Universidade Católica Dom Bosco
Centro de Ciências Exatas e Tecnológicas
Curso de Engenharia de Computação

Rastreamento da Posição dos Olhos para Detecção da Direção do Olhar

Emerson Galves Moretto

Prof. Orientador: Dr. Hemerson Pistori

Relatório Final submetido como um dos requisitos para a obtenção do grau de Engenheiro de Computação.

UCDB - Campo Grande - MS - Dezembro de 2004

Agradecimentos

Agradeço imensamente o meu orientador, o Professor Dr. Hemerson Pistori, pela paciência, pela contínua força de entusiasmo e principalmente pela ajuda nos momentos difíceis. O Professor foi o fator essencial para o desenvolvimento desse trabalho.
A minha tia, Conceição Butera, pela constante cobrança e confiança.
Ao pessoal do projeto SIGUS, pelas dicas e informações compartilhadas.
Ao corpo docente pelo ótimo trabalho realizado durante toda a minha graduação.
Ao meu irmão por usar o pentium 100 MHz para fazer seus trabalhos enquanto eu fazia meu projeto de graduação no outro computador melhor.
E aos meu amigos de sala, que foram e sempre serão companheiros e que jamais me negaram ajuda.

Resumo

O baixo custo dos dispositivos de captura de imagens, juntamente com o aumento significativo na capacidade de processamento dos computadores pessoais da atualidade, abrem espaço para o desenvolvimento de novos tipos de interação homem-máquina. Este relatório apresenta um estudo de interação homem-máquina, através da visão humana, que utiliza técnicas de Visão Computacional e um dispositivo de captura digital (*Webcam*). Essa abordagem consiste, basicamente, na estimativa da direção do olhar a um objeto, no caso, uma região da tela de um monitor através de um dispositivo de captura digital. O processo de rastreamento da posição dos olhos é uma parte fundamental para estimar a direção do olhar. Técnicas de Visão Computacional, como a transformada de Hough, são utilizadas para rastrear a localização dos olhos e são abordadas por este estudo, que visa também evitar o uso de técnicas intrusivas e expansivas.

Abstract

The low cost of the devices of capture of images together with the significant increase in the capacity of processing of the personal computers of the present time, they open space for the development of new methods of interaction man-machine. This report presents a study of interaction man-machine, through the human vision, that uses techniques of Computer Vision and a device of digital capture (Webcam). That approach consists in the estimate of the direction of the gaze to an object, in the case, an area of the screen of a monitor through a device of digital capture. The process of tracking of the position of the eyes is a fundamental part to detect the direction of the gaze. Techniques of Computer Vision, as transformed it of Hough, are used to trace the location of the eyes and they are approached by this study, that also seeks to avoid the use of intrusive and expansible techniques.

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

Um dos principais objetivos das pesquisas de interação homem-máquina atualmente é desenvolver novos meios de interação que possam atender a uma maior quantidade de usuários. O surgimento de alguns desses novos meios vem da necessidade de atender a segmentos da sociedade que apresentam determinadas deficiências físicas. O baixo custo de dispositivos de captura de imagens juntamente com o aumento significativo da capacidade de processamento dos computadores pessoais [Pis03] também são fatores que permitem desenvolver novos meios de interação homem-máquina. Em torno dessas e de outras justificativas, novos tipos de interação vêm emergindo, como a detecção da direção do olhar.

A substituição dos tradicionais teclados e mouses por sinais visuais capturados por um simples dispositivo de captura digital de baixo custo, oferece um conjunto de aplicações que podem viabilizar e facilitar a utilização de novos meios de interação homem-máquina. Editores de texto, jogos, navegadores para Internet, clientes de e-mail e outros aplicativos acionados através de movimentos do globo ocular, por exemplo, poderiam ser utilizados por pessoas tetraplégicas, realizando a inclusão digital.

Hoje em dia existem várias ferramentas disponíveis de interação através da visão humana, como óculos especiais (figura 1.1), eletro-oculograma[1] (figura 1.2), sistemas com emissores e receptores infra-vermelho (figura 1.3), dentre outros que permitem detectar a direção do olhar. Esses dispositivos, no entanto, possuem algumas limitações, como o alto custo e falta de disponibilidade. Esses aparelhos são chamados de intrusivos, ou seja, são aparelhos que requerem um contato direto com o usuário ou que o usuário "vista"o

[1]Aparelho que registra o movimento dos olhos através de eletrodos fixados em determinadas regiões da cabeça

equipamento. Devido a essa característica, provocam certo desconforto ao usuário, pois há um contato direto com o olho e/ou com a pele.

Figura 1.1: Óculos especias

Figura 1.2: Eletro-oculograma

Figura 1.3: Sistema com emissores infra-vermelho

A interação homem-máquina através da visão humana se destaca por ser um meio de interação com um alto poder de comunicação, tanto na velocidade quanto na quantidade de informação que pode ser obtida. Estudos indicam que objetos apresentados em uma tela de computador podem ser apontados com maior facilidade e comodidade com os olhos do que com um mouse [Jac91]. A direção do olhar e a movimentação da cabeça são focos de pesquisas nesta área pela sua vasta aplicabilidade, como por exemplo, no controle de cadeira de rodas[RB02].

A detecção da direção do olhar pode ser aplicada na Ciência Cognitiva [LQX98], que é a ciência dos fenômenos constitutivos dos aparelhos e os comportamentos psicobiológicos e das interações entre estes aparelhos e o comportamento humano. Essa ciência descreve, explica, e, eventualmente, simula as principais disposições e capacidades da cognição humana; a linguagem, o raciocínio, a percepção, a coordenação motora e planificação[2]. Suas aplicações consistem em programas que analisam os modos como o ser humano olha, pensa, fala, compreende, aprende, procurando elaborar dados para uma investigação de como o ser humano interpreta os processos mentais.

Hoje em dia existem vários caminhos para estimar a direção do olhar. Nenhuma dessas técnicas atendem de forma ideal, ou seja, ainda não satisfazem todos requisitos de usabilidade definidos por Glenstrup e Engell-Nielsen [AJG95] citados abaixo:

- Oferecer um campo de visão que contenha a cabeça do usuário por completo;
- Não ter contato com o usuário;
- Possuir exatidão de cerca de 1% de tolerância de erros, limitado pelos efeito acumulativos da não-linearidade, distorção, ruído, atraso e outras fontes de erros;
- Oferecer uma boa taxa de atualização (cerca de 100Hz seria uma boa taxa);
- Ter todo o processo em tempo-real, para permitir movimentos fisiológicos;
- Ser fácilmente usável por diferentes usuários;
- Ser tolerante às rotações angulares da cabeça do usuário;

Autores denotam que esses requisitos são desejáveis, no entanto, devemos notar que eles não são todos os pré-requisitos para uma interface de detecção da direção do olhar funcional.

Este trabalho propõe-se a estudar e implementar técnicas para o rastreamento da posição dos olhos aplicadas à detecção da direção do olhar com o objetivo de evitar técnicas que possam vir a causar falta de portabilidade e

[2]Retirado do sítio http://an.locaweb.com.br/Webindependente/CienciaCognitiva/

aumento de custo. O uso de um dispositivo de captura digital e um computador com um poder de processamento razoável [AH00] (pentium 200mhz) já nos possibilita selecionar um conjunto de características que nos permita estimar a direção do olhar.

Um outro objetivo deste trabalho é desenvolver protótipos que possam ser utilizados pelo projeto Plataforma SIGUS. O Sistema de apoio a Implementação de programas com interface GUiada por Sinais visuais (SIGUS) é um ambiente computacional com programas-fonte livres, para facilitar a implementação de sistemas com interface guiada por sinais visuais, integrando e complementando as bibliotecas de processamento digital de sinais, *ImageJ*[3] e de aprendizagem computacional. Os protótipos desenvolvidos neste trabalho consistem em aplicações, mais especificamente módulos, para a Plataforma SIGUS que utilizem a localização dos olhos para detectar a direção do olhar.

Desenvolver material e informações suficientes para a continuação dessa pesquisa também fazem parte dos objetivos deste trabalho. Todo o código desenvolvido esteve sempre aberto e a disposição de outros pesquisadores o tempo todo na página do trabalho[4]. A colaboração entre os membros do Projeto SIGUS esteve sempre presente durante a pesquisa deste trabalho.

O texto deste trabalho está divido em seis grandes partes: revisão bibliográfica, plataforma SIGUS, Transfomada de Hough, protótipos desenvolvidos, resultados e experimentos e considerações finais. A revisão bibliográfica (cap. 2) oferece um levantamento da literatura na área destacando os pontos mais relevantes dos trabalhos já desenvolvidos. No capítulo sobre a plaforma SIGUS (cap. 3) é abordado a ligação deste trabalho com a plataforma e seus conceitos. O capítulo da Transformada de Hough (cap. 5) apresenta a definição, teoria e um exemplo da transformada ilustrado através de figuras e um algoritmo. Os três protótipos desenvolvidos estão apresentados no capítulo 6. Nele é feito uma abordagem e uma ilustração do funcionamento para cada protótipo. Os códigos-fonte dos protótipos estão nos anexos deste trabalho. O capítulo 7 apresenta os resultados de testes realizados através dos protótipos desenvolvidos. Neste capítulo é descrito o ambiente de desenvolvimento. E por fim, o último capítulo é dedicado às conclusões e sugestões para trabalhos futuros.

[3]Software livre para processamento digital de sinais disponível em http://rsb.info.nih.gov/ij/

[4]Página do Trabalho: http://www.ec.ucdb.br/~ emoretto

Capítulo 2

Revisão Bibliográfica

Este capítulo faz um levantamento das diferentes técnicas atualmente disponíveis para estimar a direção do olhar e destaca as principais características dos trabalhos mais relevantes.

A direção do olhar pode ser estimada tanto pela observação dos movimentos da cabeça, quanto pelo movimento do globo ocular e da íris. Estas estimativas podem ser obtidas por técnicas geométricas([KNK99],[CHM02]) ou algébricas([CC95],[ZZ02b],[ZZ02a],[SA03]) de inferência do foco do olhar a partir de características específicas da face projetada na imagem em 2D, aprendigem computacional ([RS97],[LQX98],[QJ02]), ou adquiridas a partir de eletro-oculograma[RB02].

