos.

O algoritmo funcional para realização do PLR efetua o controle da intensidade luminosa dos LEDs através da biblioteca Wiring Pi [59]. O protótipo funcional é capaz, finalmente, de capturar imagens oculares obtidas sob controle dinâmico de intensidade luminosa incidente nos olhos e efetuar o processamento do fluxo de vídeo em tempo real, retornando o diâmetro pupilar encontrado e posição de seu centro.

Pode-se configurar a iluminação dinâmica como se deseje, utilizando-se estímulos do tipo degrau ou gradual, por exemplo. Como descrito em [4], a resposta pupilar obtida depende diretamente da intensidade luminosa utilizada em um estímulo do tipo degrau – o protótipo permite a variação de intensidade de 0 a 100

O estímulo luminoso utilizado na aplicação consiste em um pulso retangulares efetuado

Figura 5.8: *Protótipo de captura e controle simultâneo do estímulo luminoso utilizando Raspberry Pi 2.*

por um LED branco. A captura do ciclo pupilar é efetuada através da câmera dedicada raspicam na faixa do infravermelho próximo, pelo dispositivo exibido na Figura 5.8. Um LED infravermelho garante a captura do reflexo síncrono do outro olho sem que haja interferência na resposta ou reflexos que possam prejudicar a detecção pupilar.

Capítulo 6

Conclusões, Contribuições e Trabalhos Futuros

6.1 Introdução

O desenvolvimento deste trabalho resultou na concepção de um protótipo capaz de desempenhar não somente a etapa de captura de ciclos pupilares utilizando controle luminoso ativo, mas também capaz de efetuar o processamento da sequência de vídeo, retornando a resposta pupilar aferida.

A concentração 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 - e de baixo custo. Não há a necessidade da utilização de circuitos externos para controle da iluminação ou software extra para sincronização dos quadros com os impulsos luminosos.

O sistema desenvolvido permite a configuração do estímulo luminoso desejado e a fácil obtenção da resposta pupilar associada com acurácia, permitindo sua utilização em diversos testes diagnósticos.

Reitera-se a não utilização de códigos de terceiros, integral ou parcialmente, e a aplicação de ferramentas livres de software e hardware. Isso permite a disponibilização livre e gratuita das aplicações desenvolvidas.

Demostrou-se a importância da boa definição dos parâmetros utilizados na filtragem das imagens, na detecção de bordas e na aplicação da Transformada de Hough. Esses parâmetros são cruciais para o bom desempenho do algoritmo e variam de acordo com o banco de dados utilizado, dispositivo utilizado para a captura e características das imagens submetidas ao software.

6.2 Principais Contribuições

A principal contribuição deste trabalho consiste na migração de um algoritmo capaz de aferir a resposta pupilar a luz (PLR) a partir de uma sequência de vídeo para plataformas livres e não proprietárias como a linguagem de programação C++ e ferramentas abertas como a biblioteca OpenCV, amplamente empregada em aplicações com diversas finalidades.

O software final é multiplataforma, possuindo compilações que podem ser executadas em ambientes Windows® ou Linux.

O desenvolvimento do software em plataforma livre possibilitou ainda outra contribuição significativa: a construção de um protótipo com o software pupilométrico embarcado capaz de efetuar o exame de forma automática e com desempenho satisfatório, viabilizado através de aperfeiçoamentos de algoritmo também sugeridos nesta dissertação.

6.3 Publicações

R. A. da Silva, A. G. C. Dias, A. C. P. Veiga, M. B. P. Carneiro. COMPARAÇÃO DE DIFERENTES MÉTODOS DE RASTREAMENTO DE CARACTERÍSTICAS DA PUPILA HUMANA. XII CEEL – ISSN 2178-8308 (CEEL–2014, CD-ROM). Outubro de 2014. Uberlândia, MG – Brasil.

6.4 Trabalhos Futuros

As ferramentas apresentadas neste trabalho apresentaram desempenho satisfatório e satisfazem os objetivos inicialmente propostos. Existem, no entanto, possibilidades de

investigação complementares a serem exploradas.

O protótipo desenvolvido pode ser melhorado, de forma a torna-lo mais ergonômico e adequado ao uso continuo. A tecnologia de impressão 3D pode representar uma possibilidade no projeto de um visor que acomode todos os elementos de hardware, incluindo alimentação, tornando o dispositivo completamente portátil.

Algumas características do algoritmo são passíveis de aperfeiçoamento adicional, como por exemplo a determinação dinâmica do tamanho da janela de corte da imagem através do diâmetro pupilar encontrado em quadros imediatamente precedentes. Isso pode resultar em melhorias do desempenho do software em tempo de processamento.

A utilidade do sistema aqui desenvolvido em testes clínicos reais deve ainda ser avaliada em trabalhos futuros, assim como a criação de um banco de imagens que possa ser utilizado em investigações relacionadas ao diagnóstico através do PLR.

