UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

FERRAMENTA VOLTADA À MEDICINA PREVENTIVA PARA DIAGNOSTICAR CASOS DE ESTRABISMO

ISRAEL DAMÁSIO MEDEIROS

BLUMENAU
2008

2008/1-17

ISRAEL DAMÁSIO MEDEIROS

FERRAMENTA VOLTADA À MEDICINA PREVENTIVA PARA DIAGNOSTICAR CASOS DE ESTRABISMO

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciências da Computação — Bacharelado.

Prof. Paulo César Rodacki Gomes, Dr. - Orientador

BLUMENAU
2008

2008/1-17

FERRAMENTA VOLTADA À MEDICINA PREVENTIVA

PARA DIAGNOSTICAR CASOS DE ESTRABISMO

Por

ISRAEL DAMÁSIO MEDEIROS

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Paulo César Rodacki Gomes, Dr. – Orientador, FURB
Membro:	Prof. Dalton Solano dos Reis, Ms. – FURB
Membro:	Prof. Francisco Adell Péricas, Ms. – FURB

Blumenau, 08 de julho de 2008.

Dedico este trabalho aos meus pais, Antônio Carlos de Medeiros e Rosane Medeiros, e a todos os amigos que, de alguma maneira, me apoiaram na realização deste.

AGRADECIMENTOS

Ao médico, Alessandro Dantas Pennela, que além de um enorme colaborador técnico, está sempre disposto a novos desafios.

Ao meu orientador, Paulo César Rodacki Gomes, que acreditou na minha capacidade e conseqüentemente na conclusão do trabalho.

Ao George Ruberti Piva que foi voluntário em disponibilizar sua imagem para ser adicionada no volume final.

A todos os pesquisadores de diferentes lugares do mundo que me ajudaram através de *e-mails*.

Sábio é aquele que conhece os limites da própria ignorância.

Sócrates

RESUMO

No presente trabalho são descritas técnicas, bem como a especificação e implementação de um software para extrair medidas, através de uma imagem digital, para prevenir o estrabismo. São utilizadas técnicas de processamento de imagens para segmentar e preparar a imagem para ser analisada, enfatizando os algoritmos de Canny para detecção de bordas e a transformada de Hough para detecção de objetos de interesse. O software é capaz de extrair com precisão as medidas propostas.

Palavras-chave: Estrabismo. Imagem digital. Processamento de imagens. Algoritmo de Canny. Transformada de Hough.

ABSTRACT

This work describes techniques, as well as the specification and implementation of a software to extract metrics, by a digital image, to prevent the strabismus. It uses techniques of image processing to segment and prepare the image to be analyzed, emphasizing the Canny algorithm to the border detection and the Hough transform to detect the objects of interest. The software is capable to extract with precision the proposal metrics.

Key-words: Strabismus. Digital image. Images processing. Canny Algorithm. Hough transform.

LISTA DE ILUSTRAÇÕES

Figura 1 – Criança com exotropia. 18
Figura 2 - (a) imagem de um quadrado degrade; (b) imagem original (a) projetada em uma matriz discreta; (c) a média de luminosidade produzida pelo foto-receptor e (d) imagem digital representada em uma matriz de inteiros. 20
Figura 3 – Fluxo para processamento de imagens. 21
Figura 4 – (a) imagem original com ruído; (b) redução de ruído aleatório com Filtro Mediana. 21
Figura 5 – (a) imagem original em escala de cinza; (b) aplicação do filtro de Sobel. 22
Figura 6 – Exemplo de desenhos básicos em Java2D. 23
Quadro 1 – Código de exemplo do Java 2D. 24
Figura 7 – Estrutura da classe `PlanarImage` 25
Quadro 2 – Código para inversão de imagem com o JAI. 25
Figura 8 – Processo de detecção de bordas por Canny. 26
Quadro 3 – Função gaussiana em uma dimensão. 27
Quadro 4 – Primeira derivada da função gaussiana em 1D. 27
Figura 9 – (a) imagem original; (b) aplicação do filtro de Canny. 27
Figura 10 – Etapas da transformada de Hough para qualquer forma geométrica. 28
Figura 11 – Circunferência de raio r e centro x_c, y_c. 29
Quadro 5 – Fórmula implícita da circunferência. 29
Quadro 6 – Fórmula da circunferência baseada em coordenadas polares. 29
Quadro 7 – Fórmula da circunferência baseada em coordenadas polares já conhecendo as coordenadas do centro de círculo. 30
Quadro 8 – Pseudocódigo para criação do espaço de Hough. 30
Figura 12 – (a) Exemplo de uma imagem; (b) seu respectivo espaço de Hough; (c) o mesmo espaço de hough com ajuste de contraste para facilitar a visualização e (d) uma imagem composta pela adição da imagem original com o espaço de Hough. 31
Figura 13 – Foto de uma retina normal. 33
Figura 14 – (a) medida entre o reflexo da córnea e o limbo no sentido nasal “x1” e (b) medida entre o reflexo da córnea e o limbo no sentido temporal “x2”. A figura é baseada no olho esquerdo do paciente. 35
Figura 15 – Diagrama de casos de uso. 36

Quadro 9 – Caso de uso definir ROI na imagem. 36
Quadro 10 – Caso de uso executar operador Canny. 37
Quadro 11 – Caso de uso executar transformada circular Hough. 37
Quadro 12 – Caso de uso requisitar relatório exame. 37
Figura 16 – Diagrama de seqüência para o caso de uso requisitar relatório exame. 38
Figura 17 – Diagrama de pacotes da ferramenta. 38
Figura 18 – Diagrama de classes do módulo de desenhos geométricos. 39
Figura 19 – Diagrama de classes para o operador de Canny. 40
Figura 20 – Diagrama de classes para a transformada circular de Hough. 41
Figura 21 – Diagrama de classes para operações auxiliares de processamento de imagens. 42
Figura 22 – Diagrama de classes do módulo de interface gráfica. 43
Figura 23 – Diagrama de classes do módulo de resultados. 44
Quadro 13 – Código fonte do método `getCircle()` que identifica uma forma circular na imagem digital através de um intervalo de raio. 45
Quadro 14 – Código fonte do método `circleTransform()`, responsável por identificar possíveis coordenadas de centro da circunferência. 46
Quadro 15 – Código fonte do método responsável por construir o espaço de Hough. 46
Quadro 16 – Código fonte do método `process()` do operador de Canny. 47
Quadro 17 – Código fonte da classe `DrawableAnnotation`. 48
Quadro 18 – Código fonte do método `paint()` da classe `LineAnnotation`. 48
Quadro 19 – Código fonte do método `paint()` da classe `DisplayJAIAnnotations`. 48
Quadro 20 – Código fonte do método `translateRectROI()` da classe `DisplayJAIEvents`. 49
Quadro 21 – Parte do código fonte do método para extrair as medidas do estrabismo. 50
Figura 24 – Tela menu *File*. 51
Figura 25 – Tela de informação para resolução abaixo do mínimo exigido. 51
Figura 26 – Tela de informação para arquivos que não sejam imagens. 51
Figura 27 – Tela principal da ferramenta com a foto do usuário carregada. 52
Figura 28 – *Line tool*. 53
Figura 29 – *Rectangle tool*. 53
Figura 30 – *Canny properties*. 53
Figura 31 – *Hough properties*. 53
Figura 32 – *View Canny result*. 53

Figura 33 – *Run Hough algorithm.* 53
Figura 34 – *Erase Hough result*. 54
Figura 35 – Exemplo de uso de ROIs da ferramenta. 54
Figura 36 – Execução do operador de Canny na ferramenta. 55
Figura 37 – Execução do algoritmo de Hough na ferramenta para detectar as íris. 56
Figura 38 – Visualização das íris detectadas. 56
Figura 39 – Tela de resultados do exame. 57
Figura 40 – Simulação de exotropia. 59
Figura 41 – Resultado da simulação de exotropia. 60
Figura 42 – Foto da face de uma pessoa com régua para calibração de medida. 61

LISTA DE SIGLAS

1D – Uma dimensão

2D – Duas dimensões

API – *Application Programming Interface*

GUI – *Graphical User Interface*

JAI – *Java Advanced Imaging* API

JUDE – *Java UML Modeling Tool*

PDF – *Portable Document Format*

RGB – *Red, Green and Blue*

ROI – *Region Of Interest*

UML – *Unified Modeling Language*

SUMÁRIO

1 INTRODUÇÃO 14
1.1 OBJETIVOS DO TRABALHO 15
1.2 ESTRUTURA DO TRABALHO 15
2 FUNDAMENTAÇÃO TEÓRICA 17
2.1 ESTRABISMO 17
2.2 TELEMEDICINA 18
2.3 IMAGEM DIGITAL 19
2.4 PROCESSAMENTO DE IMAGENS 21
2.5 JAVA 2D 23
2.5.1 JAI – Java *advanced imaging* API 24
2.6 OPERADOR DE CANNY 26
2.7 TRANSFORMADA DE HOUGH 28
2.7.1 Transformada de Hough para formas circulares 28
2.8 TRABALHOS CORRELATOS 31
2.8.1 Biometria com enfoque em reconhecimento de íris 32
2.8.2 Processamento e reconhecimento de imagens digitais da retina humana 32
3 DESENVOLVIMENTO DA FERRAMENTA 34
3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO 34
3.2 ESPECIFICAÇÃO 35
3.2.1 Módulo de desenhos geométricos 39
3.2.2 Módulo de processamento de imagem 40
3.2.3 Módulo de interface gráfica 42
3.2.4 Módulo de resultados 43
3.3 IMPLEMENTAÇÃO 44
3.3.1 Técnicas e ferramentas utilizadas 44
3.3.1.1 Detecção de íris e reflexo na córnea ocular 44
3.3.1.2 Anotações na imagem através de desenhos geométricos 47
3.3.1.3 Desenhos interativos de ROIs e extração de medidas do estrabismo 48
3.3.2 Operacionalidade da implementação 51
3.3.2.1 Abrindo uma foto para exame 51
3.3.2.2 Recursos da ferramenta 52

3.3.2.3 Utilizando os recursos da ferramenta 54
3.4 RESULTADOS E DISCUSSÃO 58
4 CONCLUSÕES 62
4.1 EXTENSÕES 63
REFERÊNCIAS BIBLIOGRÁFICAS 64

1 INTRODUÇÃO

O estrabismo é uma entidade caracterizada pela perda do paralelismo entre os olhos e pode ocasionar alterações na função visual como a perda da estereopsia. Em condições normais, os músculos que fazem mover os olhos trabalham de forma coordenada, permitindo a visão binocular, ou estereopsia, fenômeno no qual o cérebro funde as imagens dos olhos e as interpreta como uma só. Corrêa (2006) afirma que a estereopsia é sinônimo de sensação espacial e isto garante que as pessoas percebam, de maneira correta, a distância entre objetos. Se os olhos não se dirigem exatamente para o mesmo ponto de fixação, o cérebro percebe duas imagens do mesmo quadro. As imagens tornam-se sobrepostas, ocasionando a confusão visual e a diplopia[1].

Embora freqüentemente não se possa determinar a causa exata, muitos casos de estrabismo estão ligados a uma herança genética. O desvio manifesto[2] pode surgir pela falta da correção óptica no momento adequado, problema exacerbado quando as crianças entram na escola e passam a usar a visão de forma mais acentuada. As doenças do sistema nervoso central tais como meningite, paralisia cerebral e a síndrome de Down, podem vir acompanhadas de estrabismo.

Muitas vezes, o estrabismo não é percebido de imediato pelas pessoas. Geralmente quando percebido, o grau da doença já é avançado e exige tratamentos mais onerosos, como uma cirurgia, por exemplo. A possibilidade de oferecer para a população uma ferramenta capaz de avaliar automaticamente a existência de desvio manifesto de um dos olhos é capaz de antecipar a ida ao médico, gerar diagnósticos mais precoces e evitar ou diminuir a incidência de complicações.

Para colaborar com a prevenção desta doença, o presente trabalho relata o desenvolvimento de uma ferramenta em forma de software que realiza a análise a partir de uma imagem da face humana, capturada por uma câmera digital com *flash*, tendo como base o teste de Hirschberg. Esta análise consiste em identificar o limbo[3], a íris e o reflexo ocasionado pelo *flash* da câmera na córnea ocular. A partir disso, são extraídas e comparadas medidas pré-estabelecidas dos olhos. Assim, o usuário pode saber se ele possui algum tipo de desvio

---

[1] Visão duplicada de um mesmo objeto.

[2] Sinônimo de estrabismo.

[3] Situado aonde termina a parte branca do olho (conjuntiva e esclera) e começa a parte colorida (íris).

manifesto horizontal, onde deverão ser verificados possíveis casos de esotropia e exotropia. A esotropia é a anomalia oculomotora mais comum nas pessoas, onde o olho é desviado em sentido nasal (SOUZA-DIAS; ALMEIDA, 1993, p. 15). Por outro lado, Souza-Dias e Almeida (1993, p. 15) afirmam que na exotropia o olho desviado encontra-se em sentido temporal.

O diagnóstico precoce do estrabismo é muito importante para permitir a sua correção e evitar complicações. Quanto mais tarde for diagnosticado e tratado, piores serão os resultados. Atualmente existe um consenso de que até o final do segundo ano de vida de uma pessoa, o estrabismo poderá ser corrigido sem seqüelas.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é explanar os conceitos e o desenvolvimento de uma ferramenta voltada à medicina preventiva, que utiliza técnicas de processamento de imagens, para diagnosticar possíveis casos de estrabismo.

Os objetivos específicos são:

a) analisar uma imagem da face humana e para cada olho identificar o limbo e o reflexo de luz na córnea ocular;
b) obter os resultados das seguintes medidas em cada olho:
   - distância do reflexo de luz da córnea ao limbo medial,
   - distância do reflexo de luz da córnea ao limbo temporal,
   - distância entre o limbo nasal e o temporal (diâmetro da íris);
c) padronizar a aquisição e análise das imagens do paciente tomando como base o teste de Hirschberg;
d) diagnosticar possíveis casos de desvios manifestos horizontais (esotropia e exotropia).

1.2 ESTRUTURA DO TRABALHO

O presente trabalho está estruturado em quatro capítulos. O segundo capítulo contém a

fundamentação teórica necessária para o entendimento do trabalho. Nele são discutidos tópicos relacionados sobre o estrabismo, telemedicina, imagens digitais, processamento de imagens, Java 2D e JAI, operador de Canny e transformada de Hough. Também são comentados alguns trabalhos correlatos à ferramenta.

O terceiro capítulo comenta sobre o desenvolvimento da ferramenta, onde são explanados os requisitos principais do problema trabalhado, a especificação contendo diagramas de casos de uso, seqüência e classes. Também são feitos comentários sobre a implementação abrangendo as técnicas e ferramentas utilizadas, operacionalidade e por fim são comentados os resultados e discussão.

O quarto capítulo refere-se às conclusões e extensões do trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados alguns conceitos relevantes ao desenvolvimento e entendimento deste trabalho. Também são realizados comentários sobre o estrabismo e os tipos de heterotropia horizontais que podem ocorrer, bem como sobre telemedicina. Em seguida, são apresentadas informações sobre imagem digital, processamento de imagens, Java 2D e JAI, operador de Canny e transformada de Hough. Por fim, são listados alguns trabalhos correlatos.

2.1 ESTRABISMO

"O estrabismo é um termo geral ao usado em casos de desalinhamento dos eixos visuais (desvio dos olhos) associado a um desequilíbrio dos músculos de extra-oculares" (CENTRO BRASILEIRO DE ESTRABISMO, 2008). Conforme Souza-Dias e Almeida (1993, p.72), o estrabismo ocorre entre 02 e 04 por cento da população. Tanto homens quanto mulheres podem vir a ter a doença igualmente. Segundo Centro Brasileiro de Estrabismo (2008), o estrabismo tem um padrão hereditário, onde é muito comum encontrar casos em famílias que possuem um histórico positivo para o problema. Por outro lado, muitos casos podem vir a ocorrer sem que o sujeito possua um histórico familiar positivo.

O estrabismo infantil pode causar uma grande perda no desenvolvimento visual e com isto produzir a chamada ambliopia. Este é um termo usado em casos de visão reduzida, que não melhora com correção visual, devido ao não uso de um dos olhos. O tratamento precoce permite que o problema seja solucionado, na maioria dos casos, com tratamentos simples.

A percepção de profundidade, visão tridimensional, também pode ser afetada devido ao estrabismo. Na idade adulta ele causa visão dupla que também é chamada de diplopia. Souza-Dias e Almeida (1993, p. 15) afirmam que quando ambos os olhos fixam um único objeto, estando alinhados e recebendo a mesma imagem, tem-se a ortotropia que é a forma normal do olhar**.** Ao contrário, quando os olhos não estão alinhados em relação a um ponto fixo, tem-se a heterotropia que é sinônimo de estrabismo.

Um dos tipos de heterotropia é a comitante. Segundo Souza-Dias e Almeida (1993, p. 15), "heterotropia comitante é aquela em que o ângulo de desvio é o mesmo em todas as

posições do olhar, independente do olho que fixar". Quando o olho desviar horizontalmente, podem-se abordar dois casos de heterotropia: exotropia e esotropia. A exotropia apresenta um desvio em sentido temporal, ou seja, o olho desviado encontra-se para fora. A Figura 1 apresenta a imagem de uma criança que possui exotropia. O olho esquerdo[4] é o que está fixando o objeto, enquanto o direito está desviado temporalmente. A esotropia também é um desvio horizontal, porém o olho desviado encontra-se em sentido nasal.

O teste de Hirschberg visa diagnosticar casos de estrabismo. Hirschberg propôs propagar um foco de luz nos olhos do paciente, tendo como objetivo comparar os reflexos propagados pela córnea. Assim, pode-se obter a angulação ou medida do desvio ocular.

Fonte: PedsEye (1999).

Figura 1 – Criança com exotropia

Há várias maneiras de se tratar o estrabismo, desde a simples oclusão de um olho, até meios cirúrgicos. A oclusão de um dos olhos é uma maneira simples de se tratar a ambliopia e que muitas vezes traz bons resultados para o paciente. Segundo Souza-Dias e Almeida (1993, p. 99), o tratamento cirúrgico visa modificar os músculos oculares, alterando o seu plano de ação.

2.2 TELEMEDICINA

Telemedicina é uma técnica que visa conciliar o uso dos computadores associado à área da saúde. O objetivo principal da telemedicina é aplicar a medicina à distância, ou seja, a aplicação médica sem requerer contato ou proximidade com o paciente. Medicina Geriátrica

[4] Em relação ao paciente.

(2008) afirma que telemedicina são serviços voltados à saúde para casos em que a distância é um grande fator crítico. Estes serviços são providos usando tecnologias de informação e de comunicação visando uma troca de informações válidas para diagnósticos, prevenção e tratamento de doenças.

Conforme Salido (2008), com a diversificação da utilização da telemedicina, o acesso à saúde fica mais fácil para a população e permite uma significativa descentralização, onde trabalhos que previamente têm sido realizados por centros de saúde primários, podem ser realizados na comunidade.

A história da telemedicina compõe de muitos sistemas que exigiram um grande investimento, mas que não sobreviveram ao estágio de entrar e modificar a rotina de um serviço clínico. Há um grande número de atividades em telemedicina no mundo, porém a demanda por novos projetos continua extremamente grande. A telemedicina ainda que aplicada como uma nova técnica continua extremamente experimental. Atualmente é suportada por programas de incentivo científico ou outros fundos e poucos programas são auto-suficientes economicamente (SALIDO, 2008).

Medicina Geriátrica (2008) ainda afirma que não há dúvida que a telemedicina fará diferença no tratamento de saúde global. Portanto, faz-se necessário aumentar os investimentos e o interesse nessa tecnologia, principalmente nos países em desenvolvimento.