Vários autores estão propondo novas técnicas para detectar a direção do olhar, e em sua maioria utilizam meios classificados como intrusivos ou expansivos[1] para facilitar a localização do olho. Essas técnicas intrusivas geralmente possuem uma taxa de acerto da localização dos olhos maior que sistemas não intrusivos e um menor custo de processamento, já que a localização do olho é facilitada. No entanto, possui várias desvantagens. O alto custo e a falta de acessibilidade são as princincipais desvantagens dessas técnicas. A figura 2.1 apresenta um hardware de um meio expansivo utilizando LEDs emissores infra-vermelho e uma câmera com capacidade de capturar esses sinais.

Com o uso de meios intrusivos, o sistema perde acessibilidade, pois não são facilmente obtidos pelos usuários, a menos que seja embutido ao monitor ou tela de cristal líquido e essa combinação seja vendida como um pro-

[1]Meios que utilizam técnicas para induzir e facilitar a detecção da pupila, como por exemplo os emissores e receptores de infra-vermelho que não tenham contato direto com o usuário.

duto. Existe, por exemplo, uma empresa chamada Tobii[2], que produz telas de cristal líquido com uma câmera e com emissores e receptores de sinais infra-vermelho embutidos, conforme na figura 2.1, ao custo de aproximadamente R$ 55.000[3]. Esses produtos tornam-se pouco acessíveis devido seu alto custo. Recentemente, a queda nos custos de dispositivos de captura de imagens, como as populares *WebCams*, impulsionou a busca de sistemas não-intrusivos de detecção da direção do olhar [Pis03].

Figura 2.1: Tela de cristal líquido com rastreador da direção do olhar

Alguns sistemas para detectar a direção do olhar utilizam um processo prévio de ajuste de valores para cada usuário que utilizar o sistema. Estes valores consistem em características de diferentes usuários, como por exemplo, a distância do usuário em relação a câmera. Sistemas de detecção da direção do olhar podem também ser classificados quanto a necessidade, ou não, deste processo de ajuste, chamado calibração. Morimoto [MRMM03] defende este processo como essencial para uma boa precisão na direção do olhar, já Haro [AH00], argumenta que é possível evitar essa parte do processo através de técnicas adaptativas.

Para poder comparar os resultados, os autores utilizam a medida da precisão angular obtida pelo seu sistema. A precisão angular é a forma de medir a taxa de acerto da direção do olhar, sendo a variação angular em relação ao ponto exato ao ponto inferido pelo sistema. Para facilitar a visualização da precisão angular, a figura 2.2 ilustra a representação geométrica.

[2]http://www.tobii.se/

[3]Em Janeiro de 2004 no sítio da empresa)

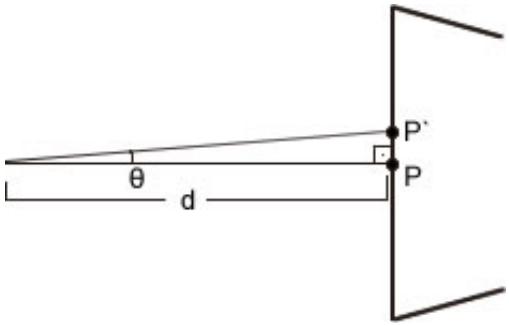

Figura 2.2: Precisão Angular

Em que θ é o ângulo da precisão angular, P é o ponto observado, P‘ é o ponto inferido pelo sistema e d é a distância do usuário em relação a tela do computador. Para obter essa variação em centímetros, basta utilizarmos a fórmula 2.1.

$$\tan(\theta) = \frac{e}{d} \tag{2.1}$$

Nesta fórmula e é o erro em centímetros. Por exemplo, uma precisão angular de 2 graus significa uma variação de até 1,39 centímetros com o usuário posicionado a 40 centímetros de distância da tela.

Os trabalhos correlatos desenvolvidos até então possuem muitas divergências em relação às técnicas utilizadas. Essas divergências se devem principalmente à característica inovadora da área em questão. Como forma de organizar os trabalhos correlatos, as seções abaixo apresentam os trabalhos mais influentes na área, dividindo-os pela característica do uso de meios intrusivos ou não. Dentre os não-intrusivos, há uma nova subdivisão, esta para separar técnicas baseadas em aprendizagem automática das técnicas baseadas em modelos algébricos.

2.1 Técnicas Intrusivas ou Expansivas

Morimoto [CHM02] utiliza em seu estudo, um meio expansivo de estimar a direção do olhar através de LEDs emissores de infra-vermelho e com uma câmera com capacidade de receber esses sinais (figura 2.1). Esse meio facilita estimar a direção do olhar, pois o sinal emitido pelos LEDs é refletido pela retina através do efeito “olho vermelho”, que é comum em fotografias com

flash. A câmera captura o quadro, que apresenta a região da retina destacada pela reflexão conforme na figura 2.4.

Figura 2.3: Sistema hardware que auxilia na detecção do olhar

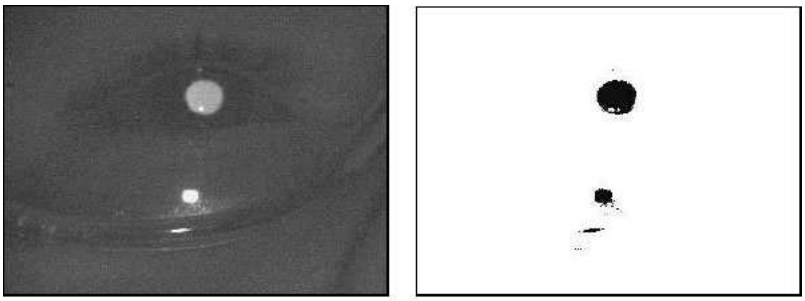

Figura 2.4: (a) reflexão do sinal infra-vermelho da pupila. (b) extração apenas da localização da pupila.

Qiang Ji e Zhiwei Zhu [QJ02] usam uma rede neural especial (GRNN - *Generalized Regression Neural Networks*)[4] que recebe parâmetros resultantes de algoritmos que encontram o reflexo dos emissores infra-vermelho através da pupila. Dentre esses parâmetros, está a coordenada do centro da pupila, essencial na detecção da direção. Dessa forma o sistema resultante torna-se mais robusto em relação ao seu uso por diferentes usuários. Os autores relatam ter obtido uma taxa de 85% de acerto para pessoas que não fizeram parte do treinamento da rede e 90% de acerto em usuários que participaram do treinamento.

[4]Rede Neural que utiliza funções probabilisticas de estimação de valores

2.2 Técnicas não Intrusivas

2.2.1 Técnicas Baseadas em Aprendizagem Automática

Stiefelhagem [RS97] utiliza em seu trabalho a imagem da região dos olhos como parâmetro de treinamento de uma rede neural artificial e relata ter obtido uma precisão angular de 1.3 a 1.9 graus. Em seus experimentos foram utilizadas 4000 imagens da região dos olhos de pessoas acompanhando o cursor que se deslocava na tela de um monitor. Essas imagens contendo apenas a região dos olhos e a posição referente do cursor, alimentaram uma rede neural artificial do tipo *feed-forward* com três camadas, utilizando a técnica de *backpropagation* para treinamento. Embora precisões de até 0.38 graus possam ser obtidas usando outros métodos [KHT02], as técnicas que utilizam redes neurais, além de não dependerem de dispositivos especiais, podem ser utilizadas em diversos tipos de ambiente e iluminação, o que em geral não ocorre com as outras técnicas. O problema com o enfoque baseado em redes neurais "tradicionais"é a característica de que o aprendizado é realizado apenas inicialmente, não sendo incremental, o que dificulta a correção de erros no modelo aprendido durante a fase de utilização do sistema.

Rikert [TR98] em sua abordagem utiliza um modelo morfológico da face para encontrar a região de contorno dos olhos. Este modelo é baseado na abordagem de modelos morfológicos multi-dimensionais de Jones [MJJ98]. Depois de encontrada, a região ao redor dos olhos é enviada como parâmetro de uma rede neural artificial juntamente com alguns outros valores, como a estimativa da direção da cabeça do usuário. O cálculo resultante desta rede neural estima a direção do olhar. Esta abordagem, de acordo com o autor, independe de diferenças entre os usuários pela capacidade de generalização da rede neural. Foram usados 287 exemplos para treinamento da rede neural, como resultado, foi obtido uma taxa de acerto de 81% utilizando o algoritmo de apredizagem *Backpropagation*.

Pistori [Pis03] apresenta em seu estudo técnicas de visão computacional que resultam em parâmetros para uma árvore de decisão adaptativa. Inicialmente, ocorre um processamento afim de encontrar uma região que compreende os olhos do usuário. A escolha dessa região baseia-se em 4 heurísticas:

1. As bordas existentes nas regiões imediatamente superiores (testa) e inferiores (maçãs da face) são bem mais tênues que as dos olhos e somem quase que completamente durante o pré-processamento;

2. Existe uma borda vertical facilmente detectável entre a íris e esclera

(figura 2.5);

Figura 2.5: Olho humano

3. A distância entre os dois olhos, quando vistos de frente, pode ser limitada inferior e superiormente;

4. Os olhos mantém-se razoavelmente alinhados em relação ao eixo horizontal.

A partir da região determinada, são extraídas características como a massa (total de pixels), centro de massa e variância em relação ao centro da região dos olhos. Esses parâmetros e a coordenada relativa da direção do olhar na tela são usados no treinamento incremental da árvore de decisão adaptativa.

2.2.2 Técnicas Baseadas em Modelos

Colombo [CC95] realiza uma abordagem em tempo real combinando movimentos da cabeça e a posição da pupila para determinar a direção do olhar. Sua técnica é aplicada em um museu virtual, onde o usuário navega pelo ambiente apenas com o uso de movimentos da cabeça e do olho. O movimento da cabeça e ações do olho, como o ato de piscar, são usados na navegação no ambiente. O autor utiliza algoritmos baseados em variações de contraste, como por exemplo, na busca pelo contorno dos olhos que servem de ponto de fixação para a movimentação da cabeça. Na deteção da posição da pupila, Colombo utiliza um modelo circular ideal (figura 2.6) de pupila e compara através de uma busca, o número máximo de correlações na imagem. A abordagem feita pelo autor não utiliza nenhum meio intrusivo ou expansivo, e seus agoritmos possuem um baixo custo de processamento, porém não permite movimentos bruscos da cabeça do usuário, pois suas técnicas de busca utilizam valores de quadros anteriores.