Diversos trabalhos científicos podem ser desenvolvidos aplicando-se o protótipo desenvolvido em outros procedimentos diagnósticos que utilizam imagens oculares. O dispositivo poderia ser, por exemplo, útil na realização do exame de fundo de olho, capaz de identificar enfermidades como diabetes e hipertensão arterial.

Adicionalmente podem ser exploradas implementações do algoritmo em ferramentas de hardware capazes de oferecer melhor desempenho no processamento massivo de imagens requerido pelo algoritmo, visto que mesmo máquinas com alto poder de processamento podem não ser capazes de executar o programa em tempo real em imagens com resolução mais elevada. Dispositivos como DSPs ou mesmo FPGAs possivelmente obterão desempenho superior.

Referências Bibliográficas

[1] HACHOL, A.; SZCZEPANOWSKA-NOWAK, W.; KASPRZAK, H.; ZAWOJSKA, I.; DUDZINSKI, A.; KINASZ, R.; WYGLEDOWSKA-PROMIENSKA, D. Measurement of pupil reactivity using fast pupillometry. *Physiological measurement*, v. 28, n. 1, p. 61, 2007.

[2] DA COSTA DIAS, A. G. *Pupilometria dinÃmica: Uma proposta de rastreamento da pupila humana em tempo real.* 2013. Dissertação (Mestrado em Física) - Universidade Federal de UberlÃÂndia, 2013.

[3] TRESADERN, P. A.; IONITA, M. C.; COOTES, T. F. Real-time facial feature tracking on a mobile device. *International Journal of Computer Vision*, v. 96, n. 3, p. 280–289, 2012.

[4] BERNADELLI, C. R. *Rastreamento em vídeo das características da pupila.* 2011. Dissertação (Mestrado em Física) - Universidade federal de UberlÃÂndia, 2011.

[5] BOVIK, A. C. *The essential guide to video processing.* 2nd. ed. Academic Press, 2009.

[6] BERNADELLI, C.; VEIGA, A.; FLORES, E.; PEREIRA, M. Iris motion tracking-using feature extraction by shape matching. In: . c2011. p. 122–127.

[7] SOLTANY, M.; ZADEH, S. T.; POURREZA, H.-R. Fast and accurate pupil positioning algorithm using circular hough transform and gray projection. In: . c2011. v. 5. p. 556–561.

[8] WIBIRAMA, S.; TUNGJITKUSOLMUN, S.; PINTAVIROOJ, C.; HAMAMOTO, K. Real time eye tracking using initial centroid and gradient analysis technique. In: . c2009. v. 2. p. 1054–1057.

[9] CIESLA, M.; KOZIOL, P. Eye pupil location using webcam. *arXiv preprint arXiv:1202.6517*, 2012.

[10] JIAN-NAN, C.; CHUANG, Z.; YAN-JUN, Q.; YING, L.; LI, Y. Pupil tracking method based on particle filtering in gaze tracking system. *International Journal of the Physical Sciences*, v. 6, n. 5, p. 1233–1243, 2011.

[11] PAMPLONA, V. F.; OLIVEIRA, M. M.; BARANOSKI, G. V. Photorealistic models for pupil light reflex and iridal pattern deformation. *ACM Transactions on Graphics (TOG)*, v. 28, n. 4, p. 106, 2009.

[12] FOTIOU, F.; FOUNTOULAKIS, K.; GOULAS, A.; ALEXOPOULOS, L.; PALIKARAS, A. Automated standardized pupillometry with optical method for purposes of clinical practice and research. *Clinical Physiology*, v. 20, n. 5, p. 336–347, 2000.

[13] LINK, N.; STARK, L. Latency of the pupillary response. *Biomedical Engineering, IEEE Transactions on*, v. 35, n. 3, p. 214–218, 1988.

[14] TAN, E.; LAMBIE, D.; JOHNSON, R.; WHITESIDE, E. Parasympathetic denervation of the iris in alcoholics with vagal neuropathy. *Journal of Neurology, Neurosurgery & Psychiatry*, v. 47, n. 1, p. 61–64, 1984.

[15] GRÜNBERGER, J.; LINZMAYER, L.; FODOR, G.; PRESSLICH, O.; PRAITNER, M.; LOIMER, N. 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.

[16] ROSSE, R. B.; ALIM, T. N.; JOHRI, S. K.; HESS, A. L.; DEUTSCH, S. I. Anxiety and pupil reactivity in cocaine dependent subjects endorsing cocaine-induced paranoia: preliminary report. *Addiction*, v. 90, n. 7, p. 981–984, 1995.

[17] SACKS, B.; SMITH, S. People with down's syndrome can be distinguished on the basis of cholinergic dysfunction. *Journal of Neurology, Neurosurgery & Psychiatry*, v. 52, n. 11, p. 1294–1295, 1989.