2.3 IMAGEM DIGITAL

> A imagem digital é a materialização de grande parte dos processos da computação gráfica. Neste sentido, ela serve como elo de ligação entre o usuário e esses procedimentos, revelando os seus resultados. A imagem está presente em todas as áreas da computação gráfica, seja como produto final, no caso da visualização, ou como parte essencial do processo de interação, no caso da modelagem. (GOMES; VELHO, 2003, p. 147).

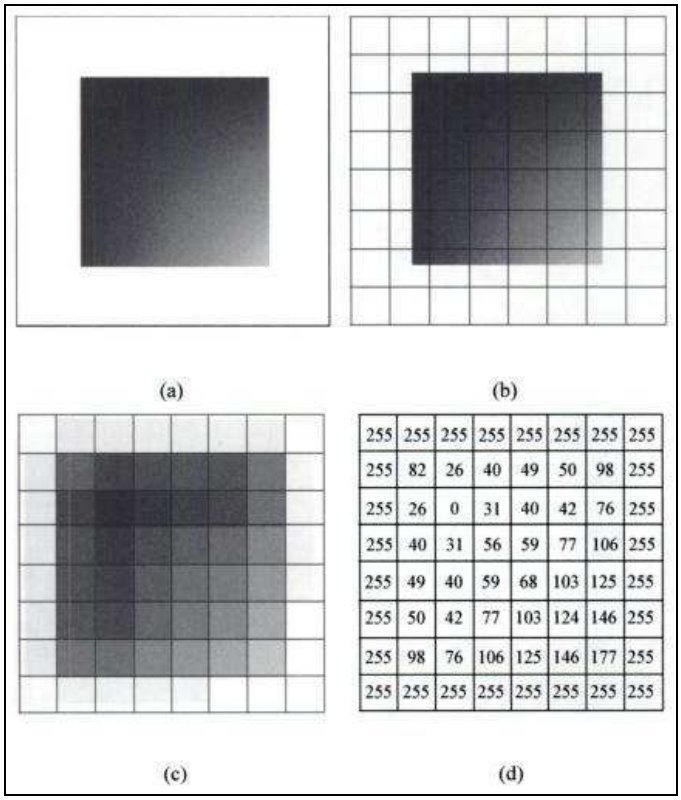
O fato de os computadores representarem a informação de uma maneira discreta faz com que as imagens também precisem ser representadas desta forma, assim podendo ser produzido algum processamento computacional com a imagem. Geralmente as imagens são transformadas em escala de cinza antes de serem processadas.

As imagens digitais podem ser armazenas em matrizes bidimensionais. Segundo McReynolds e Blythe (2005) a saída de um processo de renderização é uma imagem digital, cujo armazenamento é realizado através de uma matriz de *pixels*. Além disso, cada *pixel* pode

ser um simples componente escalar ou um vetor de valores escalares separados para cada componente de cor.

A maioria dos algoritmos de processamento de imagens utiliza como entrada imagens em escala de cinza. Isto ocorre devido a sua discretização, onde a escala de cinza pode assumir entre os valores 0 e 255, tornando o *pixel* um componente escalar simples.

A Figura 2 (a) exibe a imagem de um quadrado degradê. Para ser representada como uma imagem em que o computador possa processar é simulada a transformada discreta da Figura 2 (a). A respectiva discretização espacial da imagem na Figura 2 (b) é mostrada em (c). Agora para prover uma representação discreta para que o computador possa processar a imagem, é necessário que se assumam valores para cada tonalidade de cor cinza da Figura 2 (c). Isto pode ser feito assumindo valores conforme a intensidade de luz de cada ponto, tendo em vista que o valor 255 representa a cor branca e 0 representa a cor preta e os valores intermediários representam uma seqüência de valores discretos em escala de cinza. Depois de todo o processo realizado, o resultado se encontra na Figura 2 (d), onde uma matriz com valores discretos da imagem original (Figura 2 – a) é obtida.



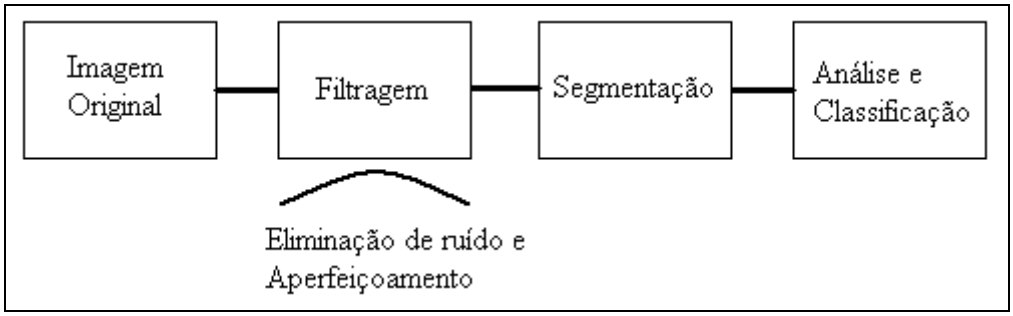
Fonte: Costa e César Jr. (1999, pg. 199).

Figura 2 - (a) imagem de um quadrado degrade; (b) imagem original (a) projetada em uma matriz discreta; (c) a média de luminosidade produzida pelo foto-receptor e (d) imagem digital representada em uma matriz de inteiros

2.4 PROCESSAMENTO DE IMAGENS

Costa e César Jr. (2001, p. 216) afirmam que o termo processamento de imagens é geralmente aplicado a métodos que recebem como entrada uma imagem e produz uma imagem modificada como saída. Por exemplo, uma imagem com um baixo contraste pode ser melhorada através de um método que realiza operações de realce e gera como saída uma imagem otimizada.

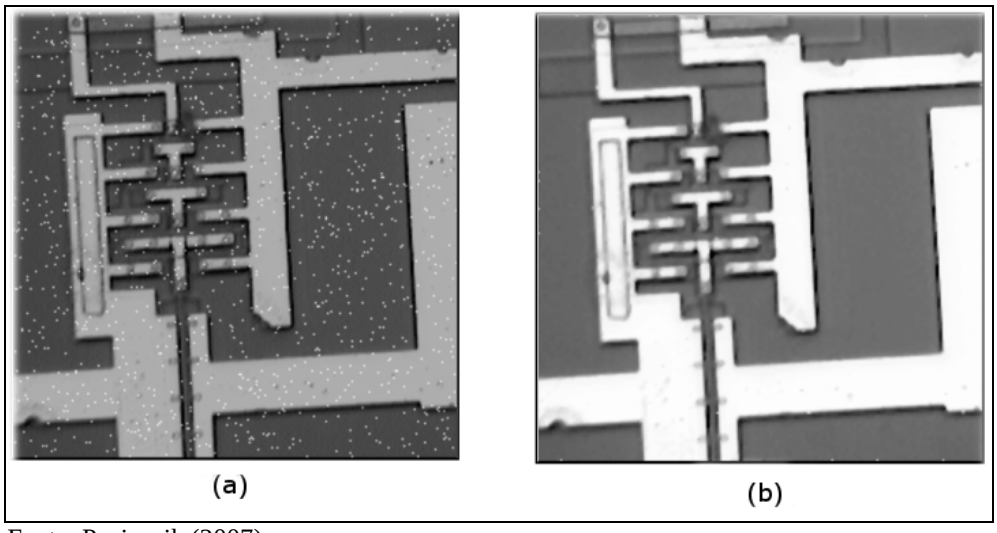
Segundo Paciornik (2007), processar uma imagem significa utilizar operações matemáticas a fim de modificar os valores dos *pixels* da imagem. Além de melhorar a qualidade, serve também para preparar a imagem para ser analisada. O fluxo genérico para se processar uma imagem encontra-se na Figura 3.



Fonte: Adaptado de Paciornik (2007).

Figura 3 – Fluxo para processamento de imagens

Na etapa de filtragem são eliminados os ruídos e é realizado um aperfeiçoamento da imagem. A Figura 4 (a) mostra uma imagem com ruídos e outra aperfeiçoada Figura 4 (b) após o processamento do Filtro Mediana.



Fonte: Paciornik (2007).

Figura 4 – (a) Imagem original com ruído; (b) **Redução** de ruído aleatório com Filtro Mediana

A etapa de segmentação é uma das mais importantes e mais críticas do processamento

de imagens. Procura-se distinguir as partículas umas das outras, interpretar *pixels* contíguos e agrupá-los por regiões ou demarcar coordenadas. Não existe nenhum modelo formalizado para a segmentação que na maioria das vezes torna-se um processo empírico.

Segundo Paciornik (2007), a segmentação da imagem é muito difícil e delicada, onde medidas são realizadas através das informações extraídas nesta etapa. Também é muito complexa, pois tenta representar para o computador um processo cognitivo extremamente sofisticado realizado através da visão humana.

Um exemplo de segmentação é a segmentação por contornos. As bordas da imagem são identificadas a partir de algum operador de derivada, por exemplo, Sobel[5]. A partir dos contornos são identificados os objetos através do processamento da imagem. Ao usar esta técnica, deve-se ficar atento em relação aos ruídos, podendo este ser um grande empecilho ao processar a imagem. Para exemplificar a detecção de bordas, a

Figura 5 (a) apresenta uma imagem em escala de cinza e a Figura 5 (b) mostra os contornos da primeira utilizando o filtro de Sobel.

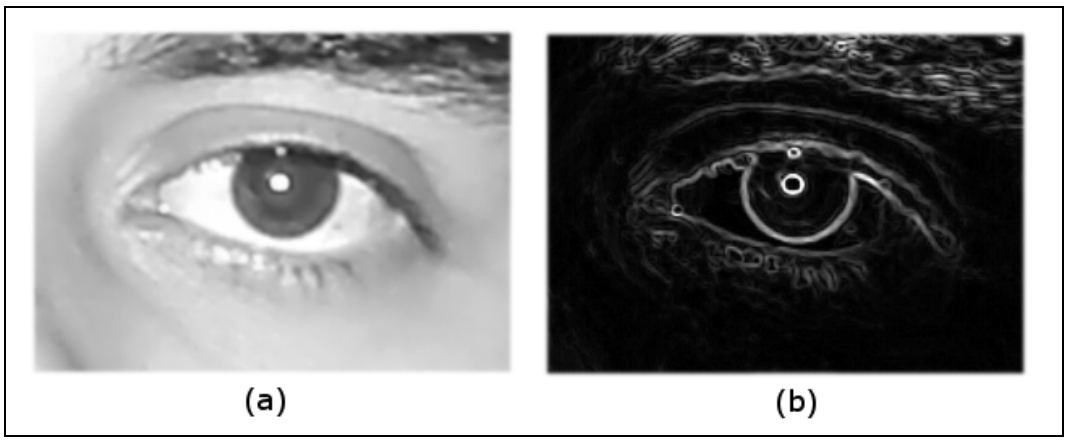


Figura 5 – (a) imagem original em escala de cinza; (b) aplicação do filtro de Sobel

Tendo ultrapassada a etapa de segmentação, pode-se então realizar as medidas sobre a imagem. Segundo Paciornik (2007) existem basicamente duas classes de medidas: medidas de campo e medidas de região. Na medida de campo, deve-se considerar as medidas que se referem ao campo como um todo, por exemplo, a área total de objetos. Na medida de região, são levadas em conta as medidas que se referem aos objetos independentemente conforme sua área, perímetro, forma, etc.

Há inúmeras vantagens em se analisar uma imagem através de um computador. Dentre várias delas, Parciornik (2007) destaca que, através da análise de uma imagem digital podem-

[5] Conforme Conci, Azevedo e Leta (2008, p. 179), este filtro consiste em detectar as alterações em uma imagem através de seus *kernels*. Estes servem para calcular gradientes horizontais e verticais da imagem, assim obtendo a magnitude e direção das bordas.

se alcançar medidas impossíveis de se obter manualmente, além de obter um resultado rápido e acurado.

2.5 JAVA 2D

Segundo Java 2D API (2008), Java 2D possui uma API com um conjunto de classes para gráficos 2D e processamento de imagens. Estas classes englobam desenhos de linha, texto e imagens em um único modelo abrangente. A API provê suporte extensivo para composição de imagem, cores acuradas e um conjunto extensivo de operadores para processamento e criação de imagens.

Uma instância de `java.awt.Graphics` é identificada como contexto gráfico. Ela é a base para desenhar em componentes ou em algum *buffer* de imagem. Conforme Niemeyer e Peck (1996, p. 331), um contexto gráfico provê métodos para realizar todas as operações básicas de desenho em sua região pré-determinada, incluindo a visualização da imagem.

Métodos da classe `Graphics` operam em um sistema de coordenadas padrão. A origem se situa no topo e na extremidade esquerda do componente (NIEMEYER; PECK, 1996, pg. 333). Um exemplo de desenho feito a partir do Java 2D pode ser encontrado na Figura 6 e o seu respectivo código fonte no Quadro 1.

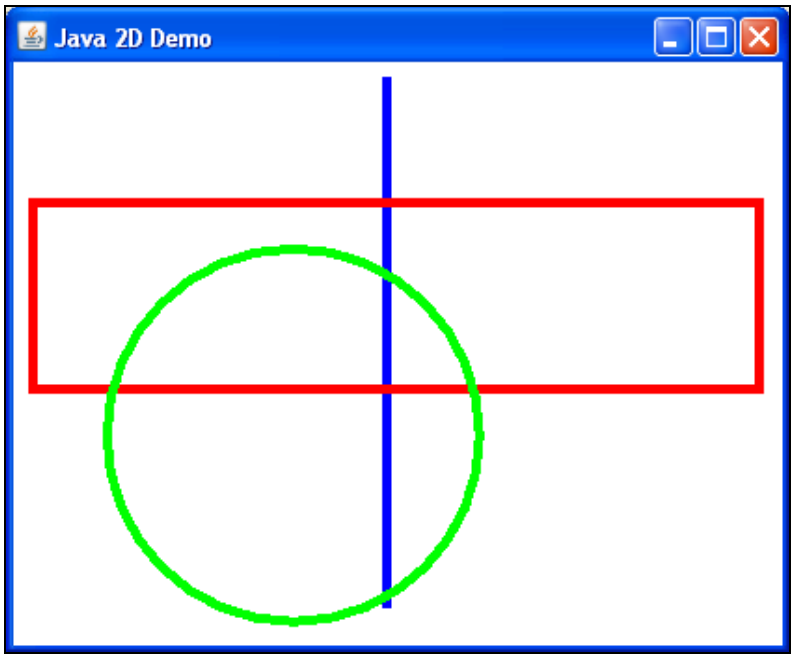


Figura 6 – Exemplo de desenhos básicos em Java2D

```
public void paint(Graphics g) {
      Graphics2D g2d = (Graphics2D) g;
      // pega o tamanho da tela
      Dimension d = getSize();
      // limpa a área especificada, adicionando a cor de fundo padrão
      g2d.clearRect(0, 0, d.width, d.height);
      // adiciona a cor azul para a instância de Graphics2D
      g2d.setColor(Color.BLUE);
      // adiciona um BasicStroke (espessura da linha dos desenhos)
      g2d.setStroke(new BasicStroke(5));
      // o tamanho do JFrame é de 400x300px
      // desenha uma linha informando os valores inicial e final dos pontos
      g2d.drawLine(200, 10, 200, 290);
      g2d.setColor(Color.RED);
      // desenha um retângulo informando o ponto superior esquerdo, largura
      // e altura
      g2d.drawRect(10, 75, 390, 100);
      g2d.setColor(Color.GREEN);
      // desenha uma elipse informando o ponto superior esquerdo, largura
      // e altura
      g2d.drawOval(50, 100, 200, 200);
}
```

Quadro 1 – Código de exemplo do Java 2D

Para a manipulação de imagens, uma das classes mais utilizadas é a `BufferedImage`. Segundo Java Platform SE 6 (2008), a classe `BufferedImage` é uma subclasse de `Image` com um *buffer* que é capaz de acessar os dados da imagem. Um `BufferedImage` é composto por um objeto da classe `ColorModel`[6] e um objeto da classe `Raster`[7] de dados da imagem. Todos os objetos de `BufferedImage` têm as coordenadas iniciais no topo esquerdo.

2.5.1 JAI – Java *advanced imaging* API

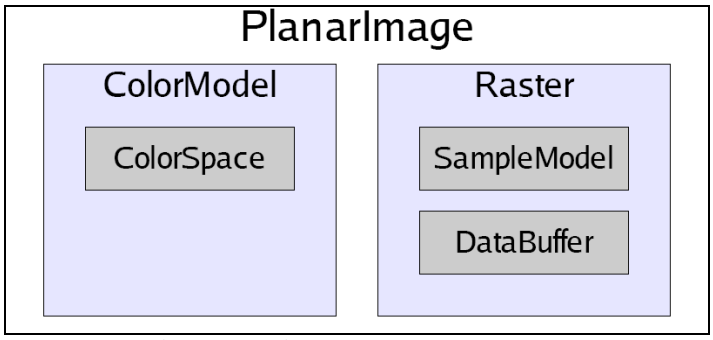
De acordo com Sun Developer Network (2008), o JAI proporciona funcionalidades de processamento de imagem com alto desempenho que podem ser acoplados em Java *applets* e aplicações. O JAI vai além das funcionalidades tradicionais das APIs de imagem, é uma plataforma independente e um *framework* extensível de processamento de imagens.

Conforme Santos (2004), a classe `PlanarImage` é a classe base para representação de imagens no JAI e é mais flexível que o `BufferedImage`. Ambas agregam várias classes para fazer com que a representação da imagem se torne flexível. A estrutura da classe

[6] Encapsula métodos para traduzir um valor de *pixel* para componentes de cor, por exemplo, RGB.

[7] Define valores para *pixels* que ocupam uma área retangular de um plano, não necessariamente se iniciando em (0, 0).

`PlanarImage` se encontra na Figura 7.



Fonte: Santos (2004, pg. 3).

Figura 7 – Estrutura da classe `PlanarImage`

Seus *pixels* são armazenados em uma instância de `Raster`, no qual contém uma instância de `DataBuffer` que é criada de acordo com regras descritas pela instância de `SampleModel`. Uma instância de `PlanarImage` também tem um `ColorModel` associado a ele, no qual contém uma instância de `ColorSpace` que determina como um valor de *pixel* pode ser traduzido para valores de cor.

Uma instância de `PlanarImage` é de apenas leitura. Seus *pixels* podem ser lidos de diferentes maneiras, entretanto não há métodos que permitam a modificação dos seus valores. A origem da imagem associada ao `PlanarImage` pode ser diferente de (0, 0) incluindo valores negativos.

Santos (2004) ainda afirma que o JAI apresenta vários operadores de imagem que podem ser aplicados com o mínimo de programação. Um exemplo de código para inversão de cores da imagem pode ser observado no Quadro 2. Em apenas duas linhas se consegue carregar a imagem e obter o resultado da inversão.

```
// lê a imagem e assume args[0] como sendo o caminho para a imagem
PlanarImage input = JAI.create("fileload", args[0]);
// inverte a imagem
PlanarImage output = JAI.create("invert", input);
```

Fonte: adaptado de Santos (2004, pg. 8).

Quadro 2 – Código para inversão de imagem com o JAI

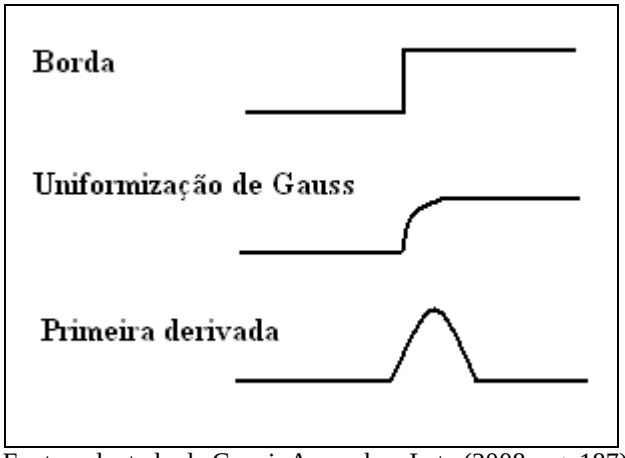
Para visualizar as imagens e facilitar na construção da GUI, existe uma classe chamada `DisplayJAI`. Conforme Java Advanced Imaging API (2007), o `DisplayJAI` não é nada mais que um `JPanel` que é capaz de conter uma imagem. A imagem e o seu *container* podem ter diferentes tamanhos, e a imagem pode ser posicionada dentro do *container*.