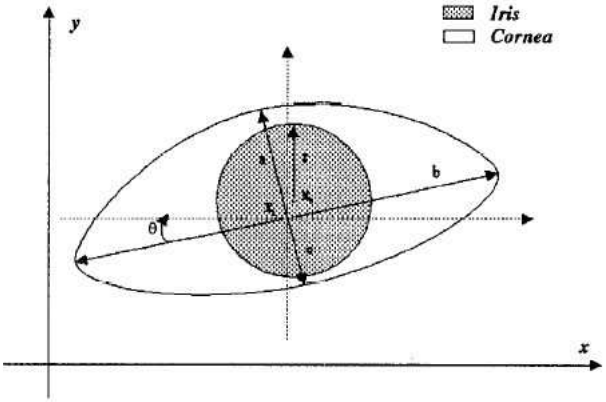

Figura 2.6: Geometria do olho

Wang e Sung [JGW03] modelam a direção do olhar como sendo a direção da normal ao plano de suporte de cada íris. Aproximando a forma da íris através de circunferências e assumindo raios, distâncias focais e fatores de escala conhecidos, é possível estimar a normal em questão a partir da projeção, elíptica, destes dois círculos. A manipulação algébrica deste problema oferece, no entanto, duas soluções para cada íris. Wang e Sung propõe a escolha das normais mais próximas entre si para resolver este impasse, e apresentam diversos resultados empíricos que sustentam a utilidade desta heurística. A detecção das duas elipses que correspondem ao contorno da íris na imagem projetada não é uma tarefa trivial, principalmente se for executada em tempo real. A estratégia de Wang e Sung exige uma alta capacidade de processamento, o que dificulta a construção de soluções de baixo custo.

Kyung-Nam [KNK99] propôs uma técnica de detecção da direção do olhar baseada na modelagem do globo ocular. Essa técnica simplifica o método de calibração, requerendo apenas dois pontos da direção do olhar na tela para realizar a calibração necessária. No entanto, este método requer certa estabilidade de movimento da cabeça do usuário, caso contrário, não apresenta bons resultados. Pesquisadores da IBM[5] também estudam a possibilidade da eliminação da calibração, esses estudos se baseiam em métodos que utilizam duas câmeras e exploram a geometria do olho. Outros trabalhos recentes como o de Zhu [ZZ02b], apresentam métodos de calibração "embutido"no processamento, mais especificamente, uma rede neural artificial infere um mapeamento de valores possíveis da direção do olhar.

[5]Informações retiradas do sítio: http://www.almaden.ibm.com/cs/blueeyes

McLaughlin [McL98] propõe em seu trabalho uma melhoria para o algoritmo original da Transformada de Hough, que consiste em utilizar um sistema de 3 equações lineares na descrição de elipses, e uma varredura, por amostragem informada, sobre os pontos que constituem as bordas da imagem original. Esta nova técnica, é denominada Transformada de Hough Aleatorizada (*Randomized Hough Transform*). O uso dessa nova técnica, além de obter uma boa taxa de acerto, apresenta um custo de processamento bem menor, viabilizando o seu uso em aplicações em tempo-real.

Capítulo 3

Plataforma SIGUS

A plataforma SIGUS é um ambiente de apoio ao desenvolvimento de aplicações com interfaces guiadas total ou parcialmente por sinais visuais. Esse projeto serve como base para o desenvolvimento de aplicações para a inclusão digital de pessoas com necessidades especiais. É coordenado pelo Prof. Dr. Hemerson Pistori e conta com vários professores e alunos colaborando para o desenvolvimento do mesmo.

O projeto tem como objetivo desenvolver um ambiente computacional com programas-fonte livres, para facilitar a implementação de sistemas com interfaces guiadas por sinais visuais, integrando e complementando as bibliotecas de processamento digital de sinais, ImageJ, de aprendizagem computacional, Weka e de dispositivos adaptativos, AdapTools.

Outros objetivos mais especifícos como, implementar um protótipo de um editor de textos para a Língua Brasileira de Sinais (LIBRAS) e implementar um protótipo de uma interface para um controlador de cadeira de rodas guiado pela direção do olhar fazem parte do projeto SIGUS.

O atual trabalho situa-se como um tópico de pesquisa da Plataforma SIGUS na questão da comunicação através de sinais visuais baseada nos movimentos dos olhos e segue as mesmas diretrizes, como por exemplo, desenvolvimento de programas-fonte livres. Outras diretrizes não menos importantes podem ser citadas:

- Desenvolvimento e pesquisa compartilhados com a equipe da platatorma;
- Pesquisa, desenvolvimento e produção livres;

- Portabilidade dos protótipos e aplicativos.

Para viabilizar o desenvolvimento da plataforma SIGUS, alguns pacotes livres já existentes, como o ImageJ, para processamento digital de imagens, o WEKA, para aprendizagem de máquina, o JMF, para manipulação de câmeras digitais e o AdapTools, para tecnologia adaptativa, serão utilizados. Todos esses pacotes são constituídos de programas na linguagem Java, o que facilitaria a portabilidade do sistema resultante, que também será desenvolvido em Java.

A implementação de uma interface gráfica e de módulos de integração ocorrerá paralelamente ao desenvolvimento de dois aplicativos, um protótipo de um editor de textos para sinais libras e um protótipo de uma interface para um controlador de cadeira de rodas. Em um processo de retro-alimentação de informações adquiridas, as dificuldades na implementação destes protótipos, utilizando o SIGUS, será utilizada para melhorar o sistema. Equipes diferentes deverão trabalhar no desenvolvimento da plataforma SIGUS e dos protótipos, aumentando assim as possibilidades de troca de informações e de uma avaliação mais independente dos resultados.

Capítulo 4

Ferramentas

Para o desenvolvimento dos módulos e das técnicas para localizar a região dos olhos, algumas ferramentas livres e gratuitas foram utilizadas. Essas ferramentas auxiliam no ganho de tempo de desenvolvimento, pois oferecem recursos como, acesso ao dispositivo de captura e métodos de pré-processamento de imagem. Procurou-se utilizar apenas ferramentas livres, gratuitas e portáteis, para que o sistema final não possua limitações de sistemas operacionais e alto custo.

4.1 Java Media Framework

Java Media Framework (JMF) é uma API[1] desenvolvida pela Sun Microsystems para incorporar multimídia em *applets* e aplicações Java. Com este pacote podemos capturar, gravar e apresentar arquivos multimídia, independentemente da plataforma usada. O JMF define uma classe "plugin" que habilita desenvolvedores a estender a funcionalidade do JMF.

A arquitetura do JMF esta dividida em:

- Data source
- Capture device
- Player
- Processor
- DataSink

[1]Application Programming Interface - conjunto de rotinas e padrões estabelecidos por um software para utilização de suas funcionalidades

- Format
- Manager

O Data Source é a mídia a ser transmitida e que pode ser, somente de vídeo, somente de audio, ou a combinação dos dois. O Processor é responsável pelo processamento que é feito na mídia de entrada (Data Source). Além de reproduzir o conteúdo do Data Source, o Processor pode ter como saída uma mídia, que por sua vez pode entrar em um Player ou processor, assim por diante. O Processor interpreta o Format, que é o formato da mídia. O DataSink é a interface que lê a mídia do DataSource e reproduz em algum destino, por exemplo um arquivo.

Um Player tem como entrada um áudio ou vídeo e processa isto ou nos alto-falantes ou no vídeo, ou em ambos. Um player logicamente tem estados, por exemplo parado, ligado, etc. O Manager, é um objeto intermediário, que interfaceia dois objetos diferentes, como por exemplo, com o manager você pode criar um player a partir de um Data Source. Um Data Source origina-se a partir de um Capture device, que representa o dispositivo captura, como por exemplo microfone, ou câmera de vídeo.

O JMF foi utilizado nos protótipos como interface de comunicação com a *Webcam*. Através de métodos como o *FrameBuffer*, foi possível capturar quadros da câmera com certa facilidade e portabilidade, pois a API JMF fornece métodos para a captura do buffer de dados vindo direto da câmera e também por funcionar sob uma máquina virtual Java.

4.2 ImageJ

ImageJ - *Image Processing and Analysis in Java* - é um pacote para processamento de imagens de domínio público, de fontes abertos, escrito em Java e inspirado no NIH[2] Image[3]. Pode ser executado tanto como um *applet*, quanto como uma aplicação, em qualquer computador que tenha uma maquina virtual Java. Este pacote oferece recursos para exibição, edição, análise, processamento, desenvolvimento e manutenção de imagens nos mais diversos formatos, como TIFF, GIF, JPEG, BMP, DICOM, FITS. Além disto, oferece suporte a "stacks", um conceito que permite a manipulação de diversas imagens "simultaneamente". O ImageJ foi desenvolvido para ser uma arquitetura aberta e de fácil extensão através de módulos que podem ser escritos também em Java.

[2]National Institutes of Health

[3]Pacote para processamento de imagens de domínio público para Macintosh

Os protótipos desenvolvidos no corrente trabalho foram desenvolvidos sobre a plataforma do ImageJ. Em um formato de módulo para a ferramenta. Tal escolha foi determinada pela disponibilidade de classes e métodos oferecidos pela biblioteca ImageJ. Utilizou-se as classes disponíveis na biblioteca para um determinado ganho de tempo no desenvolvimento de técnicas mais simples, como o processo de binarização de imagem. Além disso, a biblioteca também oferece a documentação completa de todas classes e métodos ali contidos. Essa documentação, a *ImageJ Doc*, foi determinante no auxílio ao desenvolvimento dos protótipos, clarificando os métodos da biblioteca com definições e exemplos.