[18] FOTIOU, F.; FOUNTOULAKIS, K.; TSOLAKI, M.; GOULAS, A.; PALIKARAS, A. Changes in pupil reaction to light in alzheimerÃ??s disease patients: a preliminary report. *International journal of psychophysiology*, v. 37, n. 1, p. 111–120, 2000.

[19] 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. 253–262, 1996.

[20] GRANHOLM, E.; MORRIS, S.; GALASKO, D.; SHULTS, C.; ROGERS, E.; VUKOV, B. 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.

[21] RIZOS, G.; TSALAMAS, C.; FOTIOU, D.; TZAMBATZAKIS, A.; GOULAS; TSIPTSIOS, I.; FOTIOU, F. Pupillometry in parkinson's disease correlations with neuroimaging techniques. In: . c2004. v. 54. p. 41–42.

[22] KEIVANIDOU, A.; FOTIOU, D.; ARNAOUTOGLOU, C.; ARNAOUTOGLOU, M.; FOTIOU, F.; KARLOVASITOU, A. 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.

[23] EUSTACE, P.; MURNAGHAN, S.; DRURY, M. Pupil constriction to dilute pilocarpine: a useful clinical sign of autonomic involvement in diabetic neuropathy. *Neurogenetics and Neuro-Ophthalmology*, p. 247–250, 1981.

[24] MACLEAN, H.; DHILLON, B. Pupil cycle time and human immunodeficiency virus (hiv) infection. *Eye-Transactions of the OSUK*, v. 7, n. 6, p. 785–786, 1993.

[25] FAN, X.; MILES, J. H.; TAKAHASHI, N.; YAO, G. Abnormal transient pupillary light reflex in individuals with autism spectrum disorders. *Journal of autism and developmental disorders*, v. 39, n. 11, p. 1499–1508, 2009.

[26] FERRARI, G. L.; MARQUES, J. L.; GANDHI, R. A.; EMERY, C. J.; TESFAYE, S.; HELLER, S. R.; SCHNEIDER, F. K.; GAMBA, H. R. An approach to the assessment of diabetic neuropathy based on dynamic pupillometry. In: . c2007. p. 557–560.

[27] BERNABEI, M.; ROVATI, L.; PERETTO, L.; TINARELLI, R. A simple portable polychromatic pupillometer for human eye annoyance measurement. In: . c2014. p. 1207–1211.

[28] DE SOUZA, J. K. S.; DA SILVA PINTO, M. A.; VIEIRA, P. G.; BARON, J.; TIERRA-CRIOLLO, C. J. An open-source, firewire camera-based, labview-controlled image acquisition system for automated, dynamic pupillometry and blink detection. *Computer methods and programs in biomedicine*, v. 112, n. 3, p. 607–623, 2013.

[29] HOVAGIMIAN, H.; WALKER, C.; GRANQUIST-FRASER, D.; TURKOVICH, J. Development of a remote pupillometer system for non-invasive, distant analysis. In: . c2012. p. 153–154.

[30] BERNADELLI, C. R.; VEIGA, A. C. P.; CARNEIRO, M. B.; FLÔRES, E. L. Sistema embarcado para controle da iluminação na aquisição de ciclos pupilares. *XII CEEL*, 2014.

[31] GONZALEZ, R. C.; WOODS, R. E. *Processamento de imagens digitais*. Edgard Blucher, 2000.

[32] MOESLUND, T. B. *Image and video processing*. Institut for Medieteknologi, Aalborg Universitet, 2009.

[33] LAGANIÈRE, R. *Opencv 2 computer vision application programming cookbook: Over 50 recipes to master this library of programming functions for real-time computer vision*. Packt Publishing Ltd, 2011.

[34] NIXON, M. *Feature extraction & image processing*. Academic Press, 2008.

[35] DAWSON-HOWE, K. *A practical introduction to computer vision with opencv*. John Wiley & Sons, 2014.

[36] CARNEIRO, M. B. P.; VEIGA, A. C. P.; FLÔRES, E. L.; CARRIJO, G. A. Aplicação de active shape models para segmentar a região da íris em imagens de olho.

[37] PARIS, S.; KORNPROBST, P.; TUMBLIN, J.; DURAND, F. *Bilateral filtering: Theory and applications*. Now Publishers Inc, 2009.

[38] TOMASI, C.; MANDUCHI, R. Bilateral filtering for gray and color images. In: . c1998. p. 839–846.

[39] DE QUEIROZ, J. E. R.; GOMES, H. M. Introdução ao processamento digital de imagens. *RITA*, v. 13, n. 2, p. 11–42, 2006.

[40] CANNY, J. A computational approach to edge detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, , n. 6, p. 679–698, 1986.

[41] DUDA, R. O.; HART, P. E. et al. *Pattern classification and scene analysis*. Wiley New York, 1973. v. 3.

[42] HART, P. E. How the hough transform was invented [dsp history]. *Signal Processing Magazine, IEEE*, v. 26, n. 6, p. 18–22, 2009.