2.6 OPERADOR DE CANNY

O operador de Canny é um filtro de convolução de primeira derivada, que minimiza os ruídos e identifica as bordas de imagens digitais mesclando um operador diferencial com um filtro gaussiano[8] (CONCI; AZEVEDO e LETA, 2008, pg. 187). De acordo com Price (2004) as características do operador de Canny são:

a) detecção: tem a habilidade de localizar e demarcar todas as bordas reais;
b) localização: consegue identificar as bordas com uma distância mínima entre a imagem real e o encontrado;
c) resposta: apenas uma por borda.

Ainda Conci, Azevedo e Leta (2008, pg. 187) dizem que levando-se em conta uma borda de uma dimensão, que é representada na Figura 8, se aplicada a função gaussiana à borda, tem-se uma variação contínua do valor inicial ao final com uma inclinação máxima no ponto em que há o degrau. Se essa continuidade é diferenciada em relação à x, essa inclinação máxima será o máximo da nova função em relação à original. A Figura 8 mostra esse resultado (primeira derivada) e o passo intermediário (uniformização de Gauss).



Fonte: adaptado de Conci, Azevedo e Leta (2008, pg. 187).
Figura 8 – Processo de detecção de bordas por Canny

Os máximos da convolução da máscara e da imagem indicarão bordas na imagem. Esse procedimento pode ser feito pelo uso de uma função gaussiana em duas dimensões (x e y). Os valores das máscaras gaussianas variam de acordo com a escolha do desvio padrão. A função gaussiana em uma dimensão pode ser vista no Quadro 3, onde σ é o desvio padrão, responsável pelo controle do grau de suavização.

[8] Serve para remover ruídos e suavizar a imagem.

$$g(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-x^2}{2\sigma^2}}$$

Fonte: adaptado de Price (2004).

Quadro 3 – Função gaussiana em uma dimensão

O objetivo do filtro de Canny para a detecção de bordas é utilizar $g'(x)$ que pode ser encontrada no Quadro 4.

$$g'(x) = \frac{-x}{\sigma^3\sqrt{2\pi}} e^{\frac{-x^2}{2\sigma^2}}$$

Quadro 4 – Primeira derivada da função gaussiana em 1D

Segundo Conci, Azevedo e Leta (2008, pg. 187), convoluindo-se a imagem com $g'(x)$, tem-se uma imagem I que exibirá as bordas, mesmo na presença de ruído. A convolução em 2D pode ser realizada através de duas convoluções com a máscara de Gauss de uma dimensão, ou seja, fazer primeiro uma convolução em 1D na direção x e depois usar a mesma máscara 1D da gaussiana na direção y.

Conforme Price (2004), o *thresholder* utilizado no operador de Canny utiliza um método chamado "histerese". A maioria dos *thresholders* utilizam um único limite para o *threshold* no qual significa que as bordas irão ser exibidas dependendo da comparação do valor da borda com o limite dado ao *threshold*. O método histerese implica em fixar um valor máximo e mínimo para a comparação com a borda. Considerando um segmento de reta, se um valor está acima do limite superior do *threshold*, ela é imediatamente aceita. Se o valor se situa abaixo do limite inferior do *threshold*, ela é imediatamente rejeitada. Pontos que estiverem situados entre os dois limites são aceitos se eles estiverem conectados a *pixels* que exibirem uma resposta "positiva", ou seja, que tem valores acima do *threshold* de limite superior. Um exemplo de resultado obtido através da aplicação do filtro de Canny pode ser observado na Figura 9.

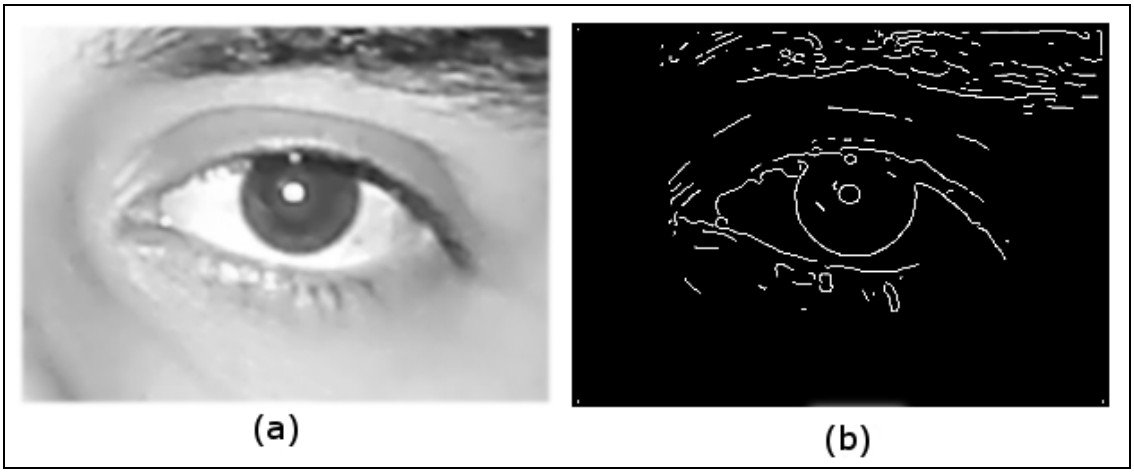
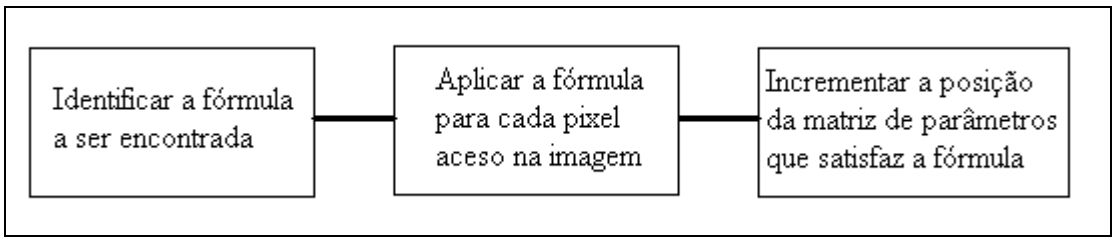


Figura 9 – (a) imagem original; (b) aplicação do filtro de Canny

2.7 TRANSFORMADA DE HOUGH

A transformada de Hough foi criada por Paul Hough no início dos anos 60. É uma técnica para reconhecimento de formas em imagens digitais que têm equações com fórmulas conhecidas, tais como retas, círculos e elipses. A partir da primeira publicação, surgiram outros trabalhos melhorando e até aplicando a transformada a formas generalizadas (CONCI; AZEVEDO e LETA, 2008, pg. 241).

O objetivo da transformada de Hough é transformar a imagem do espaço digital (x, y) em uma representação na forma dos parâmetros descritos pela curva que se deseja identificar na imagem. Essa transformação é aplicada de maneira em que todos os pontos pertencentes a uma mesma curva sejam mapeados num único ponto no espaço dos parâmetros da curva desejada. Para isso, o espaço dos parâmetros é discretizado e representado como uma matriz de inteiros, onde cada posição da matriz equivale à um intervalo no espaço real dos parâmetros. Cada ponto da imagem que satisfizer a equação paramétrica procurada incrementa uma unidade o contador na sua posição da matriz. No final do processo o contador que tiver o valor mais alto corresponde aos parâmetros da curva descrita na imagem. O processo para aplicação da transformada de Hough para qualquer forma geométrica é sintetizado na Figura 10.



Fonte: adaptado de Conci, Azevedo e Leta (2008, p. 242).

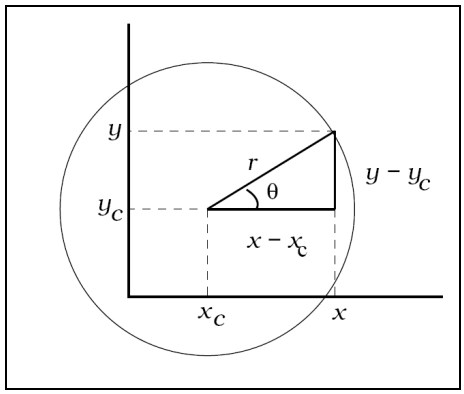
Figura 10 – Etapas da transformada de Hough para qualquer forma geométrica

Para aplicar esta técnica, geralmente utiliza-se de um pré-processamento de imagem com o objetivo de identificar os contornos dos objetos que a compõem, por exemplo, utilizando-se o operador de Canny.

2.7.1 Transformada de Hough para formas circulares

A circunferência é o lugar geométrico dos pontos do plano eqüidistantes de um ponto

fixo, chamado centro. A distância entre o centro e qualquer ponto da circunferência chama-se raio. A Figura 11 representa uma circunferência com raio e ponto central variável.



Fonte: Pistori, H., Pistori, J. e Costa (2005, p. 2).

Figura 11 – Circunferência de raio r e centro x_c, y_c

Conforme Conci, Azevedo e Leta (2008, pg. 242), na localização de círculos usando a transformada de Hough, pode-se fazer uso da fórmula implícita da circunferência, localizada no Quadro 5, onde x_0 e y_0 são as coordenadas cartesianas do centro do círculo e r o seu respectivo raio.

$$(x - x_0)^2 + (y - y_0)^2 = r^2$$

Fonte: adaptado de Conci, Azevedo e Leta (2008, pg. 246).

Quadro 5 – Fórmula implícita da circunferência

Porém, com a fórmula implícita do Quadro 5 é difícil de evidenciar as coordenadas do centro x_0 e y_0 como função das demais variáveis. De acordo com Pistori, H., Pistori, J. e Costa (2005, pg. 3), a fórmula mais utilizada é baseada em uma representação utilizando coordenadas polares, na qual pode ser vista no Quadro 6.

$$\begin{aligned} &x_0 = x - \rho\cos\theta \\ &y_0 = y - \rho sen\theta \\ &onde: \\ &\rho = r \\ &\cos(\theta) = \frac{x}{r} \\ &sen(\theta) = \frac{y}{r} \end{aligned}$$

Quadro 6 – Fórmula da circunferência baseada em coordenadas polares

Desta maneira, para obter os valores x e y já conhecendo as coordenadas de centro da circunferência, basta explicitar x e y.

$$\begin{aligned} x &= x_0 + \rho\cos\theta \\ y &= y_0 + \rho sen\theta \end{aligned}$$

Fonte: adaptado de Conci, Azevedo e Leta (2008, p. 247).

Quadro 7 – Fórmula da circunferência baseada em coordenadas polares já conhecendo as coordenadas do centro de círculo

Para utilizar a transformada de Hough deve-se previamente descobrir o valor real do raio de círculo, ou pelo menos saber o intervalo de raio. Segundo Pistori, H., Pistori, J. e Costa (2005, pg. 2), tem-se como entrada um conjunto de coordenadas (x, y) e deseja-se encontrar prováveis valores para os parâmetros (x_0, y_0), nos quais correspondem aos pontos centrais das circunferências. Para isso, é necessário construir um "espaço de Hough", que pode ser análogo a uma matriz com a mesma dimensão da imagem digital, em que as colunas e linhas representam os prováveis valores de x_0 e y_0. Cada célula da matriz tem como valor inicial zero, e para cada ponto (x, y) da imagem, é incrementada, no espaço de Hough, uma unidade em todas as células (x_0, y_0) que representam centros de circunferências de raio r. No final do processo, as células que contiverem os valores mais elevados indicam o ponto central da circunferência. O pseudocódigo para a criação do espaço de Hough encontra-se no Quadro 8.

```
entrada: Matriz I, nxm, representando a imagem binarizada.
saída: Matriz H, com o mesmo tamanho da imagem, representando o espaço de
Hough.
   para x = 0 até n faça
      para y = 0 até m faça
         se I(x,y) = 255 então
            para θ = 0 até 2*π faça
               x0 = x − r * cos(θ)
               y0 = y − r * sen(θ)
               H(x0,y0) = H(x0,y0) + 1
            fim para
         fim se
      fim para
   fim para
```

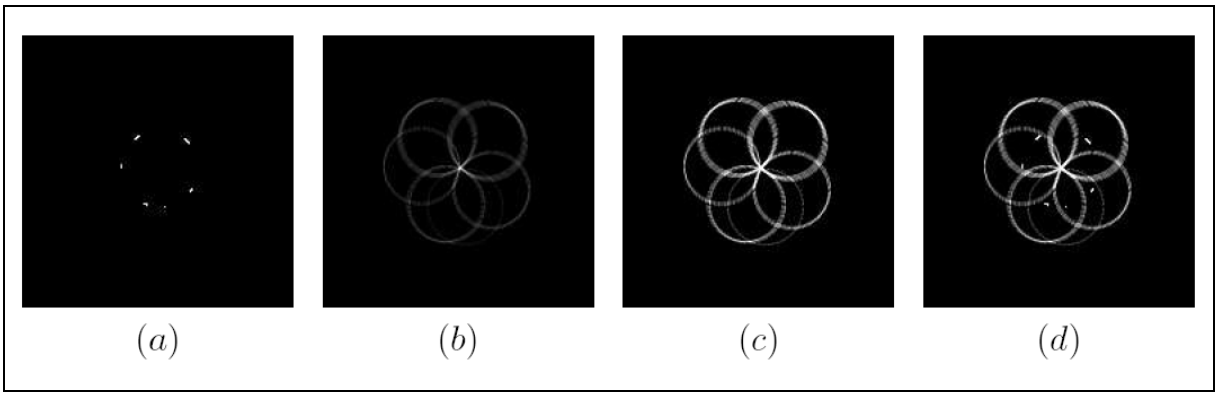
Fonte: adaptado de Pistori, H., Pistori, J. e Costa (2005, p. 3).

Quadro 8 – Pseudocódigo para criação do espaço de Hough

O pseudocódigo do Quadro 8 mostra como pode ser criado um espaço de Hough H a partir de uma imagem digital I. Pistori, H., Pistori, J. e Costa (2005, pg. 3) ainda afirmam que após a criação do espaço, a detecção da circunferência passa a ser um problema simples para encontrar os pontos de máximo no espaço de Hough. Vale salientar que o conceito de transformada de Hough se aplica somente quando é possível distinguir os pontos na imagem pertencentes ao contorno, ou borda, dos objetos. Então, o pseudocódigo do Quadro 8 assume

que a imagem é previamente processada por um filtro de detecção de bordas e por um filtro de binarização.

A Figura 12 mostra o espaço de Hough através de uma imagem em tons de cinza. O brilho é diretamente equivalente ao valor acumulado em cada célula da matriz. Esse espaço de Hough corresponde à imagem que contém uma circunferência altamente corrompida que pode ser visualizada na Figura 12 (a). O espaço de Hough pode ser visualizado na Figura 12 (c). O espaço de Hough foi construído com o valor de raio idêntico ao raio do círculo original. É possível perceber que o valor máximo, mais claro, do espaço equivale ao centro da circunferência presente na imagem. As demais circunferências correspondem aos centros das prováveis circunferências. Este efeito destaca-se mais na imagem, pois faz a combinação do espaço de Hough com a imagem original.



Fonte: Pistori, H., Pistori, J. e Costa (2005, p. 3).

Figura 12 – (a) Exemplo de uma imagem; (b) seu respectivo espaço de Hough; (c) o mesmo espaço de hough com ajuste de contraste para facilitar a visualização e (d) uma imagem composta pela adição da imagem original com o espaço de Hough

2.8 TRABALHOS CORRELATOS

Dentre os trabalhos pesquisados, os que mais se assemelham com o presente trabalho são: Biometria com Enfoque em Reconhecimento de Íris (PRADO JR., 2005) e Processamento e Reconhecimento de Imagens Digitais da Retina Humana (OSAWA, 2004).

2.8.1 Biometria com enfoque em reconhecimento de íris

Segundo Prado Jr. (2005, p. 10), a biometria aliada com a tecnologia da informação pode ser realizada através de mensurações fisiológicas ou comparação de comportamento e peculiaridades dos seres com o intuito de identificá-los.

O trabalho visa reconhecer a íris do olho humano para garantir a autenticidade da pessoa. Prado Jr. (2005, p. 37) afirma que para reconhecer a íris são necessárias várias etapas:

a) após a aquisição da imagem, é necessário isolar a região da íris na imagem digital do olho;
b) a região da íris pode ser aproximada por dois círculos: um para o limite da esclera[9]/íris e outro com limites interiores ao primeiro para a íris/pupila;
c) os cílios e as pálpebras normalmente fazem a oclusão das partes superiores e inferiores da região da íris, corrompendo os dados da mesma;
d) outras técnicas devem ser utilizadas para isolar e excluir as partes do item c, assim como para identificar a região ocular.

Após o reconhecimento da íris, são aplicadas técnicas de codificação visando o seu reconhecimento biométrico.

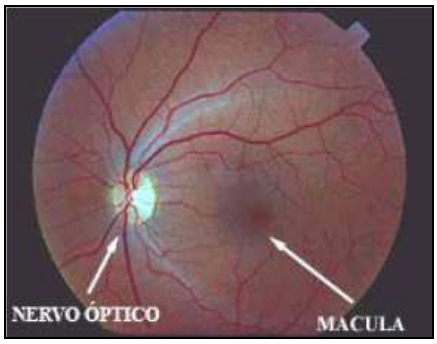
2.8.2 Processamento e reconhecimento de imagens digitais da retina humana

Osawa (2004, p. 13) afirma que o trabalho visa empregar técnicas de processamento de imagens em uma imagem de exame de retinografia. O objetivo é identificar componentes da retina em busca de possíveis irregularidades. Dentre eles destacam-se a detecção do nervo óptico, mácula e exsudatos[10]. A Figura 13 apresenta uma foto de uma retina normal, onde são apontados o nervo óptico e a mácula. O processo é realizado através de filtros de processamento de imagens.

---

[9] É a parte branca dos olhos.

[10] Segundo Osawa (2004, p. 17) gordura e proteína podem vazar e se depositar em placas na retina, conhecidas como exsudatos.



Fonte: Osawa (2004, f. 18).

Figura 13 – Foto de uma retina normal

Sintetizando, o trabalho foi desenvolvimento utilizando as seguintes técnicas:

a) foi realizado um pré-processamento a fim de delimitar a área a ser tratada;
b) utilizou-se a camada RED de uma imagem RGB, pois nela os vasos sanguíneos não são muito aparentes, assim facilitando a detecção do nervo óptico;
c) para detectar a região do nervo óptico foram utilizadas as técnicas de Otsu e Sobel;
d) segundo Osawa (2004, p. 41) para identificar a mácula é realizada uma busca nas proximidades da região do nervo óptico, visto que as duas estruturas estão separadas por uma distância pequena, aproximadamente duas vezes o diâmetro do disco óptico;
e) para a identificação dos exsudatos, utilizou-se a camada GREEN da imagem RGB devido ao fato de os exsudatos possuírem alta intensidade de cor nesta camada. Também foi utilizada a técnica de limiarização dinâmica para calcular um limiar de diferenciação entre o fundo da imagem e um exsudato, com isso alcançando um resultado mais preciso.

3 DESENVOLVIMENTO DA FERRAMENTA

Este capítulo detalha as etapas do desenvolvimento da ferramenta. São ilustrados os principais requisitos, a especificação, a implementação (mencionando técnicas e ferramentas utilizadas, bem como a operacionalidade da ferramenta) e por fim são listados resultados e discussão.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

A ferramenta proposta basicamente recebe como entrada uma imagem digital, calcula e exibe medidas eminentes ao estrabismo.