Capítulo 5

Transformada de Hough

5.1 Definição

A Transformada Hough é um método matemático proposto por Paul Hough em 1962, patenteada pela IBM e reformulada computacionalmente por Duda e Hart[ROD72] em 1972. A transformada é utilizada desde 1995 como um método de processamento digital de imagens. É um método poderoso na detecção de linhas e círculos a partir de imagens cujas bordas foram previamente realçadas [McL98]. É uma transformada não reversível que faz uma parametrização de formas geométricas dado sua localização, em um espaço acumulado (Espaço de Hough). De modo geral, a transformada trabalha em um domínio definido pelos possíveis parâmetros da equação que descreve o ente geométrico.

A transformada é dividida em dois tipos, a Clássica, que é utilizada para determinar formas geométricas regulares como retas, quadrados, retângulos, círculos, etc, e a Generalizada, que pode determinar formas bi-dimensionais de qualquer tamanho e orientação, porém necessita de um alto custo de processamento, sendo utilizada somente em casos onde a transformada clássica é inviável e em casos em que se necessita de um resultado melhor qualificado e sem prioridade de tempo. É bastante utilizada em análise anatômica de imagens médicas.

A principal vantagem da Transformada de Hough é uma certa tolerância a ruídos. Os ruídos não influenciam incorretamente na localização das formas geométricas. O único problema que ruídos na imagem podem causar é um certo gasto de processamento desnecessário, pois a transformada realiza uma varredura em todos os pontos da imagem.

Para utilizar a transformada, as bordas dos objetos da imagem devem ser previamente realçadas. No trabalho atual é feito uma detecção de bordas através de um módulo contido na ferramenta ImageJ e em seguida, uma binarização linear. A transformada é executada com a imagem composta de apenas valores 0 (para preto) e 255 (para branco).

A transformada é o processo principal para o funcionamento do protótipo da localização da região dos olhos, pois é utilizada para encontrar a borda da íris. Através dela é possível desenvolver um estimador da direção do olhar sem o uso de meios intrusivos e mesmo com uma simples *Webcam*, como no trabalho de Young ([DY95]). Foi possível também seu uso na aplicação em tempo real, experimentos citados no capítulo 10 demonstram uma média de tempo de execução da transformada suficiente para acompanhar os movimentos dos olhos e da cabeça.

5.2 Transformada de Hough para Retas

Teoricamente, a transformada de Hough pode ser utilizada na detecção de qualquer figura geométrica definida por equações paramétricas. No entanto, seu cálculo torna-se impraticável quando as equações são não-lineares e envolvem mais de 2 parâmetros, como é o caso das elipses [Pis03]. Dessa forma, o Espaço de Hough fica com a quantidade de dimensões diretamente proporcional à quantidade de parâmetros da forma geométrica, exigindo uma alta capacidade de processamento. A descrição a seguir, demonstrada através da Transformada de Hough clássica para detectar retas, serve apenas para facilitar o compreendimento da trasformada.

O princípio fundamental da Transformada de Hough é que existe um número infinito de retas que passam através de algum ponto, cada uma em uma orientação diferente. A finalidade da transformada é determinar quais destas retas teóricas passam através das retas de uma imagem, isto é, que retas sobrepõem ao mais próximo as retas da imagem. A figura 5.1 mostra uma imagem original (1) e a imagem (2) a representação das retas que melhor sobrepuseram as retas da imagem.

No caso de retas, descritas pela equação $y = ax + b$, temos um domínio bidimensional com dois eixos ortogonais para os parâmetros a e b. Aplicando a Transformada de Hough, temos um Espaço de Hough que determina uma

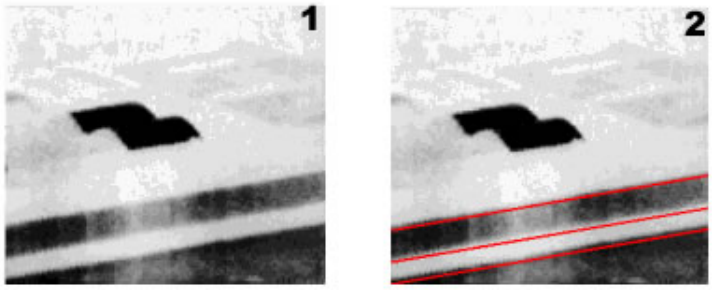

Figura 5.1: Representação das possíveis retas encontradas na imagem

possível reta e a existência desta reta na imagem original é calculada a partir de uma varredura nos pontos que participam das bordas realçadas. No entanto, para acumular esses valores no Espaço de Hough é necessário que ele tenha dimensões (a, b). Como a representa a inclinação da reta tangente, variando de $+\infty$ até $-\infty$, podemos notar que este caso torna inviável. Uma solução para este caso é a utilização da equação da reta com coordenadas polares, representada na equação 5.1.

$$r = x\cos\theta + y\sin\theta \tag{5.1}$$

Assim, cada reta é representada por dois parâmetros, r e θ, que representam respectivamente, a menor distância da reta em relação a origem e o ângulo da reta em relação ao eixo x em questão conforme ilustrado na figura 5.2. Dessa forma, o Espaço de Hough pode ser representado por r em relação a θ em um domínio finito.

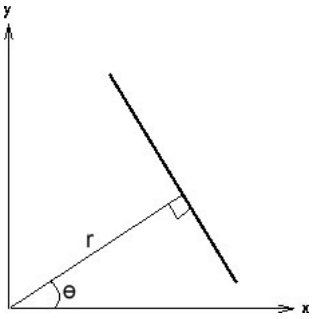

Figura 5.2: Representação da reta usando coordenadas polares

Para cada ponto (x, y) contido nesta reta, r e θ são constantes.

O valor das coordenadas (x, y) da imagem é conhecido e serve para encontrar os valores da equação da reta r e θ. Os valores da reta são encontrados da seguinte forma:

Para cada ponto (x, y) da imagem, é incrementado no Espaço de Hough (r, θ) todas as possíveis retas que passam pelo ponto.

Assim, a partir da característica invariante desses dois valores que representam a reta para cada ponto (x, y), é possível notar que a acumulação desses valores em um espaço bi-dimensional (r, θ) pode representar a informação de quantas retas são similares. Essa quantidade de acumulação para cada reta representa as retas que se sobrepõem nos valores da imagem. Em outras palavras, o espaço de Hough representa a probabilidade de ocorrência de reta na imagem.

Na prática, a imagem onde é realizada a busca deve possuir apenas valores binários. A figura 5.3 mostra a imagem original.

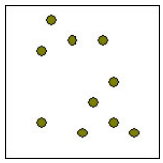

Figura 5.3: Representação da imagem

A partir da imagem original, é realizado uma busca para a representação das possíveis retas nos pontos (x, y) da imagem. A figura 5.4 ilustra a localização da reta.

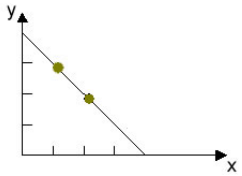

Figura 5.4: Representação da possível reta passante pelos dois pontos

Assim, a reta encontrada é incrementada no Espaço de Hough na posição (r, θ), representado pela figura 5.5.

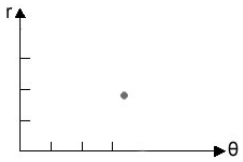

Figura 5.5: Espaço de Hough com o valor acumulado da reta

Sucessivamente todas as possíveis retas em uma imagem serão acumuladas no espaço de Hough, ao final, basta localizar os pontos de maior valor no Espaço de Hough para detectar as possíveis retas.

5.3 Transformada de Hough para Círculos

No trabalho presente, foi utilizada a transformada clássica para encontrar círculos na imagem como forma de localizar e rastrear a região da íris do olho humano. A diferença fundamental na utilização da transformada para detectar círculos é apenas na equação da forma geométrica, definida por:

$$(x-a)^2 + (y-b)^2 = R$$

Em que (x, y) é um ponto qualquer na imagem, (a, b) o centro do círculo e R o raio. Neste caso, é necessário uma pré fixação do valor raio do(s) círculo(s) a ser(em) encontrado(s), pois utilizando um Espaço de Hough tridimensional (a, b, R), o custo de processamento torna-se impraticável.

O algoritmo da Transformada de Hough, representado abaixo, percorre a imagem calculando o Espaço de Hough para cada ponto da imagem.

Apresentaremos agora uma representação prática da trasformada através de uma imagem gerada artificialmente contendo dois círculos idênticos de raio 40 a serem detectados e um círculo de raio 30. A figura 5.6, representa a imagem original binarizada.

Para aplicação da transformada a imagem deve estar com as bordas realçadas. A figura 5.7 ilustra a imagem após a detecção de bordas.

Com um raio pré-definido de 40 pixels para este exemplo, a visualização do espaço de Hough, representada na figura 5.8, aponta a probabilidade de ocorrer círculo na imagem. De fato, os pontos mais claros na imagem representam uma grande acumulação, assim definindo a existência de círculo naquela localização. Como neste caso utilizamos as coordenadas (a, b) do centro do círculo como plano cartesiano do espaço de Hough, a sua visualização aponta a localização exata do centro do possível círculo encontrado.

Algoritmo - Preenchimento do espaço de Hough

```
1: para x = 0 até Largura da Imagem faça
2:   para y = 0 até Altura da Imagem faça
3:     para θ = 0 até Altura do Espaço de Hough faça
4:       se Imagem[x][y] então
5:         r ← x cos(θ) + y sin(θ)
6:         Hough[r][θ] + +
7:       fim se
8:     fim para
9:   fim para
10: fim para
```

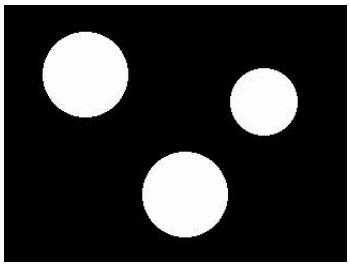

Figura 5.6: Imagem original com valores binários

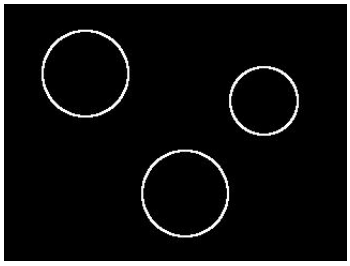

Figura 5.7: Imagem com as bordas realçadas

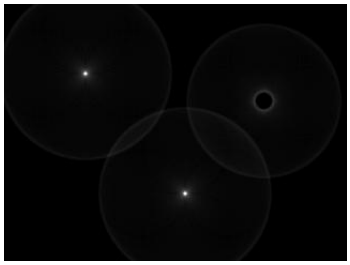

Figura 5.8: Visualização do espaço de Hough

Aplicando a transformada, o Espaço de Hough é incrementado com as possíveis localizações do círculo. Assim, resta apenas localizar os pontos de maior valor (branco) do espaço para poder determinar a localização dos círculos encontrados, representado na figura 5.9.