[43] VC, H. P. Method and means for recognizing complex patterns, Dec. 18 1962. US Patent 3,069,654.

[44] ANTOLOVIC, D. Review of the hough transform method. *With an Implemen-tation of the Fast Hough Variant for Line Detection,"Department of Computer Science, Indiana University, and IBM Corporation*.

[45] HASSANEIN, A. S.; MOHAMMAD, S.; SAMEER, M.; RAGAB, M. E. A survey on hough transform, theory, techniques and applications. *arXiv preprint arXiv:1502.02160*, 2015.

[46] PEDERSEN, S. J. K. Circular hough transform. *Aalborg University, Vision, Graphics, and Interactive Systems*, 2007.

[47] CASIA. Casia-iris-v3. http://biometrics.idealtest.org. [Web, acessado em 06/07/2015].

[48] LAVI. Iris db2. http://iris.sel.eesc.usp.br. [Web, acessado em 18/11/2015].

[49] LEAVERS, V. F. *Shape detection in computer vision using the hough transform*. Springer, 1992.

[50] SONKA, M.; HLAVAC, V.; BOYLE, R. *Image processing, analysis, and machine vision*. Cengage Learning, 2014.

[51] MENDES, A. Arquitetura de software: desenvolvimento orientado para arquitetura. *Rio de Janeiro: Campus*, 2002.

[52] BRADSKI, G.; KAEHLER, A. *Learning opencv: Computer vision with the opencv library*. "O'Reilly Media, Inc.", 2008.

[53] Raspberry Pi Foundation. Raspicam documentation. http://www.raspberrypi.org/documentation/hardware/camera.md. [Web, acessado em 18/11/2015].

[54] UPTON, E.; HALFACREE, G. *Raspberry pi user guide*. John Wiley & Sons, 2014.

[55] DENNIS, A. K. *Raspberry pi home automation with arduino*. Packt Publishing Ltd, 2013.

[56] OPENCV. Open source computer vision. http://opencv.org. [Web, acessado em 17/10/2015].

[57] CMAKE. Documentation cmake. http://cmake.org. [Web, acessado em 15/10/2015].

[58] RASPBIAN. Raspbian so. https://www.raspbian.org. [Web, acessado em 11/10/2015].

[59] Wiring Pi. Gpio interface library for the raspberry pi. http://wiringpi.com. [Web, acessado em 18/11/2015].

Apêndice A

Códigos fonte

A.1 Rastreamento pupilar em sequências de vídeo

Programa de rastreamento pupilar que, efetua busca de circunferência pupilar em imagem suavizada pelo filtro de mediana; Método aplicado a vídeo derivado de imagens do banco de dados LAVI Iris DB2, descrito na subseção 3.3.

Requer bibliotecas:

- OpenCV

```
#include <opencv2/opencv.hpp>
#include <iostream>
#include <fstream>

using namespace std;
using namespace cv;

int main(int argc, char** argv) {
        char* videoName = argv[1];
        const int64 inicio = getTickCount();
```

```
        Mat FrameAtual, suave_image;
        int numFrames = 0;
        vector<Vec3f> circles;
        ofstream arquivo;
        namedWindow("Video Processado",CV_WINDOW_AUTOSIZE);
        VideoCapture capture(videoName);
        if (!capture.isOpened())
                return 1;
        bool stop(false);
        double FrameRate = capture.get(CV_CAP_PROP_FPS);
        int Delay = 1000 / FrameRate;
        arquivo.open("raios.txt", ios::out);
        while (!stop) {
                if (!capture.read(FrameAtual))
                        break;
                ++numFrames;
                cvtColor(FrameAtual, FrameAtual, CV_BGR2GRAY);
                medianBlur(FrameAtual, suave_image, 11);
                HoughCircles(suave_image, circles, CV_HOUGH_GRADIENT,
                    3, FrameAtual.rows / 1.5, 30, 120, 50, 120);
                vector<Vec3f>::const_iterator itc = circles.begin();
                cvtColor(FrameAtual, FrameAtual, CV_GRAY2BGR);
                while (itc != circles.end()) {
                        circle(FrameAtual, Point((*itc)[0], (*itc)
                            [1]), (*itc)[2], Scalar(0,255,255), 1);
                        line(FrameAtual, Point((*itc)[0]-20, (*itc)
                            [1]), Point((*itc)[0]+20, (*itc)[1]),
                            Scalar(0, 0, 200), 1, 8);
```

```
                        line(FrameAtual, Point((*itc)[0], (*itc)
                            [1]-20), Point((*itc)[0], (*itc)[1]+20),
                            Scalar(0, 0, 200), 1, 8);
                        ++itc;
                }
                imshow("Video Processado", FrameAtual);
                if (!circles.empty()) {
                        cout << "\nRaio encontrado no frame " <<
                            numFrames << ": " << circles[0][2] << " na
                             posicao " << circles[0][0] << "," <<
                            circles[0][1];
                        arquivo << circles[0][2] << endl;
                }
                else {
                        cout << "\nRaio nao encontrado no frame " <<
                            numFrames << ".";
                        arquivo << endl;
                }
        }
        cout << "\n\n\nProcessamento finalizado. Tempo total de
            processamento: " << (getTickCount() - inicio) * 1000 /
            getTickFrequency() << "ms";
        waitKey(0);
        arquivo.close();
        return 0;
}
```