Sendo assim, foram identificados os seguintes requisitos:

a) recuperar uma imagem da face humana (Requisito Funcional - RF);
b) permitir que sejam definidas ROIs para análise da imagem (RF);
c) identificar a íris e o limbo do olho humano na imagem digital (RF);
d) identificar o reflexo do flash na córnea ocular na imagem digital (RF);
e) calcular a medida entre o reflexo da córnea e o limbo no sentido nasal, Figura 14 (a), e temporal horizontal, Figura 14 (b) (RF);
f) calcular o diâmetro da íris (RF);
g) realizar o diagnóstico do estrabismo a partir das medidas estabelecidas (RF);
h) gerar relatório do exame contendo as medidas extraídas (RF);
i) implementar a ferramenta utilizando a tecnologia Java em conjunto com a biblioteca JAI. (Requisito Não Funcional - RNF);
j) utilizar a biblioteca iText para a geração do relatório do exame em formato pdf (RNF).

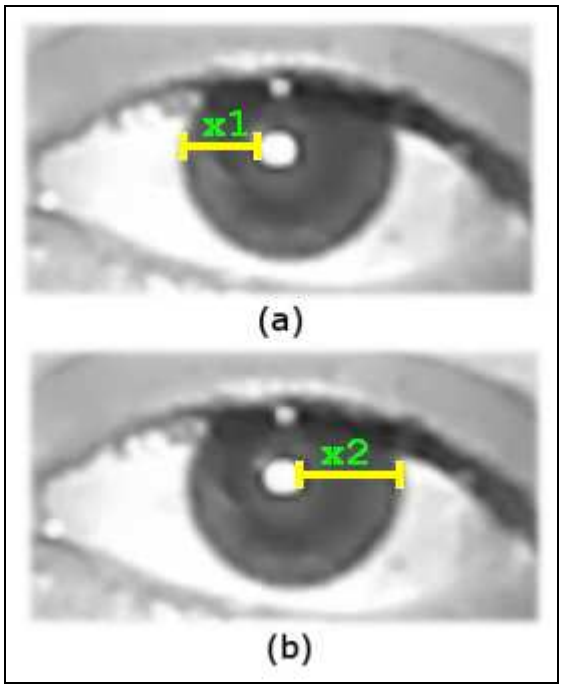


Figura 14 – (a) medida entre o reflexo da córnea e o limbo no sentido nasal “x1” e (b) medida entre o reflexo da córnea e o limbo no sentido temporal “x2”. A figura é baseada no olho esquerdo do paciente

3.2 ESPECIFICAÇÃO

A especificação do presente trabalho foi desenvolvida utilizando a notação UML (UML, 2002) em conjunto com a ferramenta JUDE (JUDE, 2008). São explanados diagramas de casos de uso, classes e seqüência. Alguns diagramas estão em sua forma resumida para melhor visualização e isto não comprometerá o entendimento do trabalho.

A ferramenta recebeu o nome de *Strabismus Detector* devido às suas características. Para apresentar a especificação foram criados quatro módulos para serem descritos:

a) módulo de desenhos geométricos: responsável por armazenar dados e apresentar na tela desenhos geométricos utilizados no projeto;
b) módulo de processamento de imagem: responsável por conter os algoritmos de Hough e Canny, bem como operações auxiliares para processar a imagem;
c) módulo de interface gráfica: responsável pela interação do usuário com o programa, bem como a manipulação de eventos;
d) módulo de resultados: módulo que interage com a biblioteca iText para gerar o relatório do exame obtido através da ferramenta.

A Figura 15 apresenta o diagrama de casos de uso com as principais interações do

usuário com o sistema.

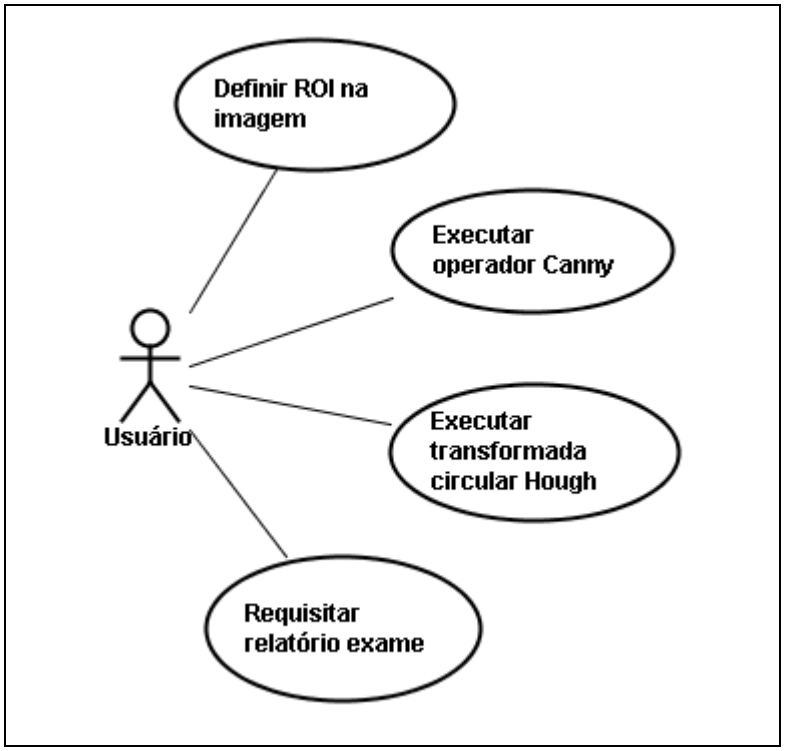


Figura 15 – Diagrama de casos de uso

O caso de uso `Definir ROI na imagem` (Quadro 9) descreve como o usuário poderá definir uma região de interesse para analisar a imagem de entrada. Este caso de uso possui apenas um cenário principal não havendo cenários alternativos ou de exceção.

Definir ROI na imagem: possibilita ao usuário definir interativamente na interface do sistema a região de interesse para análise da imagem.	
Pré-condição	Uma imagem deve estar carregada pelo sistema.
Cenário principal	1) O usuário seleciona a opção de definição de ROI. 2) O sistema apresenta o formato da seleção escolhida pelo usuário. 3) O usuário desloca a caixa de seleção (ROI) para o local desejado. 4) O sistema valida a posição.
Pós-condição	A ROI é definida com sucesso.

Quadro 9 – Caso de uso definir ROI na imagem

O segundo caso de uso, `Executar operador Canny` (Quadro 10), explana como o usuário pode executar o operador de Canny na imagem, tendo a possibilidade de informar parâmetros de filtragem. Além do cenário principal, este caso de uso possui um fluxo alternativo responsável por informar o que ocorrerá caso o usuário omita informações de filtragem.

Executar operador Canny: possibilita ao usuário executar o algoritmo de Canny tendo a possibilidade de informar os parâmetros de filtragem.	
Pré-condição	Uma imagem deve estar carregada pelo sistema.
Cenário principal	1) O usuário seleciona a opção para executar o operador de Canny podendo informar os parâmetros de filtragem. 2) O sistema exibe o resultado da filtragem.
Fluxo Alternativo 01	No passo 1, caso o usuário não informe os parâmetros de filtragem o sistema assume os valores *default* estabelecidos pelo desenvolvedor.
Pós-condição	O operador de Canny é executado com sucesso.

Quadro 10 – Caso de uso executar operador Canny

O terceiro caso de uso, Executar transformada de Hough (Quadro 11), explana como o usuário pode executar o algoritmo de Hough na imagem de entrada. Além do cenário principal, existem dois cenários alternativos.

Executar transformada circular Hough: possibilita ao usuário executar a transformada circular de Hough tendo a possibilidade de informar os parâmetros de filtragem.	
Pré-condição	Uma imagem deve estar carregada pelo sistema.
Cenário principal	1) O usuário seleciona a opção para executar o algoritmo de Hough podendo informar os parâmetros de identificação. 2) O sistema exibe o resultado da identificação.
Fluxo Alternativo 01	No passo 1, caso o usuário não informe os parâmetros de filtragem o sistema assume os valores *default* estabelecidos pelo desenvolvedor.
Fluxo Alternativo 02	No passo 1, caso alguma ROI esteja ativada, o sistema irá executar o algoritmo de Hough somente dentro dos limites da ROI.
Pós-condição	O algoritmo de Hough é executado com sucesso.

Quadro 11 – Caso de uso executar transformada circular Hough

O quarto e último caso de uso, Requisitar relatório exame (Quadro 12), trata da geração do relatório de exame que o usuário poderá solicitar. Por ordem cronológica, este caso de uso é o último que poderá ser requisitado pelo usuário. Ele possui apenas um cenário principal não havendo cenários alternativos ou de exceção. Sua execução pode ser vista através do diagrama de seqüência apresentado na Figura 16

Requisitar relatório exame: possibilita ao usuário salvar um relatório de exame em formato pdf.	
Pré-condição	O sistema deve ter identificado as duas íris na imagem.
Cenário principal	1) O usuário seleciona a opção de salvar os resultados obtidos pelo exame. 2) O sistema apresenta tela para salvamento do arquivo. 3) O usuário escolhe diretório e nome do arquivo. 4) O sistema salva relatório.
Pós-condição	O relatório de exame é gerado com sucesso.

Quadro 12 – Caso de uso requisitar relatório exame

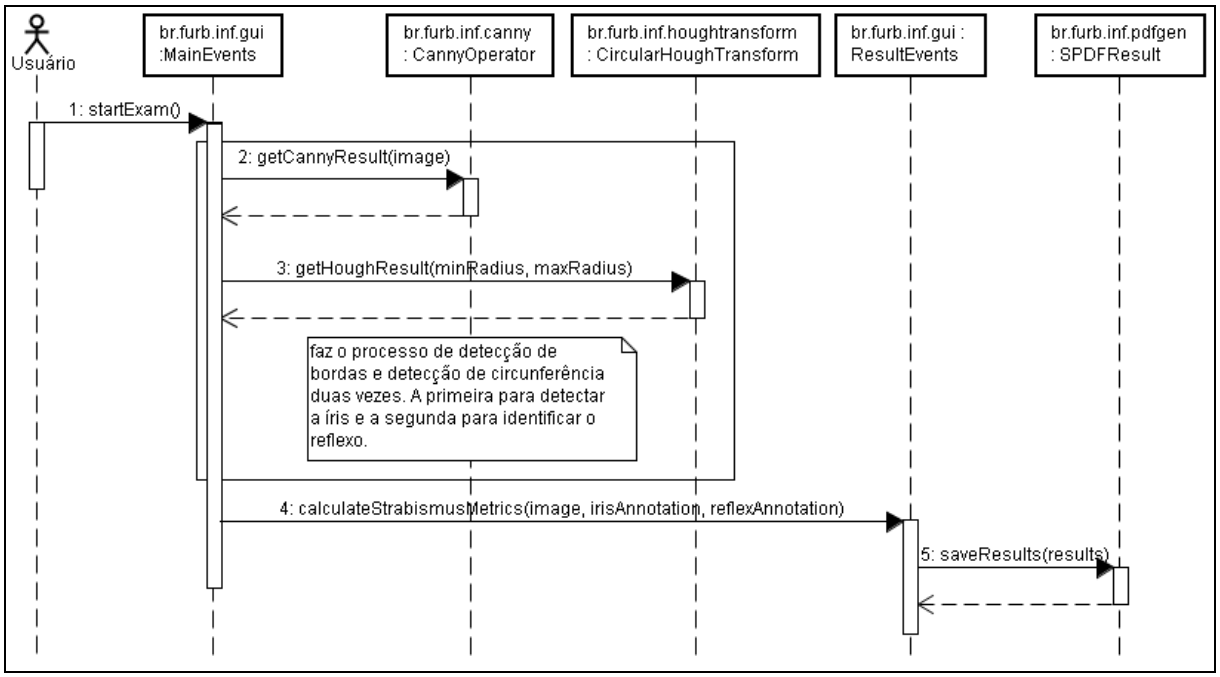


Figura 16 – Diagrama de seqüência para o caso de uso requisitar relatório exame

Para apresentar como as classes da ferramenta estão estruturadas e relacionadas utilizou-se o diagrama de classes. A Figura 17 apresenta, de uma maneira geral, como os pacotes da ferramenta interagem entre si através de ligações lógicas. A seguir serão explanados os módulos de desenvolvimento da ferramenta.

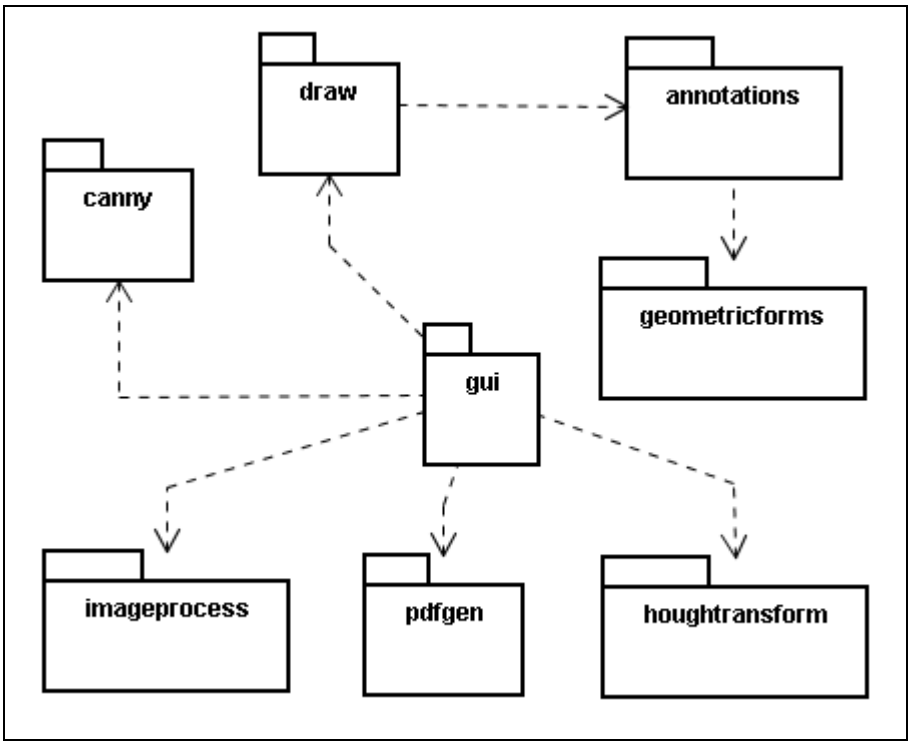


Figura 17 – Diagrama de pacotes da ferramenta

3.2.1 Módulo de desenhos geométricos

O módulo de desenhos geométricos tem como objetivo armazenar informações sobre determinada forma e exibir o respectivo desenho na tela para o usuário. A Figura 18 mostra de maneira macro o diagrama de classes do módulo de desenhos geométricos.

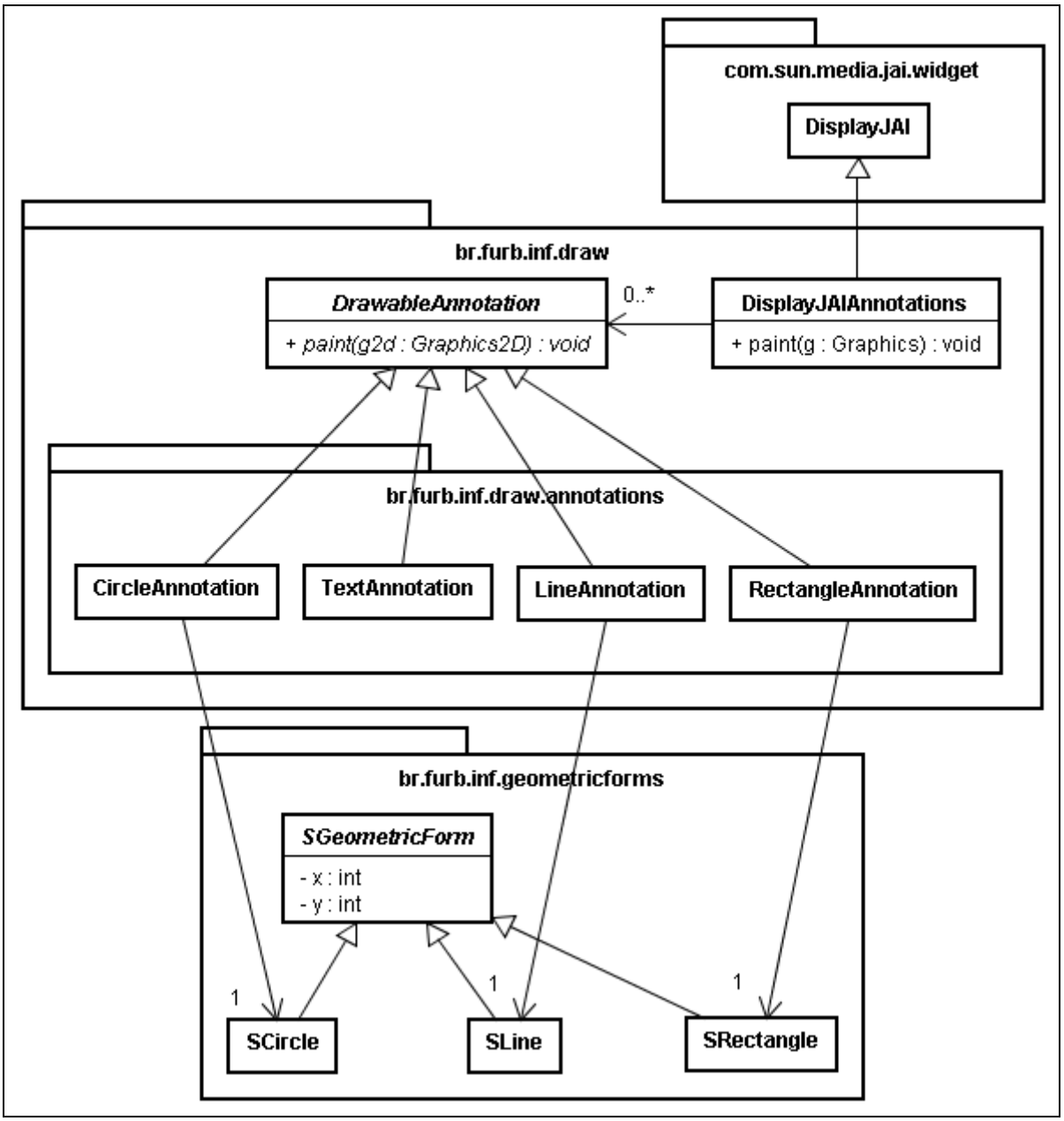


Figura 18 – Diagrama de classes do módulo de desenhos geométricos

Basicamente são três formas geométricas que a ferramenta utiliza: círculo, linha e retângulo. Analisando o pacote `br.furb.inf.geometricforms` da Figura 18, nota-se que existe uma classe geral `SGeometricForm` e três subclasses: `SCircle`, `SLine` e `SRectangle`. Para padronizar classes provenientes da ferramenta, utilizou-se da nomenclatura **`S`**`(nome da`

`classe`), aonde o **`S`** vem de *strabismus*. A classe `SGeometricForm` contém as coordenadas (x, y), além de outros atributos como cor e contorno do desenho. As subclasses de `SGeometricForm` contém atributos relevantes à sua forma, onde o `SCircle` contém o raio, `SLine` contém (x_2, y_2), e `SRectangle` possui a largura e a altura do retângulo. Agora olhando para o pacote `br.furb.inf.draw.annotations` da Figura 18, pode-se identificar as classes `CircleAnnotation`, `TextAnnotation`, `LineAnnotation`, e `RectangleAnnotation`. Todas estas classes são generalizadas através da classe abstrata `DrawableAnnotation` que contém o método `paint()`. As classes que herdam de `DrawableAnnotation` implementam o método `paint()` para a sua devida forma geométrica, assim explicando também o motivo de haver uma associação de `CircleAnnotation` com `SCircle`, `LineAnnotation` com `SLine` e assim por diante. No pacote `br.furb.inf.draw` pode-se observar a classe `DisplayJAIAnnotations` que também contém um método `paint()` que é responsável por desenhar todos os objetos descendentes de `DrawableAnnotation` que estão no projeto e evidentemente associados ao `DisplayJAIAnnotations`. Além disso, a classe `DisplayJAIAnnotations` é generalizada através de `DisplayJAI` que nada mais é que um `JPanel` que permite associar uma imagem a ele, no caso do projeto a imagem será a foto do usuário e os desenhos são realizados na foto.