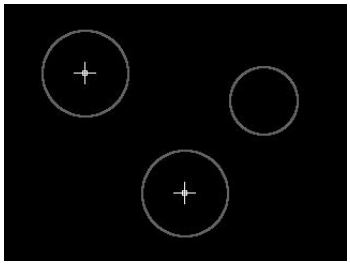

Figura 5.9: Localização do círculo na imagem

5.4 Transformada de Hough Aleatorizada

Também chamada de Transformada de Hough Randomizada, essa variação consiste basicamente na diferença em relação ao tamanho da amostra utilizada para localizar as formas geométricas. A transformada clássica percorre a imagem por completa podendo comprometer o desempenho de um sistema, como por exemplo, em sistemas de tempo real.

A transformada aleatorizada faz a varredura somente em uma determinada amostra da imagem escolhendo os pontos a serem verificados aleatoriamente. Autores como Young[DY95] utilizam cerca de 1/4 da imagem original para a varredura. De fato a quantidade a ser percorrida é variante para cada caso, em problemas que necessitarem de maior definição, como o de localizar a região o olho, é preciso uma varredura completa.

Capítulo 6

Protótipos Desenvolvidos

Foram desenvolvidos protótipos para o rastreamento da localização dos olhos utilizando linguagem de programação Java, e sobre a biblioteca *ImageJ* na forma de módulo. A metodologia de desenvolvimento dos protótipos foi baseada nos conceitos da plataforma SIGUS, estão documentados e com possibilidade de ser utilizados para outras aplicações. Os códigos-fonte dos protótipos desenvolvidos estão nos Anexos.

6.1 Interface com Webcam

Este módulo surgiu da necessidade de uma comunicação direta com o dispositivo de captura. Outros métodos já disponíveis de interface com a câmera não puderam ser aproveitados, pois em sua maioria eram específicos para determinado sistema operacional, não tinham métodos suficientes ou não eram facilmente manipuláveis. A partir desses problemas foi desenvolvido o módulo de interface com a *Webcam*.

O módulo de interface com a *Webcam* foi desenvolvido utilizando métodos e classes das ferramentas JMF e ImageJ. É um módulo para a ferramenta ImageJ bastante simples de ser utilizado. Através dele é possível capturar quadros do dispositivo de imagem que deve ser previamente configurado no JMF. Um de seus métodos, o *captureCamera* é responsável por capturar um quadro do dispositivo e armazenar em uma matriz a imagem no formato de cores RGB. Cada ponto da matriz é composto de três componentes, um para cada cor (RGB). Cada chamada deste método captura o quadro corrente, portanto, basta colocá-lo dentro de uma estrutura de controle de repetição (*loop*) para fazer a captura dos quadros em tempo real.

Este módulo possui três métodos:

1. startCamera()
2. stopCamera()
3. captureCamera()

Estes e outros métodos estão mais bem documentados no próprio código-fonte.

6.2 Pré-Processamento

O primeiro passo para o rastreamento da posição dos olhos, é a fase de pré-processamento através de software, que é realizada para cada quadro capturado da câmera. Com o quadro capturado e armazenado em uma matriz, o primeiro processo é a transformação da imagem no formato de cores RGB para tons de cinza. Essa transformação é feita a partir da fórmula 6.1, que é executada para cada ponto da imagem. Resulta em valores de 0 (preto) a 255 (branco).

$$P'[x][y] = (P[x][y][azul] * 0.11) + (P[x][y][vermelho] * 0.3) + (P[x][y][verde] * 0.59) \quad (6.1)$$

Em que P' é o ponto da nova imagem em tons de cinza e P é o ponto da imagem original no formato RGB.

Com a imagem em tons de cinza, o próximo passo é uma convolução que serve para amenizar ruídos na imagem conforme ilustrado na figura 6.2. Esses ruídos provém da própria câmera utilizada, pois a resolução da maioria das *Webcams* geralmente é baixa.

A máscara utilizada pela convolução está ilustrada na tabela 6.1.

1	0	1
1	0	1
1	0	1

Tabela 6.1: Máscara da convolução

O próximo passo a ser realizado, é a Detecção de Bordas através do algoritmo Sobel. A detecção serve para realçar as bordas da imagem para que

Figura 6.1: À esquerda, a imagem original, à direita, a imagem resultante da convolução

a Transformada de Hough possa ser executada posteriormente. Nesse algoritmo, são executados duas convoluções com máscaras distintas, uma para detectar bordas na vertical, representada na tabela 6.2, e a outra máscara para as bordas horizontais, representada na tabela 6.3.

1	2	1
0	0	0
-1	-2	-1

Tabela 6.2: Máscara da convolução vertical

1	0	-1
2	0	-2
1	0	-1

Tabela 6.3: Máscara da convolução horizontal

A imagem final, figura 6.2, é o a combinação das duas convoluções, representadas na figura 6.3.

A binarização ou limiarização é a conversão da imagem em tons de cinza para preto e branco e é último passo executado antes da Transformada de Hough. Essa conversão é feita de forma linear, através de um limiar de divisão. Este valor, escolhido empiricamente, é 80. Assim os pontos da imagem em tons de cinza que forem maiores que 80, são ajustados para 255 (branco) e os menores para 0 (preto). Para a escolha do valor não foram realizados testes e experimentos, pois uma variação do valor do mesmo não interfere significativamente no resultado já que a detecção de bordas deixa o histograma da imagem bastante definido. Assim um valor qualquer, não sendo próximo aos polos extremos da imagem (0 e 255), pode binarizar a imagem de forma

Figura 6.2: Imagem resultante da Detecção de Bordas

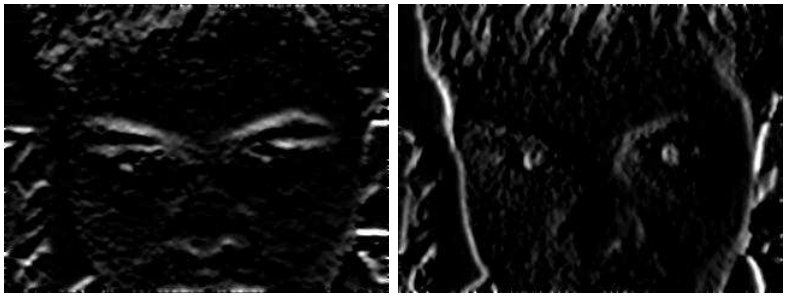

Figura 6.3: À esquerda, a imagem resultante da convolução vertical, à direita, a imagem resultante da convolução horizontal

satisfatória.

As imagens resultantes do pré-processamento pode ser observado na figura 6.4, e a numeração dos passos descrita abaixo:

1. Imagem original
2. Resultante da conversão para tons de cinza
3. Convolução de suavização
4. Detecção de Bordas Sobel
5. Imagem binarizada

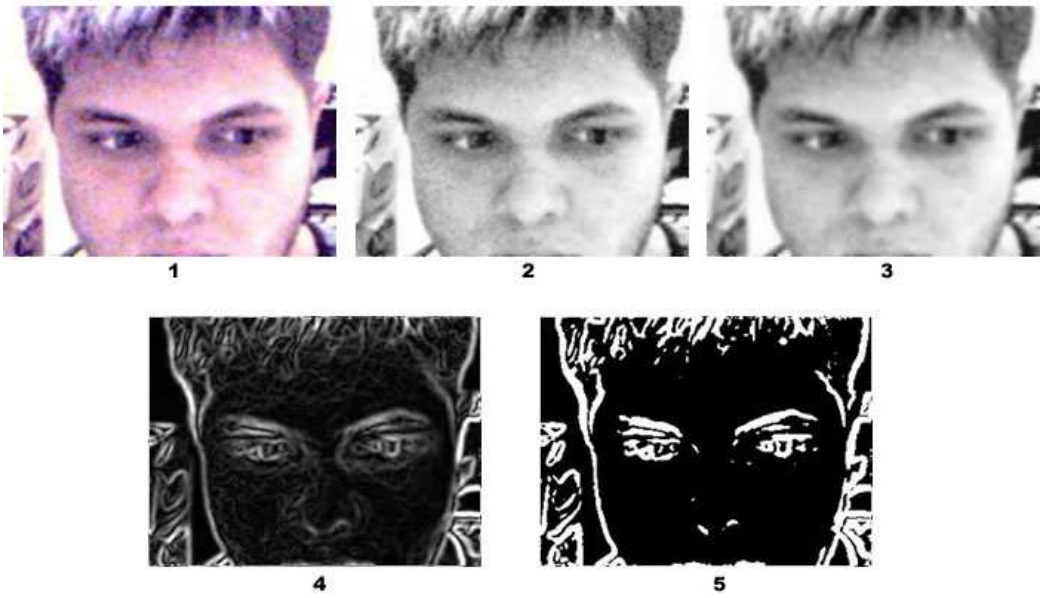

Figura 6.4: Sequência do pré-processamento

6.3 Eliminador de Ruídos

Ruídos e informações não desejadas fazem parte da imagem capturada pela câmera. Ruídos são geralmente más formações na imagem resultante de fatores externos como a iluminação e interferências. Outras informações não desejadas, como outras partes da face e/ou do ambiente, podem interferir no resultado da localização dos olhos.

O módulo Eliminador de Ruídos consiste basicamente de uma classe desenvolvida para eliminar ruídos e informações não desejadas na imagem. Consequentemente, esse módulo foi um fator de aumento na taxa de quadros por segundo, pois eliminando regiões não desejadas a varredura na imagem é significativamente diminuída.