A.2 Rastreamento pupilar utilizando as otimizações propostas

Programa para otimizar a busca pupilar pelo método de Hough em vídeo derivado das 16.212 imagens do banco CASIA:

- Define Região de interesse (ROI) dentro da imagem inteira a ser processada (imagem cortada para dimensão 250x250 em torno do último centro pupilar encontrado);
- Define faixa de busca de circunferências com raio em tono do último encontrado (vizinhança de 20px).

Requer bibliotecas:

- OpenCV

```
#include <opencv2/opencv.hpp>
#include <iostream>
#include <fstream>

using namespace std;
using namespace cv;

int main(int argc, char** argv) {
        char* videoName = argv[1];
        const int64 inicio = getTickCount();
        Mat FrameAtual, suave_image, roi;
        int numFrames = 0;
        vector<Vec3f> circles;
        ofstream arquivo;
        int centerx, centery, raio_encontrado;
```

```
namedWindow("Video Processado", CV_WINDOW_AUTOSIZE);
namedWindow("ROI", CV_WINDOW_AUTOSIZE);

VideoCapture capture(videoName);
if (!capture.isOpened())
        return 1;
bool stop(false);
double FrameRate = capture.get(CV_CAP_PROP_FPS);
int Delay = 1000 / FrameRate;
arquivo.open("raios.txt", ios::out);
while (!stop) {
        if (!capture.read(FrameAtual))
                break;
        ++numFrames;
        cvtColor(FrameAtual, FrameAtual, CV_BGR2GRAY);
        if (!circles.empty()) {
                roi = FrameAtual(Range(int(centerx - 125),
                    int(centerx + 125)), Range(int(centery -
                    125), int(centery + 125)));
        }
        else {
                centery = FrameAtual.cols / 2;
                centerx = FrameAtual.rows / 2;
                roi = FrameAtual(Range(int(centerx - 125),
                    int(centerx + 125)), Range(int(centery -
                    125), int(centery + 125)));
        }
```

```
medianBlur(roi, suave_image, 11);
if (!circles.empty()) {
        HoughCircles(suave_image, circles,
            CV_HOUGH_GRADIENT, 3, FrameAtual.rows /
            1.5, 90, 120, raio_encontrado - 10,
            raio_encontrado + 10);
        raio_encontrado = circles[0][2];
}
else {
        HoughCircles(suave_image, circles,
            CV_HOUGH_GRADIENT, 3, FrameAtual.rows /
            1.5, 90, 120, 25, 80);
        raio_encontrado = circles[0][2];
}
vector<Vec3f>::const_iterator itc = circles.begin();
cvtColor(roi, roi, CV_GRAY2BGR);
cvtColor(FrameAtual, FrameAtual, CV_GRAY2BGR);

while (itc != circles.end()) {
        circle(roi, Point((*itc)[0], (*itc)[1]), (*
            itc)[2], Scalar(0, 255, 255), 1);
        line(roi, Point((*itc)[0] - 15, (*itc)[1]),
            Point((*itc)[0] + 15, (*itc)[1]), Scalar
            (0, 0, 200), 1, 8);
        line(roi, Point((*itc)[0], (*itc)[1] - 15),
            Point((*itc)[0], (*itc)[1] + 15), Scalar
            (0, 0, 200), 1, 8);
        rectangle(FrameAtual, Point(centery-125,
```

```
                        centerx-125), Point(centery + 125, centerx
                         + 125), Scalar(0, 55, 255), +2, 4);
                ++itc;
            }
            centerx = centerx - 125 + circles[0][1];
            centery = centery - 125 + circles[0][0];

            imshow("Video Processado", FrameAtual);
            imshow("ROI", roi);
            if (!circles.empty()) {
                cout << "\nRaio encontrado no frame " <<
                    numFrames << ": " << circles[0][2] << " na
                     posicao " << circles[0][0] + 150 << ","
                    << circles[0][1] + 150;
                arquivo << circles[0][2] << endl;
            }
            else {
                cout << "\nRaio nao encontrado no frame " <<
                    numFrames << ".";
                arquivo << endl;
            }
            waitKey(0);
        }
        cout << "\n\n\nProcessamento finalizado. Tempo total de
            processamento: " << (getTickCount() - inicio) * 1000 /
            getTickFrequency() << "ms";
        waitKey(0);
        arquivo.close();
```

```
        return 0;
}
```