3.2.2 Módulo de processamento de imagem

Este módulo tem como objetivo segmentar a imagem através do operador de Canny, identificar formas circulares pela transformada circular de Hough e prover de operações auxiliares para processar a imagem. A Figura 19 apresenta o diagrama de classes para o operador de Canny.

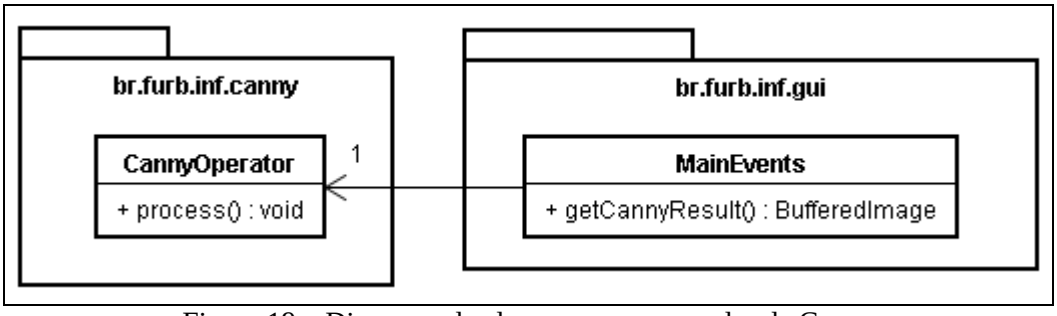


Figura 19 – Diagrama de classes para o operador de Canny

O pacote `br.furb.inf.canny` contém a classe `CannyOperator` proveniente ao

operador de Canny. A classe `CannyOperator` é responsável por pegar uma imagem como entrada, processar e retornar como saída uma imagem segmentada com as bordas detectadas. O método `process()` é o principal método desta classe na qual é responsável por organizar a seqüência de processamento que resume-se em:

a) computar os gradientes, gerando as máscaras de convolução gaussianas;
b) fazer a histerese;
c) identificar as bordas.

No pacote `br.furb.inf.gui` da Figura 19 tem-se a classe `MainEvents` com o respectivo método `getCannyResult()` no qual é responsável por executar o método de Canny na imagem de entrada selecionada pelo usuário.

A Figura 20 apresenta o diagrama de classes para a transformada circular de Hough onde a estrutura está similar ao diagrama de classes proveniente do operador de Canny.

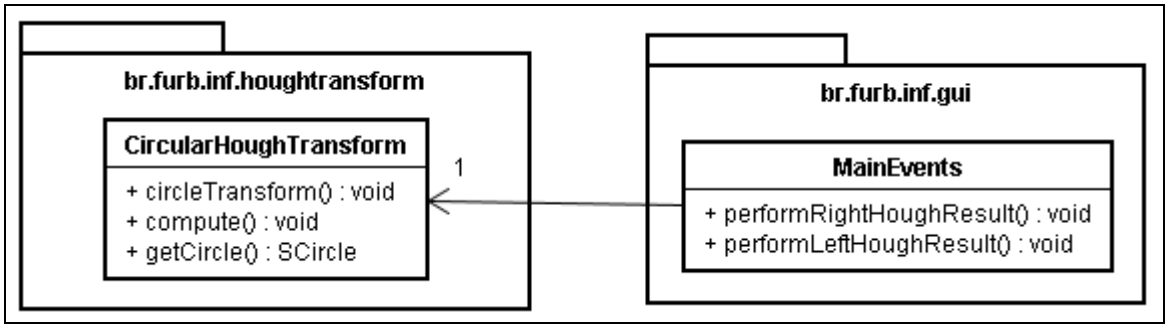


Figura 20 – Diagrama de classes para a transformada circular de Hough

No pacote `br.furb.inf.houghtransform` está localizada a classe `CircularHoughTransform` que é responsável por detectar formas circulares na imagem digital de entrada. O método `circleTransform()` é responsável por identificar as possíveis coordenadas de centro da circunferência. Já o método `compute()` é responsável por construir o espaço de Hough e o método `getCircle()` procura, através de um intervalo de raios, uma circunferência em conjunto com os dois métodos citados anteriormente. O pacote `br.furb.inf.gui` possui a classe `MainEvents` que é a mesma do diagrama da Figura 19. Aqui são exibidos os métodos provenientes da detecção de círculos da imagem digital de entrada, neste caso para o olho direito, `performRightRoughResult()`, e para o olho esquerdo, `performLeftHoughResult()`.

A Figura 21 exibe o diagrama de classes para operações auxiliares de processamento de imagens (`ImageUtils`). Nele são citados alguns métodos que são utilizados para o pré-processamento, entre outras finalidades. Os nomes dos métodos são de certa forma auto-explicativos, onde pode-se destacar o método `imageToPixels(...)`, no qual transforma a imagem digital de entrada em um único *array* de inteiros sendo esta a forma de entrada para o

algoritmo de Hough. Destaca-se também o método `getIrisImage(...)`, que recebe como parâmetro a imagem do rosto do usuário e as coordenadas do ponto central (x, y) e o raio (`SEye`) e retorna uma ROI contendo a íris.

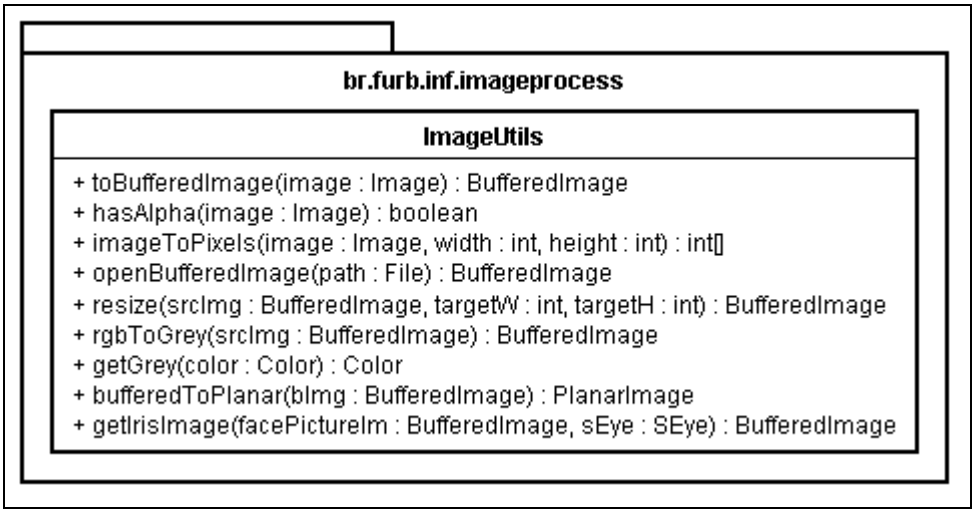


Figura 21 – Diagrama de classes para operações auxiliares de processamento de imagens

3.2.3 Módulo de interface gráfica

O módulo de interface gráfica tem como objetivo agregar as funcionalidades da ferramenta, bem como prover meios de o usuário interagir de uma maneira simples e interativa com o sistema. O diagrama de classes deste módulo pode ser analisado através da Figura 22. O pacote `br.furb.inf.gui` contém classes de interface utilizando componentes `javax.swing`, bem como classes que tratam de implementar possíveis eventos que venham a ser disparados pelo usuário. As classes abstratas `ResultGUI`, `MainGUI` e `AboutGUI` são responsáveis por conter os componentes de interface swing. Também possuem toda a estrutura de posicionamento e *layout*, porém sem nenhum tratador de eventos. A classe `MainGUI` é responsável pela tela principal do sistema, `ResultGUI` pela tela de resultados proveniente das medidas do estrabismo e `AboutGUI` pela tela "sobre" da ferramenta.

As classes responsáveis por tratar os eventos da ferramenta são: `ResultEvents`, `MainEvents`, `AboutEvents` e `DisplayJAIEvents`. A classe `MainEvents` é responsável por reunir praticamente todas as funcionalidades do sistema, implementando *listeners* nos componentes `javax.swing` de `MainGUI`. Além disso, é a principal classe tratadora de eventos da ferramenta. `ResultEvents` e `AboutEvents` utilizam a mesma ideologia, implementando

listeners nos componentes de `ResultGUI` e `AboutGUI` respectivamente.

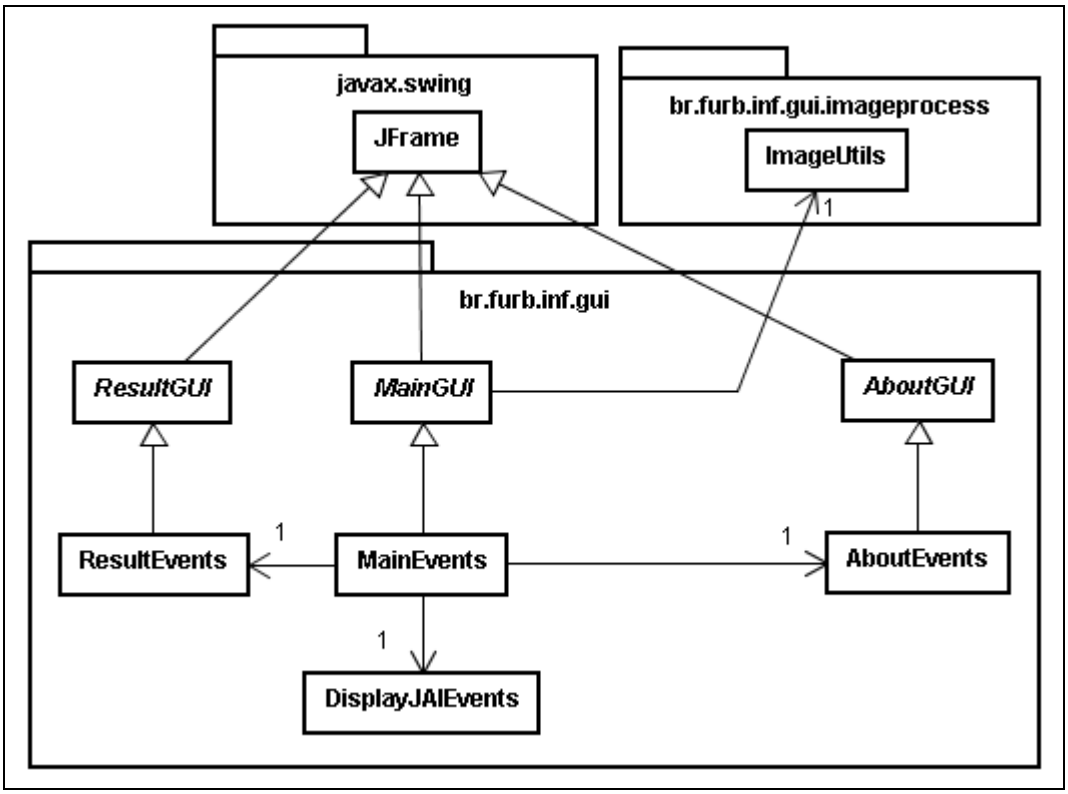


Figura 22 – Diagrama de classes do módulo de interface gráfica

Já a classe `DisplayJAIEvents` é responsável por implementar o `java.awt.event.MouseListener` e o `java.awt.event.MouseMotionListener` no *canvas* da aplicação que nada mais é que uma instância de `DisplayJAIAnnotatitions` associado ao `JFrame` (`MainGUI`). Esta implementação é devida aos desenhos interativos que servem para definir ROIs.

3.2.4 Módulo de resultados

O objetivo deste último módulo é exibir os resultados das medidas em tela para o usuário e também fornecer meios para que o usuário salve este resultado em um arquivo PDF. O diagrama de classes do módulo de resultados pode ser observado através da Figura 23. Como já citado anteriormente, a classe `ResultEvents` do pacote `br.furb.inf.gui` é responsável por implementar *listeners* de eventos e exibir em tela os resultados das medidas do estrabismo. Porém, ela agrega uma outra funcionalidade que é representada através da classe `SPDFResult` do pacote `br.furb.inf.pdfgen`. Esta classe é responsável por gerar o

resultado do exame em um arquivo PDF com auxílio da biblioteca iText (ITEXT, 2008).

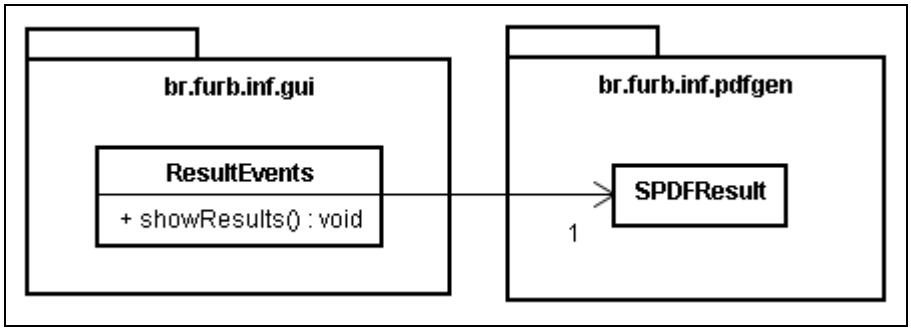


Figura 23 – Diagrama de classes do módulo de resultados

3.3 IMPLEMENTAÇÃO

A seguir são exibidas as técnicas e ferramentas utilizadas e a operacionalidade da implementação da ferramenta.

3.3.1 Técnicas e ferramentas utilizadas

Para a implementação da ferramenta foi utilizada a linguagem de programação Java, contando com suas extensões Java2D e JAI. O ambiente de desenvolvimento escolhido foi o Eclipse (ECLIPSE, 2008). Para o desenvolvimento da interface gráfica utilizou-se o *plugin* Jigloo (JIGLOO, 2007) para o Eclipse.

3.3.1.1 Detecção de íris e reflexo na córnea ocular

A detecção da íris e do reflexo na córnea ocular ocorre principalmente pela detecção de bordas, que é realizada através da classe `CannyOperator` e pela transformada circular de Hough, implementada através da classe `CircularHoughTransform`. Pelo Quadro 13 é possível analisar o código fonte do método `getCircle()` da classe `CircularHoughTransform` que, através de um intervalo de raio, consegue identificar uma forma circular na imagem digital.

```
/**
* Get the coordinates of a circular form in the image.
* @param image the unidimensional image array
* @param imagewidth the width of the image
* @param imageheight the height of the image
* @param minRadius the minimum radius of the circle to be found
* @param maxRadius the maximum radius of the circle to be found
* @return the circle within the best radius interval.
*/
public SCircle getCircle(int image[], int imagewidth, int imageheight,
float minRadius, float maxRadius){
     int bestCenterx = 0;
     int bestCentery = 0;
     float bestRadius = 0;
     double max = Double.MIN_VALUE;
     //for all radius min <-> max (incremented by 0.5)
     for(float i = minRadius; i <= maxRadius; i += 0.5){
          circleTransform(image, imagewidth, (double)i);
          for(int y = 0; y < height; y++){
               for(int x = 0; x < width; x++){
                    if((double)values[y * width + x] > max){
                        max = values[y * width + x];
                        bestCenterx = x;
                        bestCentery = y;
                        bestRadius = i;
                    }
                }
            }
        }
     return new SCircle(bestCenterx, bestCentery, (float)bestRadius);
 }
```

Quadro 13 – Código fonte do método `getCircle()` que identifica uma forma circular na imagem digital através de um intervalo de raio

O intervalo de raio é incrementado por 0.5 *pixels*[11] a cada iteração. A melhor coordenada de centro (`besterCenterX` e `bestCenterY`) e o melhor raio (`bestRadius`) são determinados pelo número máximo de votos (`max`), ou seja, o maior número de pontos de intersecção no espaço de Hough.

No Quadro 14 encontra-se o código fonte do método `circleTransform()` que também é proveniente da classe `CircularHoughTransform`. Nele são calculados e armazenados os valores de $\rho * \cos\theta$ e $\rho * sen\theta$ que são parte da fórmula da circunferência baseada em coordenadas polares (Quadro 6) e utilizados pelo método `compute()` (Quadro 15) para construir o espaço de Hough.

[11] Este valor foi escolhido devido ao seu tempo de resposta e precisão ao executar o método através de testes realizados.

```
/**
* Try to find the circle's center coordinate.
* @param image the unidimensional image array
* @param imagewidth the width of the image
* @param radius the circle's radius
*/
public void circleTransform(int image[], int imagewidth, double radius) {
     this.radius = radius;
     //circle perimeter
     int amax = (int)Math.round(2 * Math.PI * radius);
     int a[][] = new int[amax][2];
     int i = 0;
     for(int j = 0; j < a.length; j++) {
          double theta = j/radius;
          int rhoj = (int)Math.round(radius * Math.cos(theta));
          int thetaj = (int)Math.round(radius * Math.sin(theta));
          if(i == 0 || rhoj != a[i][0] && thetaj != a[i][1]) {
               a[i][0] = rhoj;
               a[i][1] = thetaj;
               i++;
          }
     }
     compute(a, image, imagewidth);
}
```

Quadro 14 – Código fonte do método `circleTransform()`, responsável por identificar possíveis coordenadas de centro da circunferência

O método `compute()` de `CircularHoughTransform` analisa os *pixels* "acesos" na imagem, ou seja, que tem um valor acima do *threshold* estipulado e incrementa a matriz acumuladora de votos (`values`) referente aos pontos centrais da circunferência.

```
/**
* Generate the hough space.
* @param a the 'rho' and 'theta' values
* @param pixels the image in pixels
* @param pixelswidth the same width of the image.
*/
public void compute(int a[][], int pixels[], int pixelswidth) {
     values = new float[height * width];
     for(int y = 0; y < height; y++) {
          for(int x = 0; x < width; x++) {
               //pixels displacement in the unidimensional matrix
               double d = pixels[(y+yoffset)*pixelswidth+(x+offset)];
               if(d > threshold){
                    for(int i = 0; i < a.length; i++){
                         int centerj = y + a[i][0];
                         int centeri = x + a[i][1];
                         if(centerj < 0 ||
                            centerj >= height ||
                            centeri < 0 ||
                            centeri >= width){
                           continue;
                         }
                         values[centerj*width+centeri]++;
                    }
               }
          }
     }
}
```

Quadro 15 – Código fonte do método responsável por construir o espaço de Hough

O método `process()` (Quadro 16) da classe `CannyOperator` é responsável por organizar a seqüência de chamada de métodos para realizar a detecção de bordas por pelo operador de Canny.

```
/**
* Process the Canny operator into an image
*/
public void process() {
     width = sourceImage.getWidth();
     height = sourceImage.getHeight();
     picsize = width * height;
     initArrays();
     readLuminance();
     computeGradients(gaussianKernelRadius, gaussianKernelWidth);
     int low = Math.round(lowThreshold * MAGNITUDE_SCALE);
     int high = Math.round( highThreshold * MAGNITUDE_SCALE);
     performHysteresis(low, high);
     thresholdEdges();
     writeEdges(data);
}
```

Quadro 16 – Código fonte do método `process()` do operador de Canny

Nele são chamados os métodos:

a) `initArrays()`: são inicializados os *arrays* de magnitude, convolução e gradientes;

b) `readLuminance()`: preenche o *array* de dados da imagem conforme o tipo de arquivo de entrada;

c) `computeGradients()`: este é o principal método porque além de gerar as máscaras de convolução gaussianas, ele também realiza a convolução nas direções *x* e *y* da imagem;

d) `performHysteresis()`: como o próprio nome já diz, é realizado o método da histerese no *array* de dados da imagem de acordo com os valores dos atributos `low` e `high` referentes ao *threshold*;

e) `thresholdEdges()`: torna a imagem binária;

f) `writeEdges()`: salva o *array* resultante do processamento em um `BufferedImage`.