O desenvolvimento desse protótipo surgiu a partir de erros cometidos na fase de localização dos olhos, que localizava pontos errados na imagem binarizada, como a região da sombrançelha e principalmente em certas partes do cabelo. A figura 6.5 ilustra esse erro.

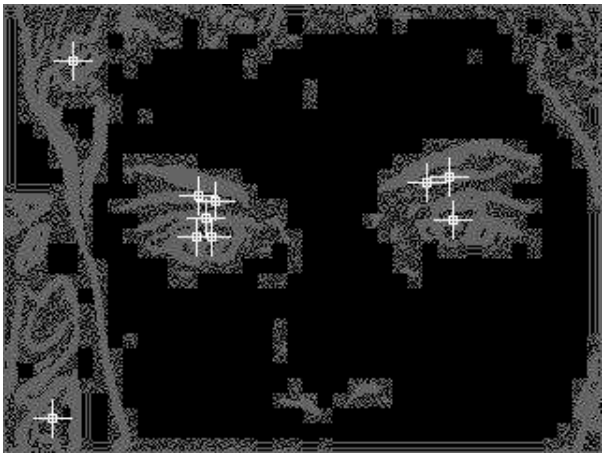

Figura 6.5: Erros comuns cometidos

Com a aplicação do módulo eliminador de ruídos, a imagem original 6.6 passa a conter apenas regiões essenciais como ilustrado na figura 6.7.

Figura 6.6: Imagem original antes da execução do eliminador de ruídos

Assim, a taxa de acerto e o desempenho da localização da região dos olhos acaba sendo aumentadas significativamente. A figura 6.8 ilustra os pontos candidatos a região dos olhos.

Seu algoritmo é baseado no princípio do "balde de tinta", algoritmo utilizado em técnicas de preenchimento de regiões e análise de partículas. Utiliza uma técnica recursiva para a localização da vizinhança de cada pixel da imagem. Para cada pixel, são buscados seus 4 vizinhos (conforme figura 6.9), e para cada novo vizinho encontrado, uma recursão é aberta buscando seus vizinhos.

Figura 6.7: Imagem após execução do eliminador de ruídos

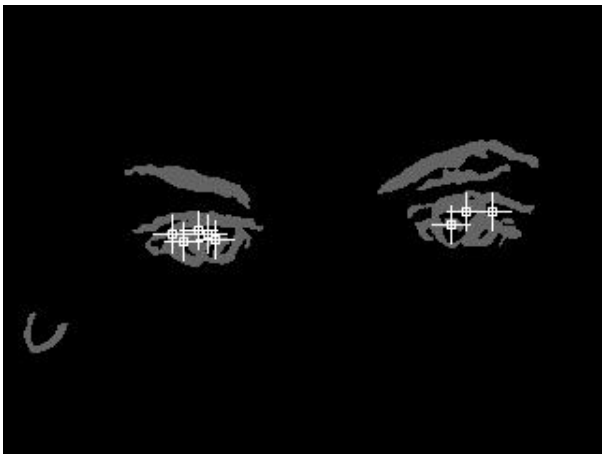

Figura 6.8: Acertos na localização

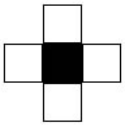

Figura 6.9: Vizinhos de cada pixel

Ao final da recursão, a região agrupada encontrada passa por um processo de seleção de validade ou não através das heurísticas abaixo:

- Quantidade de pixel
- Largura da região localizada
- Altura da região localizada
- Proporção da largura em relação a altura

Os valores de limiares superiores e inferiores, para cada heurística, foram baseados em um conjunto de 40 imagens capturadas para localizar a região dos olhos. Podemos notar que, a quantidade de pixels da região dos olhos não compreende um alto valor comparado a regiões agrupadas de outras partes da imagem, como por exemplo, o cabelo. A proporção da largura em relação a altura representa a orientação da região agrupada, podendo distinguir esta de regiões com forma vertical, como por exemplo, o nariz ou partes do cabelo.

6.4 Localizador dos Olhos

Baseado principalmente na Transformada de Hough Clássica, este módulo calcula a transformada sobre a imagem resultante do Eliminador de Ruídos marcando a região dos olhos. Funciona como uma estrutura de controle de repetição, exibindo na tela a imagem resultante do processo com os pontos marcados através de pequenas cruzes, conforme ilustrado na figura 6.10. Durante sua execução, é exibida uma janela informando dados sobre a coordenadas dos pontos e o tempo de processamento para cada quadro.

Este módulo funciona depedentemente dos módulos acima e de alguns outros algoritmos já implementados no ImageJ, como a Detecção de Bordas e a Suavização.

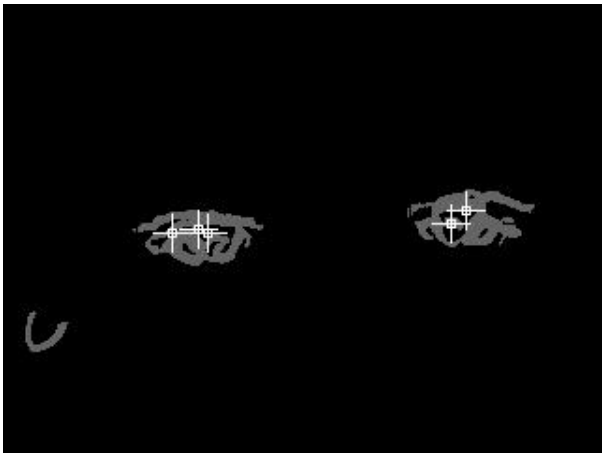

Figura 6.10: Resultado

O módulo Localizador dos Olhos é uma estrutura de controle de repetição que a cada repetição executa métodos dos outros módulos. Assim, seu conteúdo fica simplificado e de fácil ajuste. Primeiramente é chamado métodos de inicialização de variáveis e da câmera, em seguida é executada a estrutura de controle de repetição. O conteúdo da estrutura pode ser ilustrado pelo algoritmo abaixo:

Algoritmo - Estrutura de controle de repetição do Localizador dos Olhos

```
1: camera ←iniciaCamera()
2: enquanto 1 faça
3:     quadro ←capturaQuadro(camera)
4:     quadro ←preProcessaQuadro(quadro)
5:     quadro ←eliminador(quadro)
6:     pontos ←transformadaHough(quadro)
7:     DesenhaPontos(pontos)
8: fim enquanto
```

Capítulo 7

Experimentos e Resultados

Vários experimentos foram realizados com os protótipos. O principal ambiente de desenvolvimento dos protótipos contava com iluminação através de lâmpadas fluorescentes que causavam certa interferência no sinal da câmera. Esta interferência se deve à taxa de atualização da câmera, que funciona entre 50Hz e 60Hz, assim, sendo seu sinai interferido pelas lâmpadas que também trabalham próximo desta faixa. Para solucionar isto, foi necessário apenas ativar o filtro de banda de 60Hz no driver da webcam. No entanto, com esse filtro ativado a imagem resultante perde um pouco de qualidade, ficando com alguns ruídos e uma certa perda de luminosidade.

A *Webcam* utilizada em todo processo, foi a Creative Webcam Go Plus III com o chipset CT7510. Possui driver para plataforma Windows e Linux. Para o sistema operacional Linux, o driver utilizado foi o *W9968cf*[1]. Em todo momento, foi utilizado a resolução da imagem adquirida da câmera de 320 pixels de largura por 240 de altura. Essa proporção é a mais usual dentre os trabalhos realizados nesta área utilizando webcam, pois praticamente todas *Webcams* conseguem capturar imagens nesta resolução. Foi utilizado o sistema de cores RGB, pois também é aceito pela maioria dos equipamentos de captura.

O ambiente de desenvolvimento é composto por computador pessoal comum e uma Webcam posicionada em cima do monitor, na parte central. O computador pessoal possui um monitor de 17 polegadas, processador Athlon de 1.6 GHz e 512 MB de memória principal. Fotos do ambiente são apresentadas na figura 7.1.

[1]Winbond W9968CF JPEG USB Dual Mode Camera Chips - Pode ser obtido gratuitamente no sítio: http://go.lamarinapunto.com

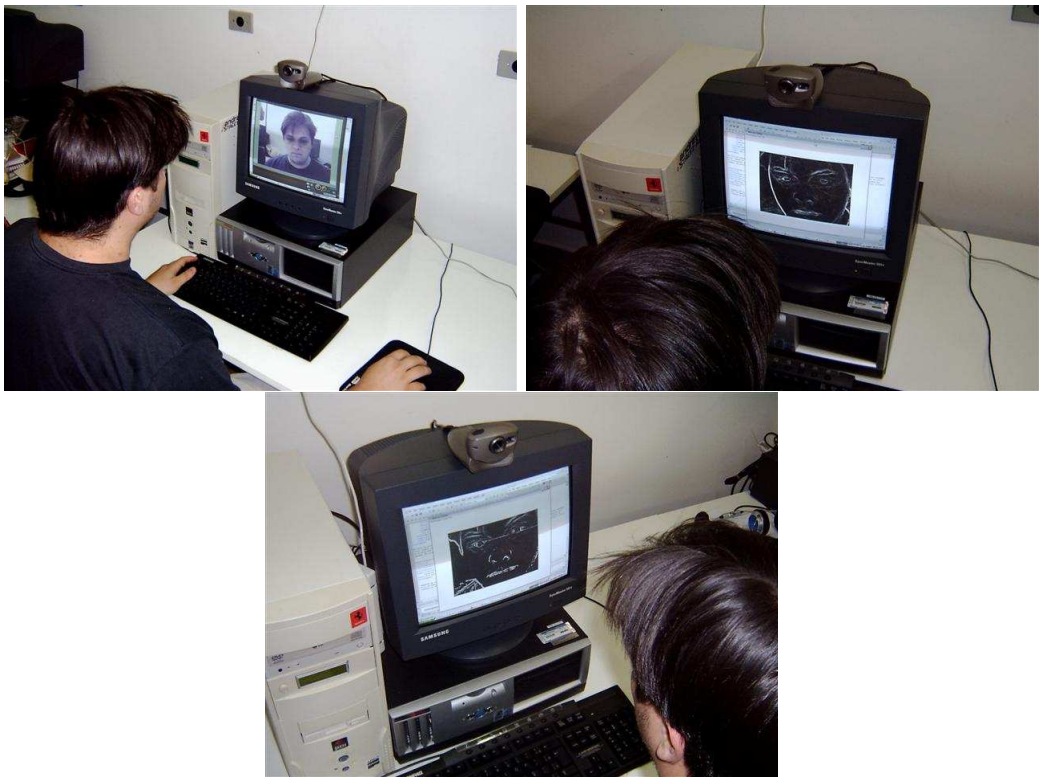

Figura 7.1: Ambiente de desenvolvimento

Através do ambiente citado foi possível obter uma taxa de quadros por segundo bastante satisfatória em relação a outros trabalhos nesta mesma área. Este valor é aproximadamente de 7 quadros por segundo.