A.3 Avaliação do efeito do redimensionamento das imagens

Programa para avaliar o efeito do redimensionamento na eficácia da detecção de bordas pupilares e consequente localização pupilar através da Transformada Circular de Hough:

- Tamanho da imagem reduzido em 2, 3 e 4 vezes. Imagens redimensionadas são salvas.
- Exibe-se a imagem reduzida em 4 vezes, sua versão suavizada e sua detecção de bordas

Requer bibliotecas:

- OpenCV

```
#include <opencv2/opencv.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <iostream>
#include <fstream>

using namespace cv;
using namespace std;

int main(int argc, char** argv)
{
        char* imageName = argv[1];
        double start = getTickCount();
        Mat image, metade, terço, quarto, suave, suave_metade,
            suave_terço, suave_quarto, bordas, bordas_metade,
            bordas_terço, bordas_quarto, hough, hough_metade;
```

```
vector<Vec3f> circles, circles_metade;
image = imread(imageName, 1);
if (argc != 2 || !image.data)
{
        printf(" No image data \n ");
        return -1;
}
cvtColor(image, image, CV_BGR2GRAY);

resize(image, metade, Size(),0.5,0.5);
resize(image, terço, Size(),0.3,0.3);
resize(image, quarto, Size(), 0.25, 0.25);

GaussianBlur(image, gaussian, Size(11, 11), 0, 0);
medianBlur(image, median, 11);
bilateralFilter(image, bilateral, 11, 11 * 2, 11 / 2);*/

medianBlur(image, suave, 11);
medianBlur(metade, suave_metade, 11);
medianBlur(terço, suave_terço, 11);
medianBlur(quarto, suave_quarto, 11);

Canny(suave, bordas, 125, 225);
Canny(suave_metade, bordas_metade, 125, 225);
Canny(suave_terço, bordas_terço, 125, 225);
Canny(suave_quarto, bordas_quarto, 125, 225);

HoughCircles(suave, circles, CV_HOUGH_GRADIENT, 3, suave.rows
```

```
        / 1.5, 90, 120, 25, 80);
    vector<Vec3f>::const_iterator itc = circles.begin();
    cvtColor(image, hough, CV_GRAY2BGR);
    while (itc != circles.end()) {
            circle(hough, Point((*itc)[0], (*itc)[1]), (*itc)[2],
                Scalar(0, 255, 255), 1);
            line(hough, Point((*itc)[0] - 15, (*itc)[1]), Point
               ((*itc)[0] + 15, (*itc)[1]), Scalar(0, 0, 200), 1,
                8);
            line(hough, Point((*itc)[0], (*itc)[1] - 15), Point
               ((*itc)[0], (*itc)[1] + 15), Scalar(0, 0, 200), 1,
                8);
            ++itc;
    }
    HoughCircles(suave_metade, circles_metade, CV_HOUGH_GRADIENT,
        3, suave_metade.rows / 1.5, 90, 60, 12, 40);
    itc = circles_metade.begin();
    cvtColor(metade, hough_metade, CV_GRAY2BGR);
    while (itc != circles_metade.end()) {
            circle(hough_metade, Point((*itc)[0], (*itc)[1]), (*
               itc)[2], Scalar(0, 255, 255), 1);
            line(hough_metade, Point((*itc)[0] - 15, (*itc)[1]),
               Point((*itc)[0] + 15, (*itc)[1]), Scalar(0, 0,
               200), 1, 8);
            line(hough_metade, Point((*itc)[0], (*itc)[1] - 15),
               Point((*itc)[0], (*itc)[1] + 15), Scalar(0, 0,
               200), 1, 8);
            ++itc;
```

```
        }
        imwrite("Metade.jpg", metade);
        imwrite("Terço.jpg", terço);
        imwrite("Quarto.jpg", quarto);
        imwrite("Imagem_Suavizada.jpg", suave);
        imwrite("Metade_Suavizada.jpg", suave_metade);
        imwrite("Terço_Suavizada.jpg", suave_terço);
        imwrite("Quarto_Suavizada.jpg", suave_quarto);
        imwrite("Imagem_Bordas.jpg", bordas);
        imwrite("Metade_Bordas.jpg", bordas_metade);
        imwrite("Terço_Bordas.jpg", bordas_terço);
        imwrite("Quarto_Bordas.jpg", bordas_quarto);*/

        resize(quarto, quarto, Size(), 4, 4);
        resize(suave_quarto, suave_quarto, Size(), 4, 4);
        resize(bordas_quarto, bordas_quarto, Size(), 4, 4);
        resize(hough_metade, hough_metade, Size(), 2, 2);

        namedWindow("Imagem", CV_WINDOW_AUTOSIZE);
        imshow("Imagem", image);
        namedWindow("Redimensionada", CV_WINDOW_AUTOSIZE);
        imshow("Redimensionada", quarto);
        namedWindow("Detecção de Bordas", CV_WINDOW_AUTOSIZE);
        imshow("Detecção de Bordas", bordas);
        namedWindow("Detecção de Bordas Redimensionada",
           CV_WINDOW_AUTOSIZE);
        imshow("Detecção de Bordas Redimensionada", bordas_quarto);
        namedWindow("Hough", CV_WINDOW_AUTOSIZE);
```

```
        imshow("Hough", hough);

        namedWindow("Hough Redimensionado", CV_WINDOW_AUTOSIZE);

        imshow("Hough Redimensionado", hough_metade);


        waitKey(0);

        return 0;
}
```