3.3.1.2 Anotações na imagem através de desenhos geométricos

Para armazenar valores e desenhar formas geométricas na tela são utilizadas as classes da Figura 18. O Quadro 17 apresenta o código fonte da classe `DrawableAnnotation` que contém apenas o método abstrato `paint()` que deve ser implementado por suas subclasses.

```
package br.furb.inf.draw;
import java.awt.Graphics2D;
public abstract class DrawableAnnotation {
     /**
      * Paints a geometric form.
      */
     public abstract void paint(Graphics2D g2d);
}
```

Quadro 17 – Código fonte da classe `DrawableAnnotation`

O Quadro 18 mostra o código fonte da implementação do método `paint()` da classe `LineAnnotation`. As coordenadas estão guardadas no objeto `sLine`, que nada mais é que uma instância da classe `SLine`. A mesma lógica serve para as demais anotações de formas geométricas.

```
/**
*  Concrete implementation of the paint method.
*/
public void paint(Graphics2D g2d) {
     g2d.setStroke(sLine.getStroke());
     g2d.setColor(sLine.getColor());
     int initialX = sLine.getX();
     int initialY = sLine.getY();
     int finalX = sLine.getX2();
     int finalY = sLine.getY2();
     //draw the line
     g2d.drawLine(initialX, initialY, finalX, finalY);
}
```

Quadro 18 – Código fonte do método `paint()` da classe `LineAnnotation`

O método que desenha todas as formas na tela é o `paint()` (Quadro 19) da classe `DisplayJAIAnnotations`. As anotações de desenho ficam armazenadas em um `ArrayList` do tipo `DrawableAnnotation` dentro da classe `DisplayJAIAnnotations`. O método `paint()` desta classe apenas itera através do `ArrayList` e executa o método `paint()` da anotação corrente, e este por sua vez é exibido na tela por meio do `DisplayJAI`.

```
/** This method paints the component and all his annotations. */
public void paint(Graphics g) {
     super.paint(g);
     Graphics2D g2d = (Graphics2D) g;
     for(DrawableAnnotation element : annotations){
          element.paint(g2d);
     }
}
```

Quadro 19 – Código fonte do método `paint()` da classe `DisplayJAIAnnotations`

3.3.1.3 Desenhos interativos de ROIs e extração de medidas do estrabismo

Para o usuário poder ter uma maior interação com a ferramenta, os desenhos de seleção

de região de interesse são interativos, ou seja, o usuário poderá mover o componente de seleção e modificar a escala dependendo do tipo de componente. Um exemplo de código fonte para mover a região de interesse em forma retangular pode ser observado no Quadro 20, através do método `translateRectROI()`.

```
/**
* Make the rectangle translation
* @param evt the mouse event
*/
public void translateRectROI(MouseEvent evt){
     if(display.getCursor().getType() == moveRectCursor.getType()){
          RectangleAnnotation rectAnnotation = mainGUI.rectROIAnnotation;
          display.removeAnnotation(rectAnnotation);
          SRectangle sRect = rectAnnotation.getSRect();
          //the variance between the mouse drag
          int xDisplacement = evt.getX() - xMouse;
          int yDisplacement = evt.getY() - yMouse;
          //the initial rect coordinates
          int actualX1 = sRect.getX();
          int actualY1 = sRect.getY();
          //the new points to be drawn
          actualX1 += xDisplacement;
          actualY1 += yDisplacement;
          //if can draw at this point
          if (canMoveRectangle(actualX1,
                               actualY1,
                               actualX1+sRect.getWidth(),
                               actualY1+sRect.getHeight())) {
               //add the new coordinates
               sRect.setX(actualX1);
               sRect.setY(actualY1);
               //add to the annotation
               rectAnnotation.setSRect(sRect);
               //add the new annotation into display
               display.addAnnotation(rectAnnotation);
               Graphics g = display.getGraphics();
               display.paint(g);
               //get the new mouse coordinates
               xMouse = evt.getX();
               yMouse = evt.getY();
          }
     }
}
```

Quadro 20 – Código fonte do método `translateRectROI()` da classe `DisplayJAIEvents`

Inicialmente é removido retângulo atual da tela e calculada a diferença entre o cursor do mouse e o retângulo. Logo após é verificado se o retângulo está dentro dos limites do *display* (`DisplayJAIAnnotations`) através do método `canMoveRectangle()`. Se estiver, então é adicionada a nova coordenada para o retângulo e então ele é desenhado novamente na tela. O Quadro 21 exibe parte do método `paintMetricLines()` da classe `ResultEvents` que serve para extrair medidas provenientes do estrabismo.

```
/** Paints the metric line and calculate the distances between
* reflex and iris.
* @param irisAnnotation the iris annotation
* @param reflexAnnotation the reflex annotation
* @param isRightIris true is the right iris */
public void paintMetricLines(CircleAnnotation irisAnnotation,
                             CircleAnnotation reflexAnnotation,
                             boolean isRightIris){
     SLine rightMetric = new SLine(0,0,0,0);
     SLine leftMetric = new SLine(0,0,0,0);
     rightMetric.setColor(Color.GREEN);
     leftMetric.setColor(Color.GREEN);
     Graphics g;
     //recover the iris metrics
     int irisXc = irisAnnotation.getSCircle().getX();
     int irisYc = irisAnnotation.getSCircle().getY();
     float irisRadius = irisAnnotation.getSCircle().getRadius();
     //recover the reflex metrics
     int reflexXc = reflexAnnotation.getSCircle().getX();
     int reflexYc = reflexAnnotation.getSCircle().getY();
     float reflexRadius = reflexAnnotation.getSCircle().getRadius();
     float hyp = irisRadius;
     float co = Math.abs(reflexYc - irisYc);
     float ca = (float)Math.sqrt((hyp*hyp) - (co*co));
     //the distance between irisXc and reflexXc
     float d = Math.abs(irisXc - reflexXc);
     float dLeft = 0, dRight = 0;
     //if the reflex is at the left side (between I and IV quadrant)
     if(reflexXc >= irisXc){
          //the resultant metric of the reflex to the left corner
          dLeft = Math.abs(ca - d - reflexRadius);
          //the resultant metric of the reflex to the right corner
          dRight = Math.abs(ca + d - reflexRadius);
     }else{
          //the resultant metric of the reflex to the left corner
          dLeft = Math.abs(ca + d - reflexRadius);
          //the resultant metric of the reflex to the right corner
          dRight = Math.abs(ca - d - reflexRadius);
     }
...
```

Quadro 21 – Parte do código fonte do método para extrair as medidas do estrabismo

Para calcular as medidas entre o reflexo da córnea e o limbo no sentido nasal e temporal, foi utilizado como base o teorema de Pitágoras. O método `paintMetricLines()` inicialmente recupera as coordenadas e medidas da íris e do reflexo, previamente identificados. A hipotenusa (`hyp`) é o raio da íris (`irisRadius`) e o cateto oposto (`co`) é a distância modular entre a coordenada y do reflexo (`reflexYc`) e a coordenada y da íris (`irisYc`). Possuindo estes dois parâmetros, calcula-se o cateto adjacente (`ca`). Também é necessário saber a distância da extremidade do reflexo até a extremidade da íris. O passo pré-eliminar é calcular a distância modular entre a coordenada x da íris (`irisXc`) e a coordenada x do reflexo (`reflexXc`). Então é realizada a medida da extremidade do reflexo até o limbo para o sentido nasal e temporal (`dLeft` e `dRight`).

3.3.2 Operacionalidade da implementação

Esta seção tem por objetivo mostrar a operacionalidade da implementação em nível de usuário. Nas próximas seções serão abordadas todas as funcionalidades da ferramenta.

3.3.2.1 Abrindo uma foto para exame

Na tela principal da ferramenta, na aba *File*, o usuário tem a opção para abrir uma foto ou sair do programa. A Figura 24 demonstra esta situação.

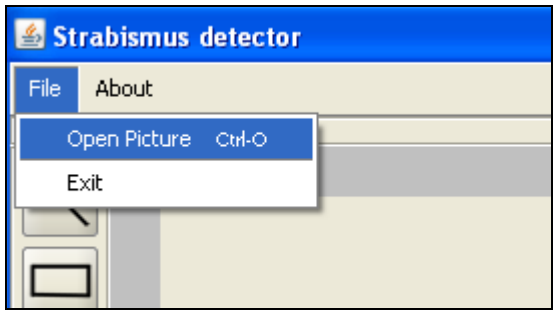


Figura 24 – Tela menu *File*

Ao selecionar uma imagem, a resolução mínima aceita pela ferramenta é de 800×600 *pixels*. Caso o usuário selecione uma imagem com resolução menor que a mínima exigida, uma mensagem é exibida conforme Figura 25.

Figura 25 – Tela de informação para resolução abaixo do mínimo exigido

Ao selecionar uma imagem, se o arquivo selecionado não corresponder a uma imagem, é exibida uma mensagem conforme a Figura 26.

Figura 26 – Tela de informação para arquivos que não sejam imagens

Se as condições referentes à Figura 25 e Figura 26 foram satisfeitas, então a ferramenta deverá exibir a foto do usuário, conforme Figura 27.

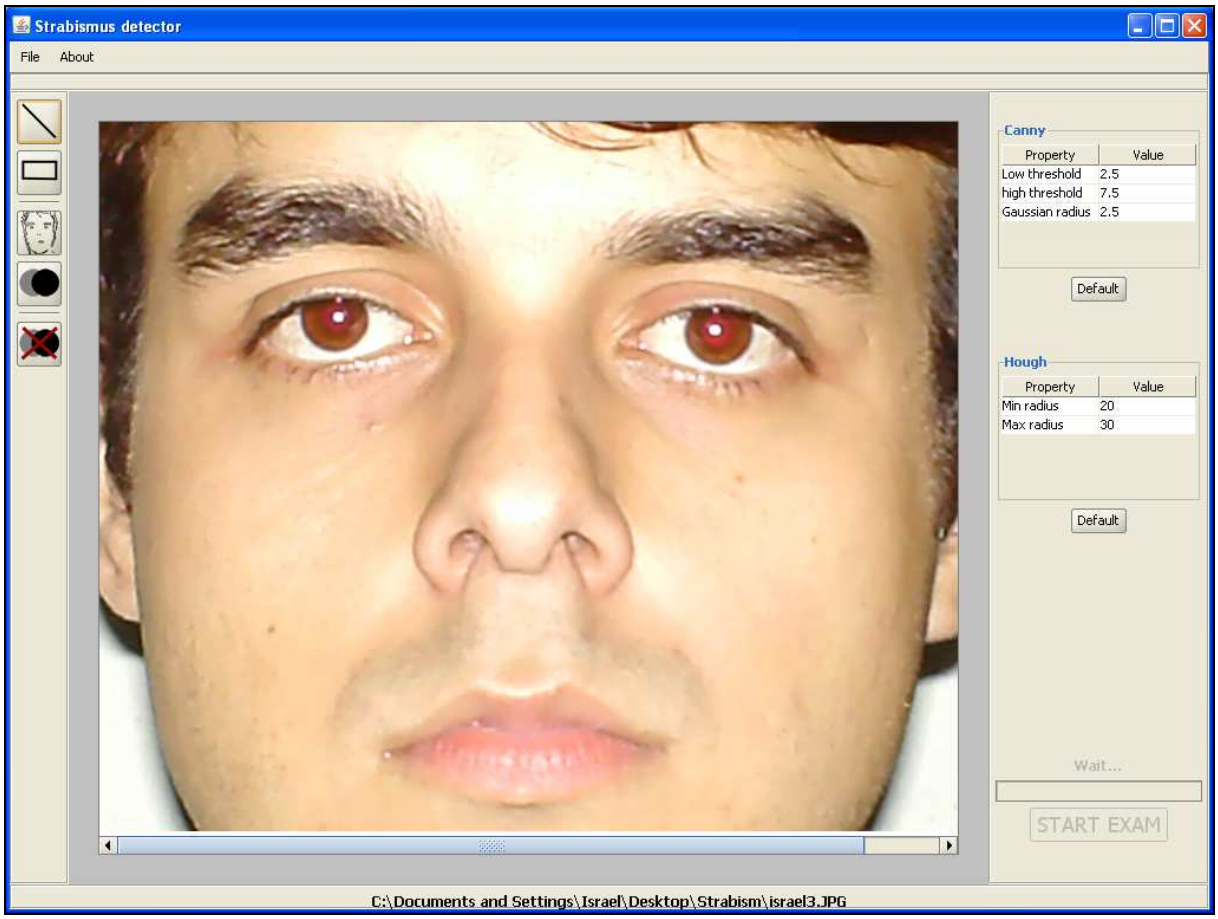


Figura 27 – Tela principal da ferramenta com a foto do usuário carregada

3.3.2.2 Recursos da ferramenta

A ferramenta disponibiliza dois recursos independentes para definição de regiões de interesse, sendo eles: *line tool* e *rectangle tool*. Por *default*, a linha (*line tool*) já vem centralizada na tela, visando separar os dois olhos. Caso seja necessário refinar mais ainda a região de interesse, o retângulo (*rectangle tool*) pode ser utilizado. O operador de Hough opera duas vezes na imagem, uma para achar a íris direita e outra para achar a íris esquerda. Algumas imagens podem ter muito ruído, possibilitando que o operador traga um resultado indesejado. Então, para eliminar as áreas de ruído, foram desenvolvidos estes dois recursos que estão situados na ferramenta conforme a Figura 28 e Figura 29 respectivamente.

Figura 28 – *Line tool*

Figura 29 – *Rectangle tool*

O usuário também pode informar os parâmetros dos algoritmos de Hough e Canny para processar a imagem. Conforme a Figura 30, podem ser informados respectivamente*: low threshold*, *high threshold* e *gaussian radius*. Estes valores são utilizados para fazer o processo de histerese e para aplicar a derivada gaussiana. A Figura 31 apresenta as propriedades do algoritmo de Hough. Podem ser informados o raio mínimo (*min radius*) e o raio máximo (*max radius*), em *pixels*, da íris. A ferramenta já vem com valores *default* e com uma opção para resgatar estes valores.

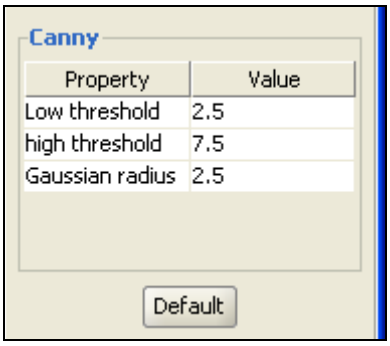


Figura 30 – *Canny properties*

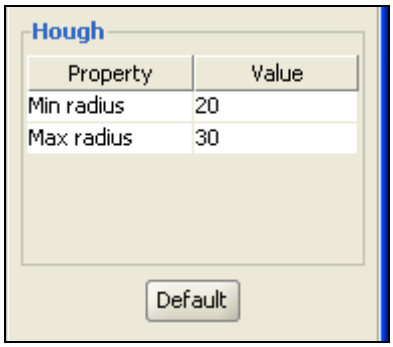


Figura 31 – *Hough properties*

A Figura 32 apresenta a opção para exibir o resultado do processamento da imagem com o operador de Canny. Quando clicado neste botão, a ferramenta executa o algoritmo pegando como parâmetros de filtragem os valores da Figura 30 e mostra na tela o resultado da execução sobre a imagem. Similarmente ocorre para a opção *run hough algorithm* exibida na Figura 33.

Figura 32 – *View Canny result*

Figura 33 – *Run Hough algorithm*

Para excluir o último resultado da execução do algoritmo de Hough, utiliza-se a opção *erase Hough result* exibida na Figura 34.

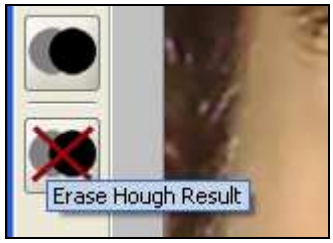


Figura 34 – *Erase Hough result*

3.3.2.3 Utilizando os recursos da ferramenta

Após a imagem estar carregada pelo sistema, pode-se delimitar as regiões de interesse. A *line tool* e *rectangle tool* são dinâmicas, ou seja, pode-se deslocar a linha e o retângulo, inclusive modificar interativamente a escala do retângulo. Vale lembrar que o retângulo não é obrigatório para a execução do operador de Hough, apenas a linha.

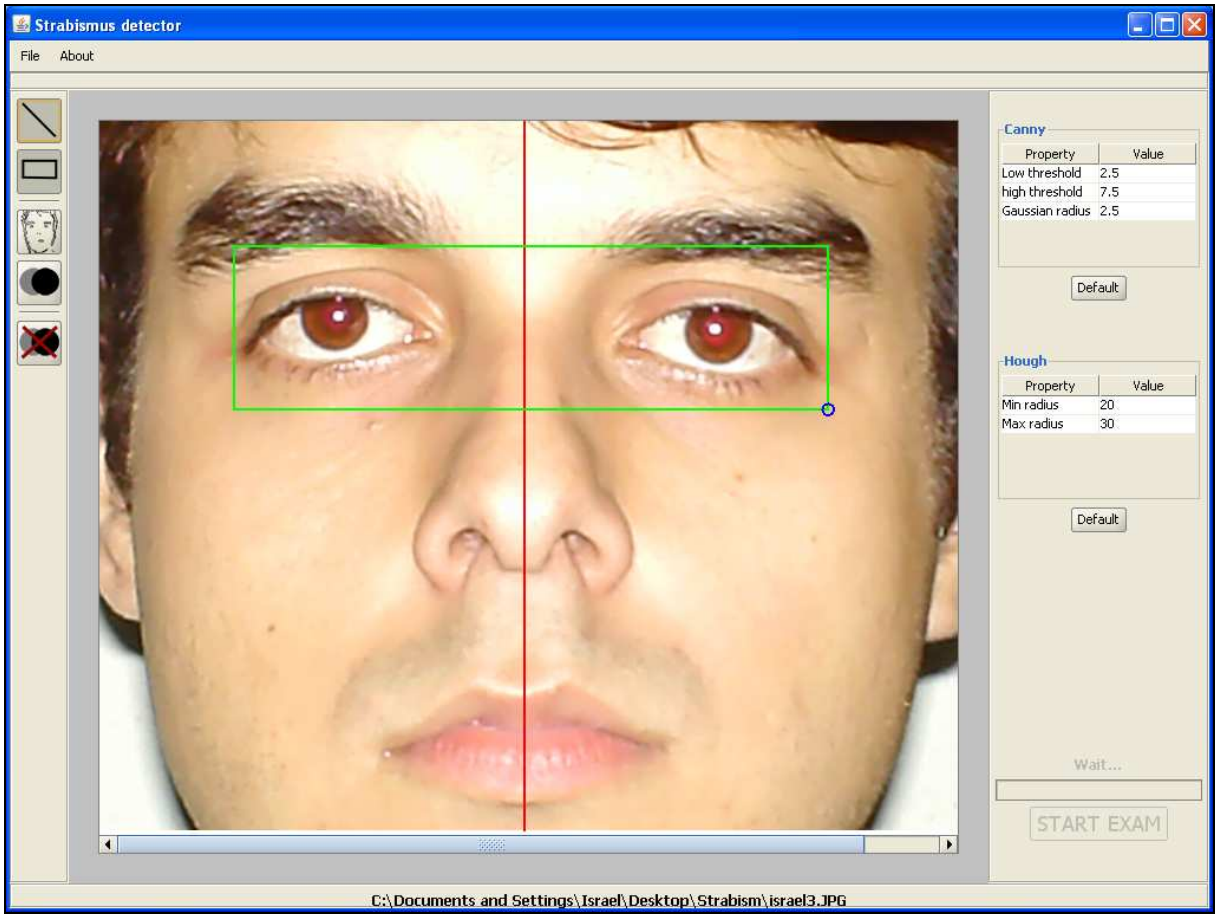


Figura 35 – Exemplo de uso de ROIs da ferramenta

Desta maneira, todo o resto que está fora dos limites do retângulo é ignorado, sendo processadas apenas as partes da imagem que estão dentro do retângulo. A linha serve para

dividir a face ao meio, possibilitando processar os olhos independentemente.