Para realização de experimentos foi gerado um conjunto de dados[2] contendo 40 imagens de um usuário olhando em diferentes pontos na tela do computador. Para cada imagem do conjunto, foi executado o módulo localizador da região dos olhos, o qual foi adaptado para gerar na tela os resultados da possível localização dos olhos. Os dados de acertos e erros estão apresentados na tabela 7.1, e demonstra um resultado bastante satisfatório e promissor.

Na tabela 7.1, a taxa de acertos parciais representa os casos onde a localização dos olhos foi encontrada de forma deficitária, como por exemplo, a localização de um olho apenas, a localização da sombrançelha e outras partes da face. Um exemplo pode ser observado na figura 7.2, onde um olho foi encontrado corretamente, e outro ponto localizado correspondende a parte do cabelo.

Acertos	29	72,5%
Acertos Parciais	4	10%
Erros	7	17,5%

Tabela 7.1: Resultados do experimento

Figura 7.2: Processo de caso de acerto parcial. Imagem original, à esquerda, após eliminação de ruídos, ao centro, e após localizador dos olhos, à direita

Um exemplo de um caso de acerto pode ser observado na figura 7.3.

[2]O conjunto pode ser obtido através da página do projeto: http://www.ec.ucdb.br/~emoretto

Figura 7.3: Processo de um caso acertado. Imagem original, à esquerda, após eliminação de ruídos, ao centro, e após localizador dos olhos, à direita

Em alguns casos de imagens considerados difíceis ou com muito ruído, o módulo desenvolvido localizou corretamente a região dos olhos conforme ilustrado na figura 7.4.

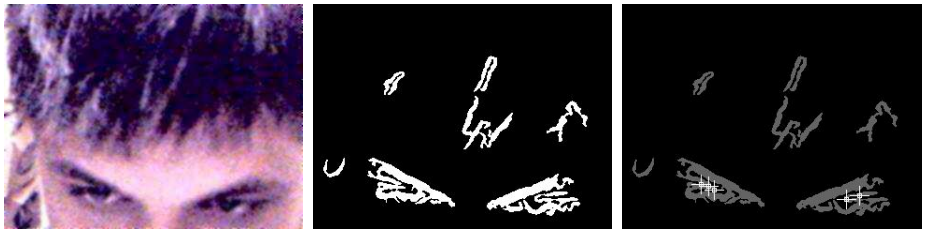

Figura 7.4: Processo de um caso considerado difícil. Imagem original, à esquerda, após eliminação de ruídos, ao centro, e após localizador dos olhos, à direita

A taxa de erro corresponde a casos onde a localização do olho foi falha, como por exemplo, a localização de objetos localizados no ambiente, localização de outras regiões da face. A tabela 7.2 ilustra as principais causas de erros.

Outros pontos na face	4	57,1%
Outros objetos	2	28,6%
Iluminação	1	14,3%

Tabela 7.2: Causas de erro

Na prática podemos ilustrar um desses casos de erro na figura 7.5, que localizou incorretamente objetos no ambiente. Podemos notar também na figura 7.5 uma certa quantidade de ruídos a mais em relação a outros casos de teste, isso se deve ao fato da proximidade do usuário em relação à câmera. Essa proximidade faz com que o dispositivo de captura não consiga obter um foco definido, formando ruídos na imagem. De acordo com as especificações técnicas da *Webcam* utilizada, recomenda-se uma distância de pelo menos 30 centímetros para um foco correto.

Figura 7.5: Processo de um caso de erro. Imagem original, à esquerda, após eliminação de ruídos, ao centro, e após localizador dos olhos, à direita

Capítulo 8

Considerações Finais

A tarefa de rastrear a região dos olhos apresenta-se como uma inovadora e estimulante área da computação. Justificativas não faltam nesse tema, como questões sociais, tecnológicas e científicas nos estimulam no desenvolvimento dessa nova abordagem. Vários autores comprovam em seus trabalhos uma gama de utilidades ligadas a este tema.

O estudo realizou um levantamento bibliográfico baseado em artigos, páginas na internet e livros. Esse levantamento serviu de base para abordagem mais detalhada do tema e para futuros trabalhos nesta área. A partir desse levantamento foi possível realizar uma análise geral nas técnicas conhecidas para obter um domínio detalhado de todo o problema e assim desenvolver protótipos para o rastreamento da região dos olhos. Com o domínio do problema foi possível formar justificativas e respostas para as decisões tomadas.

Foram desenvolvidos alguns módulos para a ferramenta ImageJ com o aprendizado a partir da apostila desenvolvida por Werner Bailer[Bai03]. A implementação da transformada de Hough clássica foi desenvolvida baseada na implementação de Pistori[Pis03]. Através da transformada, o estudo conseguiu localizar e rastrear a região dos olhos satisfatoriamente. Todo o material desenvolvido durante este estudo está disponível. Códigos-fontes abertos e documentados, testes realizados e dificuldades encontradas também foram armazenados e disponibilizados.

O objetivo de compartilhar informações e ajudar outros membros do projeto SIGUS foi alcançado. A todo momento, tudo o que foi desenvolvido foi disponibilizado a outros pesquisadores do projeto.

Resta a trabalhos futuros uma continuação do mesmo a fim de estimar a direção do olhar. Para a continuação, um aprimoramento detalhado nas técnicas desenvolvidas podem ajudar no desempenho e na taxa de acerto do sistema. Seria interessante também uma análise detalhada de técnicas geométricas para estimar a direção do olhar, pois os estudos que utilizam essa característica possuem geralmente uma boa taxa de acerto e um desempenho excelente comparado à outras técnicas.

A ferramenta Weka - *Waikato Environment for Knowledge Analysis* - é uma biblioteca de algoritmos de aprendizado de máquina e mineração de dados e que pode ser utilizada para a comparação e escolha de uma técnica de aprendizagem de máquina para detectar a direcção do olhar. A ferramenta é composta por uma interface gráfica onde é possível realizar experimentos e análises entre os algoritmos e conjuntos de dados em pouco tempo.

O material desenvolvido, informações técnicas, códigos-fontes e um conjunto de dados podem ser encontrados na página do trabalho.
http://www.ec.ucdb.br/~emoretto

Anexo A

Código-fonte: Interface com Webcam

```
package camera;

import java.io.*;
import java.util.*;
import java.awt.*;
import java.awt.image.*;
import javax.imageio.*;
import javax.media.*;
import javax.media.control.*;
import javax.media.format.*;
import javax.media.util.*;

/*

GetFrame by Emerson G Moretto (emoretto@ec.ucdb.br)

Essa classe captura quadros da webcam no formato RGB.
Funciona em multiplataforma.
É necessário configurar no JMRegistry o modelo da câmera previamente.

http://www.ec.ucdb.br/~emoretto

 */


public class GetFrame {

    private Player player = null;
    private int sleep = 0;
    private boolean flagLigada = false; //Camera ligada ou nao
```

```
public GetFrame(int sleep) {
    this.sleep = sleep;
}

public void startCamera() throws Exception {

    //Localizador de Dispositivos de vídeo no formato RGB
    CaptureDeviceManager bossMan = new CaptureDeviceManager();
    //Pega uma lista de dispositivos e formatos encontrados
    Vector list = bossMan.getDeviceList(new RGBFormat());
    //Seleciona o primeiro elemento da lista
    CaptureDeviceInfo device = (CaptureDeviceInfo) list.elementAt(0);
    MediaLocator loc = device.getLocator();

    //Cria a interface de controle da câmera
    this.player = Manager.createRealizedPlayer(loc);
    this.player.start();

    //Espera sleep segundos para a camera ser inicializada
    Thread.sleep(this.sleep);
    flagLigada=true;
}

public void stopCamera() {
    this.player.close();
    this.player.deallocate();
    flagLigada = false;
}

public boolean Ligada() {
    return flagLigada;
}

public BufferedImage captureCamera() {
    // Captura um quadro da câmera
    FrameGrabbingControl frameGrabber = (FrameGrabbingControl)
    this.player.getControl("javax.media.control.FrameGrabbingControl");

    Buffer buf = frameGrabber.grabFrame();

    //Converte um quadro para um buffer de imagem para ser processado
    Image img =
    (new BufferToImage((VideoFormat)buf.getFormat()).createImage(buf));

    BufferedImage buffImg =
    new BufferedImage(
     img.getWidth(null),
     img.getHeight(null),
     BufferedImage.TYPE_INT_RGB);
```

```
        Graphics2D g = buffImg.createGraphics();
        g.drawImage(img, null, null);
        return(buffImg);
    }
}
```