A.4 Rastreamento pupilar em tempo real no raspberry Pi 2

Programa para efetuar o rastreamento pupilar em tempo real a partir de fluxo de vídeo proveniente da câmera Pi NoIR capturado em infravermelho:

Requer bibliotecas:

- OpenCV
- Raspicam

```
#include <opencv2/opencv.hpp>
#include <raspicam/raspicam_cv.h>
#include <iostream>


using namespace std;
using namespace cv;


int main() {


        Mat FrameAtual, suave_image;
        long count = 0;
        vector<Vec3f> circles;


        namedWindow("Video Processado", CV_WINDOW_AUTOSIZE);


        raspicam::RaspiCam_Cv capture;
        capture.set(CV_CAP_PROP_FORMAT, CV_8UC1 );
        capture.set( CV_CAP_PROP_FPS, 60 );
        capture.set ( CV_CAP_PROP_FRAME_WIDTH, 320 );
```

```
capture.set ( CV_CAP_PROP_FRAME_HEIGHT, 240 );
//capture.setFormat(raspicam::RASPICAM_FORMAT_GRAY);
//capture.setCaptureSize(640,480);

if(!capture.open()) {cerr<<"Erro ao abrir camera\n";return
    -1;}
usleep(500000);

bool stop(false);

        double FrameRate = capture.get(CV_CAP_PROP_FPS);
        cout<<"\nFPS camera: "<<FrameRate;
        int Delay = 1000 / 60;
        double time=cv::getTickCount();

while (!stop) {
  capture.grab();
  capture.retrieve(FrameAtual);

        medianBlur(FrameAtual, suave_image, 11);
        HoughCircles(suave_image, circles, CV_HOUGH_GRADIENT,
            3, suave_image.rows/1.5, 90, 120, 25, 80);
        vector<Vec3f>::const_iterator itc = circles.begin();
        while (itc != circles.end()) {
                circle(FrameAtual, Point((*itc)[0], (*itc)
                    [1]), (*itc)[2], Scalar(255), 2);
                ++itc;
        }
```

```
                imshow("Video Processado", FrameAtual);
                count++;

                if (cv::waitKey(Delay) >= 0)
                        stop = true;
                cout<<endl<<count;
        }

        double secondsElapsed = double ( cv::getTickCount()-time ) /
            double ( cv::getTickFrequency() ); //tempo em segundos
        cout<<endl<< secondsElapsed<<" segundos para capturar "<<
            count<<"  frames : FPS = "<< ( float ) ( ( float ) ( count
             ) /secondsElapsed ) <<endl;
        capture.release();
        return 0;

}
```

A.5 Captura de ciclo pupilar com estímulo luminoso simultâneo - PLR

Programa para capturar vídeo de 10 seg e localizar circunferências correspondentes à borda da pupila através do método de Hough.

Método aplicado a feed da câmera Raspi Cam;

LEDs controlados através da biblioteca WiringPi;

- 89 frames -> LED apagado;
- 1 frame -> LED aceso;
- até o frame 200 -> LED apagado.

Requer bibliotecas:

- OpenCV
- WiringPi
- Raspicam

```
#include <opencv2/opencv.hpp>
#include <raspicam/raspicam_cv.h>
#include <wiringPi.h>
#include <iostream>


using namespace std;
using namespace cv;


int main() {


        Mat FrameAtual, suave_image;
        long count = 0;
```

```
vector<Vec3f> circles;
namedWindow("Video Processado", CV_WINDOW_AUTOSIZE);


raspicam::RaspiCam_Cv capture;
capture.set(CV_CAP_PROP_FORMAT, CV_8UC1 );
capture.set( CV_CAP_PROP_FPS, 60 );
capture.set ( CV_CAP_PROP_FRAME_WIDTH, 320 );
capture.set ( CV_CAP_PROP_FRAME_HEIGHT, 240 );
capture.set( CV_CAP_PROP_FPS, 60 );
if(!capture.open()) {cerr<<"Erro ao abrir camera\n";return
    -1;}
usleep(500000);
bool stop(false);
wiringPiSetup();
pinMode(0, OUTPUT);
digitalWrite(0,LOW);
double FrameRate = capture.get(CV_CAP_PROP_FPS);
cout<<"\nFPS camera: "<<FrameRate;
int Delay = 1000 / 60;
double time=cv::getTickCount();
Size S = Size(320,240);
int fourcc = CV_FOURCC('H','2','6','4');
VideoWriter saida_Video;
saida_Video.open("Video.avi", fourcc, 8, S, false);


while (count<=200) {
  capture.grab();
  capture.retrieve(FrameAtual);
```

```
                medianBlur(FrameAtual, suave_image, 11);
                HoughCircles(suave_image, circles, CV_HOUGH_GRADIENT,
                    3, suave_image.rows/1.5, 90, 120, 25, 80);
                vector<Vec3f>::const_iterator itc = circles.begin();
                while (itc != circles.end()) {
                        circle(FrameAtual, Point((*itc)[0], (*itc)
                            [1]), (*itc)[2], Scalar(255), 2);
                        ++itc;
                }
                saida_Video.write(FrameAtual);
                count++;
                if(count==90)
                        digitalWrite(0,HIGH);
                if(count==91)
                        digitalWrite(0,LOW);
                if (cv::waitKey(Delay) >= 0)
                        stop = true;
                cout<<"\n"<<count;
        }
        double secondsElapsed= double ( cv::getTickCount()-time ) /
            double ( cv::getTickFrequency() ); //tempo em segundos
        cout<<endl<< secondsElapsed<<" segundos para capturar "<<
            count<<"  frames : FPS = "<< ( float ) ( ( float ) ( count
             ) /secondsElapsed ) <<endl;
        capture.release();
        return 0;
}
```

A.6 Rastreamento em vídeo de ciclo pupilar obtido para análise de PLR

Programa para efetuar o rastreamento pupilar em sequências de vídeos capturadas pelo protótipo Raspberry Pi 2 utilizando a câmera Pi NoIR. Vídeo contendo 200 frames representa a resposta pupilar a um estímulo luminoso de forma degrau aplicado.

Requer bibliotecas:

- OpenCV

```
#include <opencv2/opencv.hpp>
#include <iostream>
#include <fstream>


using namespace std;
using namespace cv;


int main(int argc, char** argv) {
        char* videoName = argv[1];
        const int64 inicio = getTickCount();
        Mat FrameAtual, suave_image, bordas_image;
        int numFrames = 0;
        vector<Vec3f> circles;
        ofstream arquivo;
        int canny_inf, canny_sup;
        canny_sup = 92;
        canny_inf = canny_sup / 2;


        namedWindow("Video Processado", CV_WINDOW_AUTOSIZE);
```

```
VideoCapture capture(videoName);
if (!capture.isOpened())
        return 1;
bool stop(false);

                                        //Condição de
   parada se usuário pressionar alguma tecla durante execução

double FrameRate = capture.get(CV_CAP_PROP_FPS);

           //Obter taxa de quadros do vídeo a ser processado
int Delay = 1000 / FrameRate;

arquivo.open("raios.txt", ios::out);

while (!stop) {
        if (!capture.read(FrameAtual))

                                        //Lê próximo frame
           , se existir; se não existe sai da execução
                break;
        ++numFrames;

        cvtColor(FrameAtual, FrameAtual, CV_BGR2GRAY);

        medianBlur(FrameAtual, suave_image, 11);

        Canny(suave_image, bordas_image, canny_inf, canny_sup
```

```
                    );

                HoughCircles(suave_image, circles, CV_HOUGH_GRADIENT,
                    3, FrameAtual.rows / 3, canny_sup, 45, 8, 37);

                vector<Vec3f>::const_iterator itc = circles.begin();

                cvtColor(FrameAtual, FrameAtual, CV_GRAY2BGR);

                while (itc != circles.end()) {
                        circle(FrameAtual, Point((*itc)[0], (*itc)
                            [1]), (*itc)[2], Scalar(0, 255, 255), 1);
                        line(FrameAtual, Point((*itc)[0] - 20, (*itc)
                            [1]), Point((*itc)[0] + 20, (*itc)[1]),
                            Scalar(0, 0, 200), 1, 8);
                        line(FrameAtual, Point((*itc)[0], (*itc)[1] -
                             20), Point((*itc)[0], (*itc)[1] + 20),
                            Scalar(0, 0, 200), 1, 8);
                        ++itc;
                }

                imshow("Video Processado", FrameAtual);
                imshow("Bordas", bordas_image);

                if (!circles.empty()) {
                        cout << "\nRaio encontrado no frame " <<
                            numFrames << ": " << circles[0][2] << " na
                             posicao " << circles[0][0] << "," <<
```

```
                        circles[0][1];
                    arquivo << circles[0][2] << endl;
                }
                else {
                    cout << "\nRaio nao encontrado no frame " <<
                        numFrames << ".";
                    arquivo << endl;
                }
                waitKey(10);
            }
            cout << "\n\n\nProcessamento finalizado. Tempo total de
                processamento: " << (getTickCount() - inicio) * 1000 /
                getTickFrequency() << "ms";
            waitKey(0);
            arquivo.close();
            return 0;
}
```