A Figura 36 mostra o resultado da execução do operador de Canny, utilizando como base os valores *default* passados como parâmetro. É muito importante o usuário poder visualizar esta etapa, pois se as bordas das íris não forem identificadas, o algoritmo de Hough não conseguirá encontrar o resultado esperado. Isto pode ocorrer dependendo da qualidade ou luminosidade da foto. Caso esta situação ocorra, o usuário tem livre arbítrio para alterar os parâmetros do filtro de Canny, podendo forçar uma detecção de bordas mais acurada.

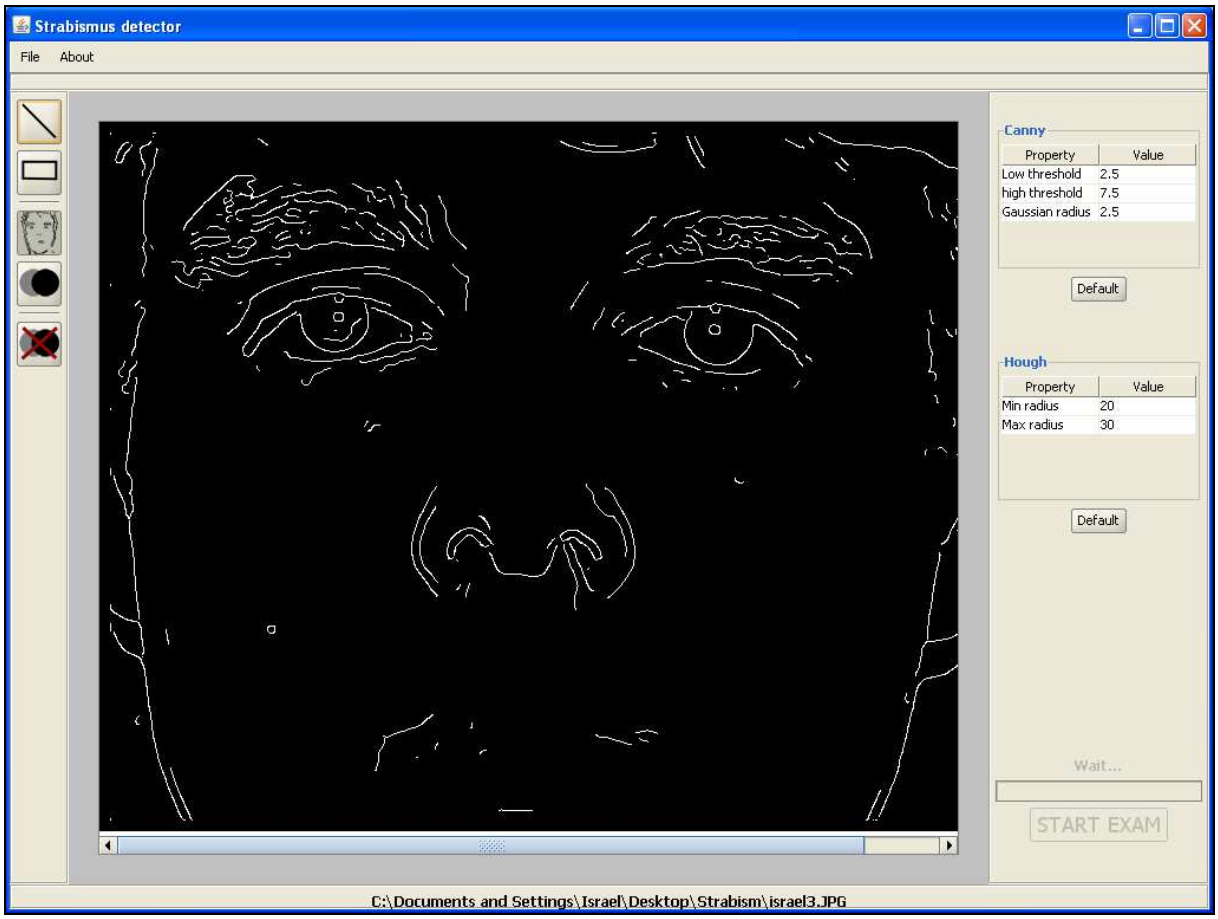


Figura 36 – Execução do operador de Canny na ferramenta

Após detectadas as bordas, pode-se iniciar o processo de identificação das íris do usuário. Ao clicar na opção *run Hough algorithm*, o sistema procurará pelas ROIs e executará o algoritmo de Hough. Analisando a Figura 37, percebe-se que as duas ROIs (*line tool* e *rectangle tool*) estão habilitadas. Visualmente pode-se perceber que as íris estão circundadas com um formato de “mira”. Isto significa que as íris foram identificadas com sucesso.

O intervalo raio é variável para cada íris, pois nem todas são do mesmo tamanho e também a distância da captura da foto é um fator muito importante para se determinar os limites da procura. O software foi testado com um padrão de aquisição de imagem semelhante à Figura 27, onde a foto precisa se iniciar um pouco acima do queixo da pessoa, até a testa.

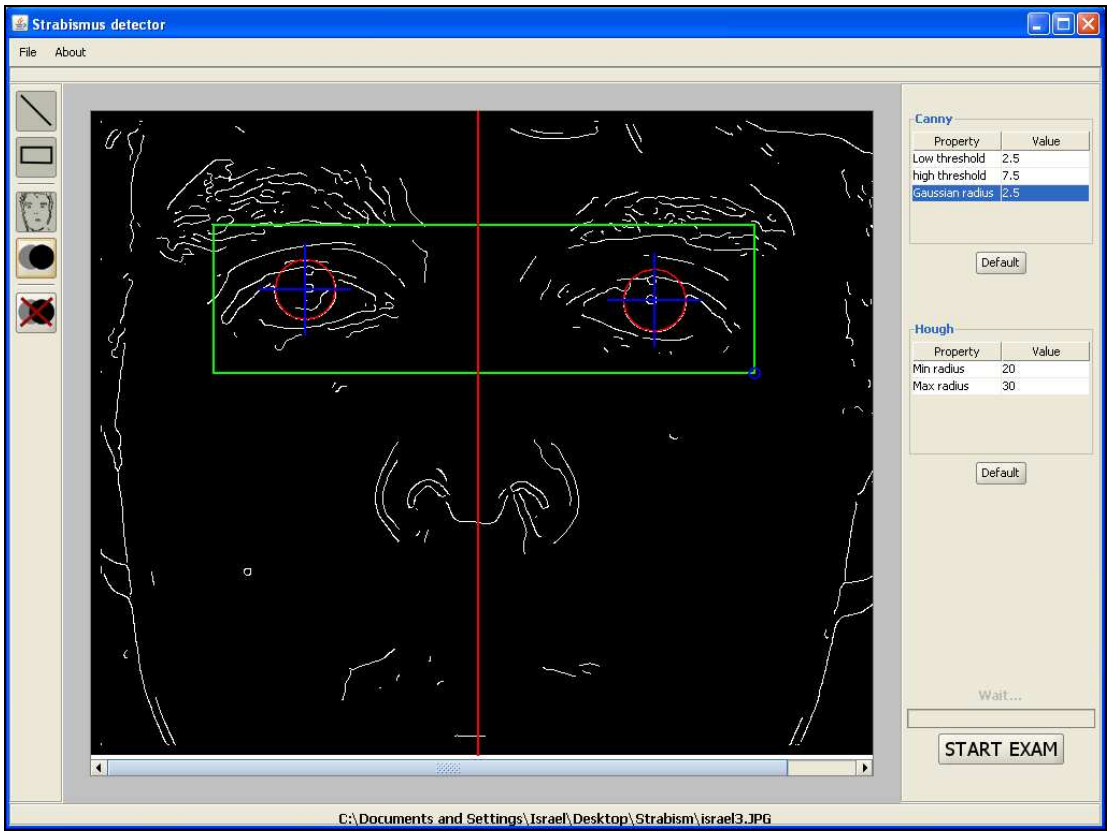


Figura 37 – Execução do algoritmo de Hough na ferramenta para detectar as íris

Para uma melhor visualização da detecção das íris, pode-se desabilitar as opções de detecção de bordas e ROIs. Esta visualização encontra-se na Figura 38.

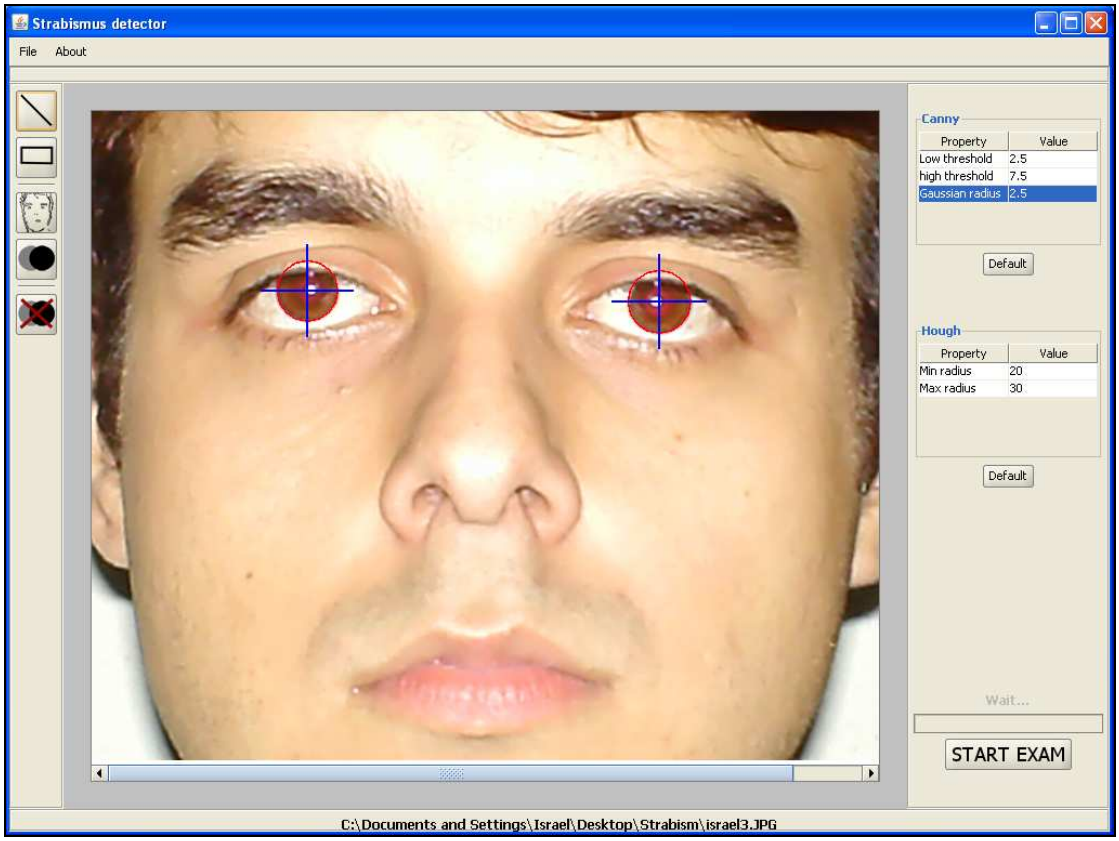


Figura 38 – Visualização das íris detectadas

Depois de identificar as íris, o botão *start exam* é habilitado. Ao clicar no botão, o sistema define uma ROI baseada no raio e ponto central da íris, aumenta a escala da imagem para 400×400 *pixels* e executa mais uma vez os algoritmos de Hough e Canny, desta vez utilizando parâmetros pré-determinados pela ferramenta para identificar o reflexo e a íris novamente. A Figura 39 exibe a tela de resultados do exame.

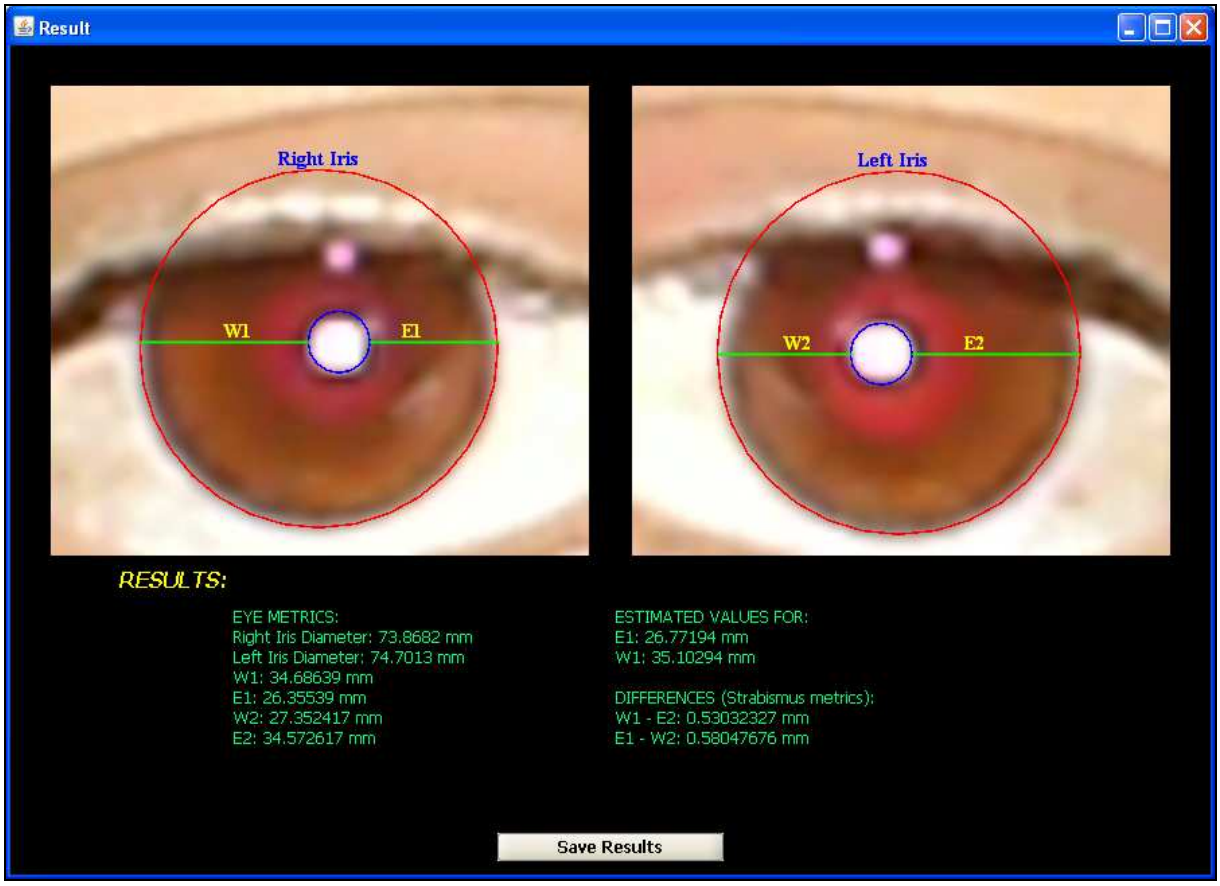


Figura 39 – Tela de resultados do exame

São listados os seguintes resultados: diâmetro da íris direita (***right íris diameter***), diâmetro da íris esquerda (***left íris diameter***), distância entre reflexo e limbo no sentido temporal para íris direita (**W1**), distância entre reflexo e limbo no sentido nasal para íris direita (**E1**), distância entre reflexo e limbo no sentido nasal para íris esquerda (**W2**), distância entre reflexo e limbo no sentido temporal para íris esquerda (**E2**). Todas as medidas estão sendo representadas em milímetros[12], porém não em escala real da íris. A escala, nesta etapa, foi aumentada para maior precisão na detecção das estruturas de interesse.

Como geralmente as íris não possuem exatamente o mesmo tamanho, foi calculada a diferença do diâmetro entre as íris e incrementada a metade desta diferença para o raio da menor íris. Como no exemplo da Figura 39 a íris direita tem um menor diâmetro, o ajuste é

[12] O software assume que um pixel equivale a 0.2777 mm.

realizado nela. Assim, têm-se os valores estimados: para **E1** e **W1**, caso a íris direita tivesse um maior diâmetro, o ajuste seria em relação a **W2** e **E2**.

Por fim, são listadas as medidas do estrabismo (*strabismus metrics*), onde a diferença entre $(W1-E2)$ e $(E1-W2)$ deve tender a zero para pacientes normais.

Caso o usuário desejar salvar os resultados, basta clicar em *save results*, selecionar uma pasta e a ferramenta gerará um arquivo no formato PDF contendo todas as medidas, inclusive as imagens das íris apresentadas na Figura 39.

3.4 RESULTADOS E DISCUSSÃO

Até o presente momento não se conhece nenhuma ferramenta, em nível de usuário doméstico, que busca os mesmos objetivos da ferramenta desenvolvida. Então, a grande dúvida inicial foi saber quais técnicas deveriam ser utilizadas para resolver o problema em um período de tempo relativamente curto.

Foram criados vários protótipos a fim de testar as técnicas estudadas contando com a ajuda da ferramenta ImageJ[13] (IMAGEJ, 2008). Em conjunto com os protótipos desenvolvidos, grande parte do processamento das imagens foi realizado pelo do ImageJ. Por exemplo: transformar uma imagem RGB para escala de cinza, detectar as bordas, achar as formas circulares, tudo isso através de *plugins*[14] que a ferramenta disponibiliza ou que são desenvolvidos por usuários da mesma, devido ao fato de esta ferramenta ser de código aberto.

Utilizando os protótipos, percebeu-se que o operador de Canny, em conjunto com a transformada circular de Hough foi o que melhor se adaptou para atingir os objetivos do desenvolvimento da ferramenta. O operador de Canny opera achando as bordas mais próximas do real, o que é muito importante para a ferramenta. A precisão nos cálculos é um fator importante para se poderem realizar exames com certo grau de confiança.

As imagens de teste foram obtidas através da câmera digital Mitsuca DS5028BR (MITSUCA, 2008) de 5 *mega pixels*. Para adquirir as imagens de teste, o zoom da câmera digital foi colocado no máximo (4x) com o flash ativado. Os testes realizados com esta máquina digital tiveram um acerto de 100% para identificar a íris e o reflexo no olho da

[13] ImageJ é uma ferramenta de procassamento de imagens baseada em Java com domínio público.

[14] Por exemplo, para detectar formas circulares, o plugin HoughCircles (PISTORI; COSTA, 2006) foi utilizado.

pessoa, também foi capaz de extrair com precisão as medidas propostas.

A Figura 39 mostra o resultado do exame feito pela ferramenta de uma pessoa que não possui estrabismo. Por outro lado, a Figura 40 simula, através do programa Photoshop (PHOTOSHOP, 2008), uma situação de estrabismo. O olho direito, em relação ao paciente, está desviado em sentido temporal (exotropia).

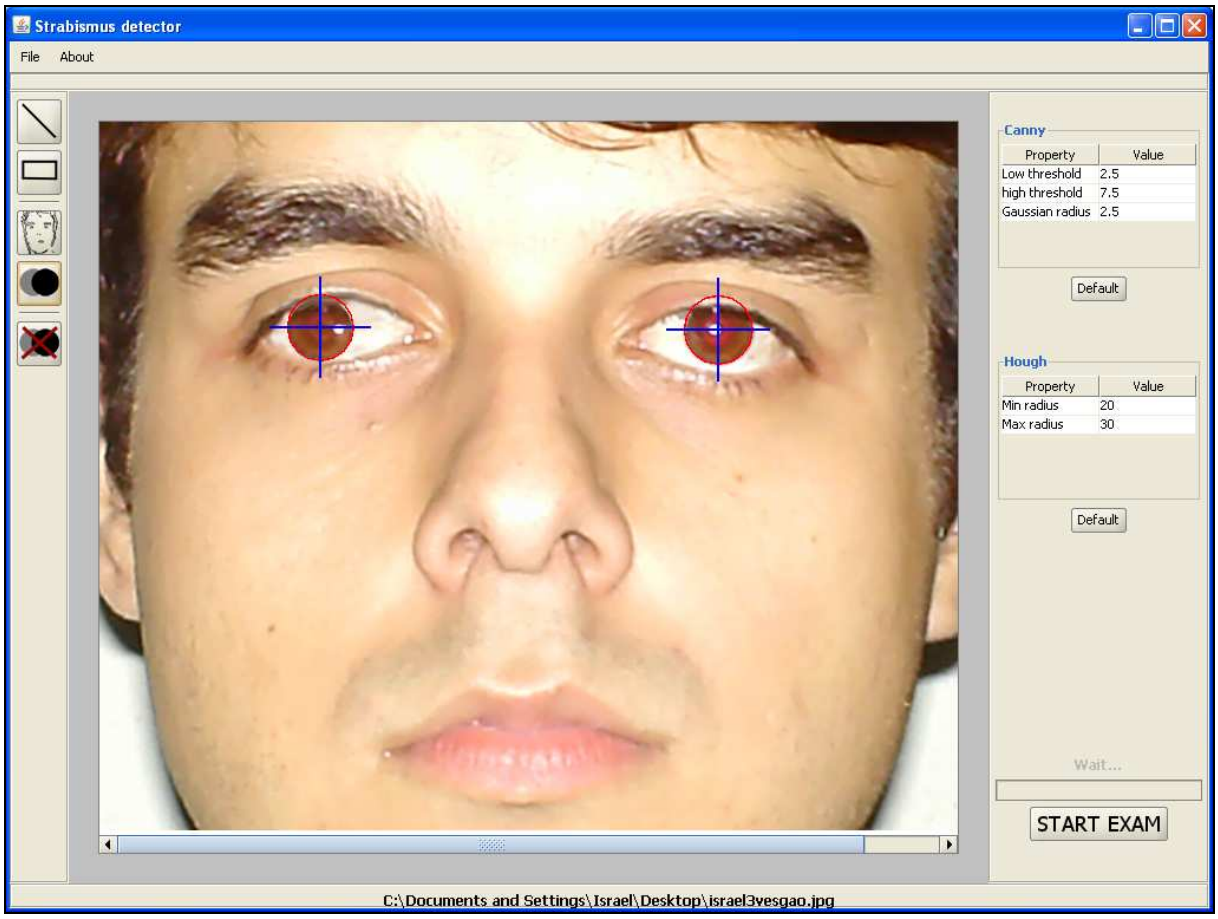


Figura 40 – Simulação de exotropia

O resultado do exame pode ser visto na Figura 41. Em *strabismus metrics* pode-se observar que o resultado de $W1-E2$ e $E1-W2$ deveria tender a zero. Entretanto $W1-E2=16{,}70062$ mm e $E1-W2=16{,}62338$ mm e estas são diferenças muito elevadas, evidenciando que a pessoa é uma forte candidata a possuir estrabismo.

Ainda analisando a Figura 41 pode-se perceber que a íris direita possui o reflexo mais deslocado em relação ao centro da íris se comparado com o olho esquerdo, então pode-se concluir que o olho direito é o olho deslocado. Para determinar que esta pessoa possui exotropia, basta comparar a medida de $E1$ em relação a $W2$, também $W1$ em relação a $E2$. Como $E1$ tem uma medida muito menor que $W2$ e $W1$ é muito maior que $E2$, sabe-se que o olho está deslocado temporalmente.

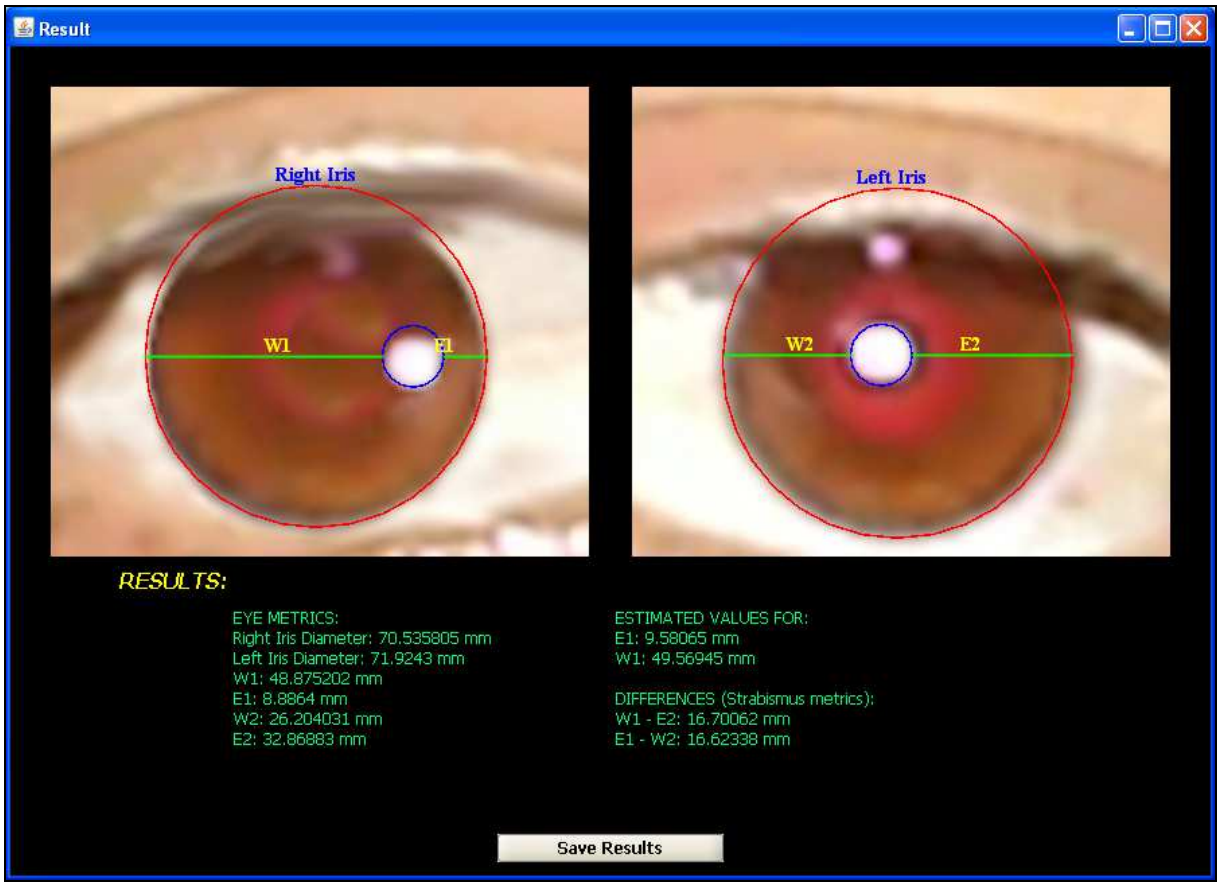


Figura 41 – Resultado da simulação de exotropia

Para a ferramenta exibir um diagnóstico informando o tipo de estrabismo (esotropia ou exotropia), seria necessário testá-la em vários pacientes estrábicos, ficando como sugestão para trabalhos futuros. Outra limitação do presente trabalho é o fato das medidas reais dos elementos avaliados não estarem sendo computadas. O diagnóstico correto necessita de medições mais precisas de distâncias e apesar das técnicas utilizadas terem reconhecido correta e eficientemente dos elementos anatômicos necessários, ainda há a necessidade de promover a calibração de câmera para cada foto, de modo a se poder correlacionar com precisão a medida de um *pixel* na imagem com a medida real em milímetros nos olhos do paciente. Uma maneira relativamente simples de obter esta calibração é solicitando ao paciente que, ao fazer a foto, coloque uma régua graduada em centímetros encostada na face, a uma distância da câmera igual à distância de seus olhos à câmera, conforme a Figura 42.

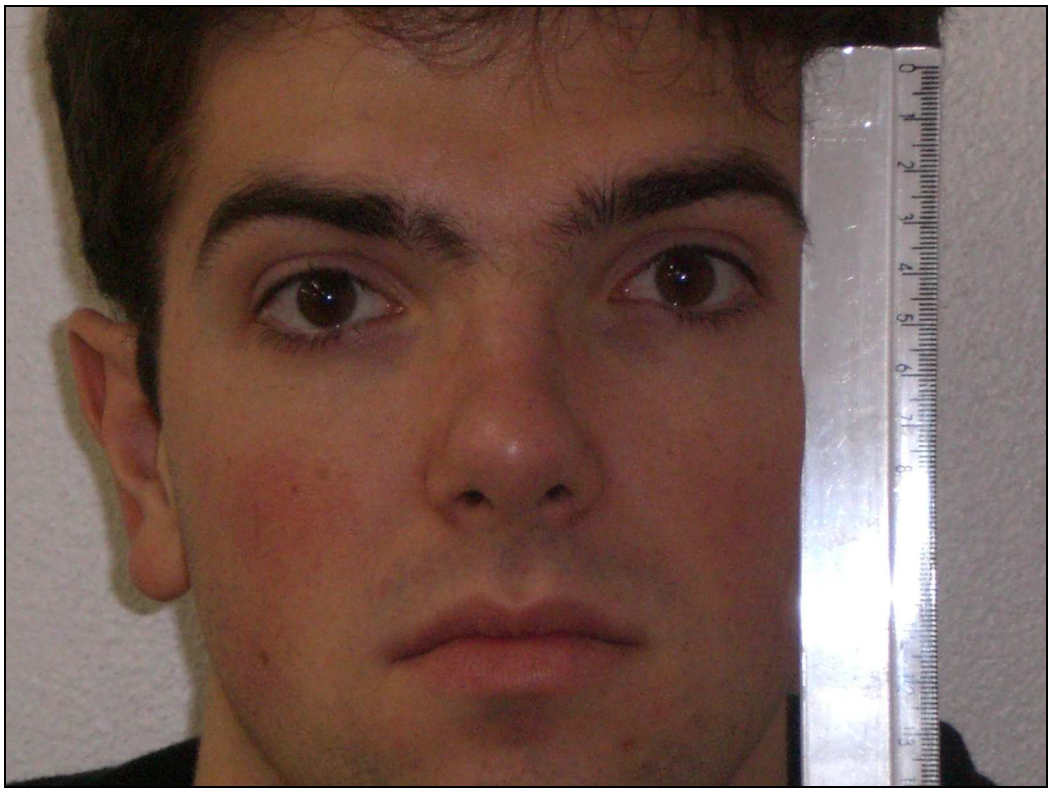


Figura 42 – Foto da face de uma pessoa com régua para calibração de medida

4 CONCLUSÕES

Depois de estudada a problemática sobre o estrabismo, decidiu-se desenvolver a ferramenta de uma maneira em que as pessoas possam utilizá-la sem dificuldades, assim prevenindo-as de uma maneira simples. Basta o usuário ter uma câmera digital com flash e um computador que ele pode extrair medidas do estrabismo em casa.

A implementação da ferramenta contempla a análise da imagem digital de uma maneira acurada. As opções para demarcar regiões de interesse na imagem (*line tool*) e (*rectangle tool*) são cruciais para se obter um resultado preciso, dependendo da qualidade da imagem. O algoritmo de Canny teve um bom resultado para identificar as bordas e o operador de Hough se mostrou útil e confiável para resolver o problema de identificar as íris e o reflexo ocasionado pelo *flash* da câmera digital. É possível obter com precisão o diâmetro das íris, através da imagem digital, bem como a distância da extremidade do reflexo até o limbo medial e temporal. Quanto às medidas, utilizou-se uma maior escala para realizá-las. Assim tem-se uma maior precisão ao identificar as bordas das íris, principalmente do reflexo na córnea ocular através dos métodos propostos.

Pelo fato de ser muito difícil conseguir fotos de pessoas que possuam estrabismo, a ferramenta não informa o tipo de estrabismo (exotropia ou esotropia), caso o usuário venha a ter esta doença. Para isso, seriam necessários testes em larga escala com pacientes estrábicos a fim de se validar estes casos.

As ferramentas utilizadas mostraram-se muito eficientes, principalmente o plugin Jigloo (JIGLOO, 2004) para o Eclipse (ECLIPSE, 2008). Com ele, pode-se “desenhar” a interface *swing* e obter um código legível como saída. A biblioteca iText (ITEXT, 2008) é de simples utilização e dispõe de vários recursos para criação de arquivos PDF. A ferramenta para especificação JUDE (JUDE, 2008) se mostrou muito estável, não apresentando erros de execução e também foi muito útil para a especificação do trabalho, atendendo a todas as necessidades do projeto. A linguagem de programação Java, incluindo suas extensões Java2D e JAI são muito bem documentadas, não havendo dificuldades ao se utilizar de algum recurso.

4.1 EXTENSÕES

Para futuras versões sugere-se que seja implementado um detector de face para a imagem digital a fim de se delimitar a área de busca das íris. Desta maneira, a ferramenta seria muito mais simples de ser utilizada por usuários que não tenham muita experiência no manuseio do computador porque as ferramentas de determinar ROIs (*line tool* e *rectangle tool*) dificilmente seriam requisitadas.

Especificar uma técnica para que o software consiga identificar o tamanho real da íris do paciente, isto ajudaria a definir a angulação do estrabismo conforme Hirschberg. Além da foto com a régua, pode-se fazer um gabarito que o usuário deve imprimir para que a ferramenta identifique automaticamente, por exemplo um círculo de diâmetro de 1 cm, assim calibrando a ferramenta.

Fazer a verificação se o reflexo da íris foi identificado corretamente, através de uma limiarização ou análise de histograma da área identificada.

Realizar testes em grande quantidade de pacientes estrábicos a fim de calibrar a ferramenta para diagnosticar casos de exotropia e esotropia.

REFERÊNCIAS BIBLIOGRÁFICAS

CENTRO BRASILEIRO DE ESTRABISMO. São Paulo, 2008. Disponível em: <http://www.cbe.org.br>. Acesso em: 13 maio 2008.

CONCI, A.; AZEVEDO, E.; LETA, F. R. **Computação gráfica**: teoria e prática. Rio de Janeiro: Campus, 2008.

CORRÊA, Z. M. S. **A sensação de profundidade (estereopsia) e a visão binocular**. [S.l.], 2006. Disponível em: <http://www.drgate.com.br/index.php?option=com_content&Itemid=67&task=view&id=151>. Acesso em: 15 set. 2007.

COSTA, L. F.; CESAR JR., R. M. **Shape analysis and classification**: theory and practice. Boca Raton: CRC Press, 2001.

ECLIPSE. [S.l.], 2008. Disponível em: <http://www.eclipse.org>. Acesso em: 27 maio 2008.

GOMES, J.; VELHO, L. **Fundamentos da computação gráfica**. Rio de Janeiro: IMPA, 2003.

IMAGEJ. [S.l.], 2008. Disponível em: <http://rsb.info.nih.gov/ij>. Acesso em: 03 maio 2008.

ITEXT. Gent, 2008. Disponível em: <http://www.lowagie.com/iText>. Acesso em: 27 maio 2008.

JAVA 2D API. Santa Clara, 2008. Disponível em: <http://java.sun.com/products/java-media/2D/index.jsp>. Acesso em: 14 maio 2008.

JAVA ADVANCED IMAGING API. Santa Clara, 2008. Disponível em: <http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs>. Acesso em: 18 maio 2008.

JAVA PLATFORM SE 6. Santa Clara, 2008. Disponível em: <http://java.sun.com/javase/6/docs/api>. Acesso em: 16 maio 2008.

JIGLOO. [S.l.], 2004. Disponível em: <http://www.cloudgarden.com/jigloo>. Acesso em: 25 maio 2008.

JUDE. Tokyo, 2008. Disponível em: <http://jude.change-vision.com/jude-web/index.html>. Acesso em: 26 maio 2008.

MCREYNOLDS, T.; BLYTHE, D. **Advanced graphics programming using OpenGL**. Amsterdam: Morgan Kaufmann Publishers, 2005.

MEDICINA GERIÁTRICA. [S. l.], 2008. Disponível em: <http://www.medicinageriatrica.com.br/2006/12/20/saude-geriatria/telemedicina-e-telesaude>. Acesso em: 13 maio 2008.

MITSUCA. [S.l.], 2008. Disponível em: <http://www.mitsuca.com.br/anahickmann>. Acesso em: 03 jun. 2008.

NIEMEYER, P.; PECK, J. **Exploring Java**. Sebastopol: O'Relly, 1996.

OZAWA, D. M. **Processamento e reconhecimento de imagens digitais da retina humana**. 2004. 46 f. Trabalho de Conclusão de Curso (Graduação em Ciência da Computação) – Centro de Ciências Exatas e Naturais, Universidade Estadual de Londrina, Londrina.

PACIORNIK, S. **Processamento digital de imagens**. [Rio de Janeiro], 2007. Disponível em: <http://www.dcmm.puc-rio.br/cursos/ipdi>. Acesso em: 19 set. 2007.

PEDSEYE. New York, 1999. Disponível em: <http://www.pedseye.com/Exotropia.htm>. Acesso em: 12 set. 2007.

PHOTOSHOP. [S.l.], 2008. Disponível em: <http://www.adobe.com/br/products/photoshop/photoshop>. Acesso em: 03 jun. 2008.

PISTORI, H; COSTA, E. R. HoughCircles. 2006. Disponível em: <http://rsb.info.nih.gov/ij/plugins/hough-circles.html>. Acesso em: 25 mar. 2008.

PISTORI, H.; PISTORI, P.; COSTA, E. R. Hough-circles: um módulo de detecção de circunferências para o ImageJ. ln: WSL, 2005, Campo Grande. **Anais eletrônicos...** Campo Grande: GPEC, 2005. Disponível em: <http://www.gpec.ucdb.br/pistori/publicacoes/pistori_wsl2005.pdf>. Acesso em: 26 maio 2008.

PRADO JR., C. A. **Biometria com enfoque em reconhecimento de íris**. 2005. 49 f. Trabalho de Conclusão de Curso (Graduação em Ciência da Computação) – Centro de Ciências Exatas e Naturais, Universidade Estadual de Londrina, Londrina.

PRICE, S. **Edges:** the Canny edge detector**.** [S.l.], 2004. Disponível em: <http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/MARBLE/low/edges/canny.htm>. Acesso em: 20 maio 2008.

SALIDO, F. **Assistência médica à distância.** [S.l.], 2008. Disponível em: <http://www.virtual.epm.br/material/tis/curr-med/temas/med5/med5t12000/tele/index.html>. Acesso em: 13 maio 2008.

SANTOS, R. Java advanced imaging API**:** a tutorial. **Rita,** [Porto Alegre], v. 11, n. 1, 2004. Disponível em: <http://www.inf.ufrgs.br/~revista/docs/rita11/rita_v11_n1_p93a124.pdf>. Acesso em: 22 maio 2008.

SOUZA-DIAS, C. R.; ALMEIDA, H. C. **Estrabismo**. São Paulo: Roca, 1993.

SUN DEVELOPER NETWORK. Santa Clara, 2008. Disponível em: <http://java.sun.com/products/java-media/2D/index.jsp>. Acesso em: 16 maio 2008.

UML. **Unified modeling language specification**: version 1.4. [S.l], 2002. Disponível em: <http://www.omg.org>. Acesso em: 28 maio. 2008.