Anexo B

Código-fonte: Eliminador de Ruídos

```
import ij.*;
import ij.plugin.filter.PlugInFilter;
import ij.plugin.*;
import ij.process.*;
import ij.gui.*;
import java.awt.*;

/*
 * Eliminator by Emerson G Moretto (emoretto@ec.ucdb.br)
 *
 * - Este plugin retira os ruidos e falsos candidatos a olho.
 * Utilizado na fase de pre-processamento da imagem para
 * localizar a regiao do olho atraves da Transformada de Hough
 *
 * - Este codigo foi projetado para ter um menor custo
 * de processamento, portanto haver algumas desnecessidades aparentes.
 *
 * http://www.ec.ucdb.br/~emoretto
 */

public class Eliminator_ implements PlugInFilter {

    //Variaveis gerais

    private int width; // Largura da Imagem
    private int height;  // Altura da Imagem
    private byte vet[],vetTemp[] ;

    //Variaveis do Algoritmo Recusivo
    private int direita;
```

```
private int esquerda;
private int cima;
private int baixo;
private int qtd_pixel;

//Variaveis de selecao de candidatos
private int w_max = 100;
private int w_min = 20;
private int h_max = 100;
private int h_min = 5;
private int qtd_pixel_min = 100;
private int qtd_pixel_max = 2000;
private double fator_wl = 0.34; //razao altura/largura


public int setup(String arg, ImagePlus imp) {

    return DOES_8G;
}


public void run(ImageProcessor ip) {

    ////////Declaracao de variaveis
    int i, j, size, offset, count,xr,yr,count_candidato, largura, altura;
    boolean candidato;
    boolean debug = false;

    ////////Codigo
    width  = ip.getWidth();
    height = ip.getHeight();
    size = width * height;
    vet = new byte[ width * height ];
    vet = (byte[]) ip.getPixels();
    vetTemp = new byte [width * height];
    count = 0;
    count_candidato=0;

    //Binarizacao
    IJ.run("Threshold");

    for(i = 0 ; i < width ; i++)
        for(j = 0 ; j < height ; j++) {
            offset = i+width*j;

            direita = 0;
            esquerda = width;
            cima = height;
            baixo = 0;
```

```
            candidato = false;
            largura = 0;
            altura = 0;

            //se o um pixel branco, entao percorre esse candidato
            if(vet[offset] == (byte) -1 && vetTemp[offset] == (byte) 0) {
                vetTemp[offset] = (byte) -1;
                qtd_pixel = 1+ busca(i,j,false);
                count++;

                largura = direita - esquerda;
                altura = baixo - cima;

                //aplicacao das heuristicas
                if( largura < w_max && largura > w_min &&
                altura < h_max && altura > h_min &&
                qtd_pixel < qtd_pixel_max && qtd_pixel > qtd_pixel_min &&
                fator_wl < (double) altura/largura  ) {
                    candidato = true;


                    //desenha um retangulo em volta do candidato
                    for(xr = esquerda-1 ; xr <= direita+1 ; xr++)
                        for(yr = cima-1 ; yr <= baixo+1 ; yr++)
                            if(xr == esquerda-1 || xr == direita+1 ||
                             yr == cima-1 || yr == baixo+1)
                                vet[xr+width*yr] = (byte)100;


                }else{
                    vetTemp[offset] = (byte) -1;
                    vet[offset] = (byte) 0;
                    busca(i,j,true);    // Manda apagar o candidato falso
                }


                if(candidato)
                    count_candidato++;
            }


            IJ.write("Encontrado "+count+" particulas");
            IJ.write("Encontrado "+count_candidato+" candidatos");
        }

    //Essa funcao recursiva e busca os 4 vizinhos (tipo carbono) do pixel
    //parametro apaga serve para apagar a particula quando true

    private int busca(int x, int y,boolean apaga) {
```

```
int conta;
conta = 0;

if((y+1) < height)
    if(vet[x+width*(y+1)] == (byte)-1 &&
    ( vetTemp[x+width*(y+1)] == (byte) 0  || apaga)) // Para baixo
    {
        vetTemp[x+width*(y+1)] = (byte)-1;

        if(apaga)
            vet[x+width*(y+1)] = (byte) 0;

        if(baixo < (y+1))
            baixo = (y+1);

        conta+= 1 + busca(x,y+1,apaga);
    }

if((x-1) > 0)
    if(vet[(x-1)+width*y] == (byte)-1 &&
    (vetTemp[(x-1)+width*y] == (byte) 0 || apaga)) // Para tras
    {
        vetTemp[(x-1)+width*y] = (byte)-1;

        if(apaga)
            vet[(x-1)+width*y] = (byte) 0;

        if(esquerda > x-1)
            esquerda = x-1;

        conta+= 1 + busca(x-1,y,apaga);
    }

if((y-1) > 0)
    if(vet[x+width*(y-1)] == (byte)-1 &&
    (vetTemp[x+width*(y-1)] == (byte) 0 || apaga)) // Para cima
    {
        vetTemp[x+width*(y-1)] = (byte)-1;

        if(apaga)
            vet[x+width*(y-1)] = (byte) 0;

        if(cima > (y-1))
            cima = (y-1);

        conta+= 1 + busca(x,y-1,apaga);
    }
```

```
            if((x+1) < width)
                if(vet[(x+1)+width*y] == (byte) -1 &&
                (vetTemp[(x+1)+width*y] == (byte) 0 || apaga)) // Para frente
                {
                    vetTemp[(x+1)+width*y] = (byte) -1;

                    if(apaga)
                        vet[(x+1)+width*y] = (byte) 0;

                    if(direita < x+1)
                        direita = x+1;

                    conta+= + busca(x+1,y,apaga);
                }

            return conta;

        }


        void showAbout() {
            IJ.showMessage("About ...",
            "This plugin remove particles from a image\n" +
            " althrough four heuristics described in the code.\n" +
            " developed by Emerson G Moretto(emoretto@ec.ucdb.br)"
            );
        }


    }
```

Anexo C

Código-fonte: Localizador da Região dos Olhos

```
import ij.*;
import ij.plugin.filter.PlugInFilter;
import ij.plugin.*;
import ij.process.*;
import ij.gui.*;
import java.awt.*;
import java.awt.image.*;
import javax.media.Player;
import camera.*;


/*
 * Eye Tracker by Emerson G Moretto (emoretto@ec.ucdb.br)
 *
 * - Este Plugin busca os olhos e marca uma cruz no centro da pupila)
 *
 * Baseado na implementação da Transformada de Hough para encontrar círculos
 *feito por Prof. Dr. Hemerson Pistori (pistori@ec.ucdb.br)
 *
 *
 *http://www.ec.ucdb.br/~emoretto
 */

public class EyeTracker_ implements PlugInFilter {

    public float raio; // Raio do circulo a ser encontrado
    public int maxCircles; // Numero de circulos a ser encontrado
    byte imageValues[]; // Vetor da imagem a ser processada
    byte imageOri[]; // Imagem Original
```

```
double houghValues[][]; // Espaço de Hough
public int largura; // Hough Space largura
public int altura;  // Hough Space heigh
Point centerPoint[]; // Center Points of the Circles Found.


//WEBCAM
public Player player=null;
public GetFrame camera = null;

public int setup(String arg, ImagePlus imp) {

    return DOES_8G;
}


public void run(ImageProcessor ip) {


    //Inicializacao da camera com atraso de 1000 milisegundos
    camera = new GetFrame(1000);
    BufferedImage buffImg = null;


    try{
        camera.startCamera();
    }catch(Exception e){
        IJ.write("Não foi possível inicializar a câmera.\n
         Tente reiniciar o ImageJ.");
    }

    int[] pixels;
    int j=0,altura2,largura2,blue,green,red,pixel_temp,x=0,y=0,k=0,x2=0,y2=0,t;

    try{

        while(true) {
            j++;

            //Capturando o quadro da câmera
            buffImg=camera.captureCamera();

            //Setando as variaveis
            pixels = new int[ largura * altura ];
            imageValues = (byte[])ip.getPixels();
            imageOri = (byte[])ip.getPixels();
            Rectangle r = ip.getRoi();
            raio = 20;
            maxCircles = 10;
```

```
pixels = buffImg.getRGB(0,0,largura,altura,pixels,0,largura);


//Conversao para Tons de Cinza
for(i = 0 ; i <  largura * altura ; i++) {

    blue = pixels[i]& 0xff;
    green = (pixels[i]>> 8) & 0xff;
    red = (pixels[i] >> 16) & 0xff;

    pixel_temp = 0;
    pixel_temp += (blue * 0.11);
    pixel_temp += (red * 0.3);
    pixel_temp += (green * 0.59);

    imageValues[i] = (byte) pixel_temp;
    imageOri[i] = (byte) pixel_temp;

}

//Deteccao de Bordas Sobel
IJ.run("Find Edges");

//Binarizacao
for(i = 0 ; i <  largura * altura ; i++) {
    if(imageValues[i] > (int) 80)
        imageValues[i] = (byte) 255;
    else
        imageValues[i] = (byte) 0;
}


//Transformada de Hough
houghTransform();


////Desenha as cruzes nos pontos onde achou o olho,
if(hough) {
    if(j>2) {
        for(t=0; t < 10; t++) {
            x = centerPoint[t].x;
            y = centerPoint[t].y;
            for(k=-10; k < 11 ; k++) {
                if((x+k+largura*y) > 10 &&
                (x+k+largura*y) < altura * largura - 10) {
                    imageValues[(x+k)+largura*y] = (byte) 255;

                }
```

```
                            if((x+largura*(y+k)) > 10 &&
                            (x+largura*(y+k)) < altura * largura - 10) {
                                imageValues[x+largura*(y+k)] = (byte) 255;
                            }

                        }
                    }
                }
            }

            //Pontos encontrados
            x = centerPoint[0].x;
            y = centerPoint[0].y;

            x2 = centerPoint[1].x;
            y2 = centerPoint[1].y;


            IJ.write("Primeiro ponto encontrado x:"+x+" y:"+y);
            IJ.write("Segundo ponto encontrado x:"+x2+" y:"+y2);


        }
    }
    catch(Exception e) {
        IJ.write("Erro!");
        if(camera.Ligada())
            camera.stopCamera();
    }
    finally {
        camera.stopCamera();
    }
}


void showAbout() {
    IJ.showMessage("About ...",
    "This plugin finds a eyes region\n" +
    "using a basic HoughTransform operator\n\n." +
    "\nAuthor: Emerson G Moretto (emoretto@ec.ucdb.br)"+
    "based on Hemerson Pistori (pistori@ec.ucdb.br)"
    );
}


private void houghTransform() {

    int i = 0;
    int j = 1;
```

```
        int k = largura - j;
        int l = altura - j;

        houghValues = new double[largura][altura];
        int i2 = Math.round(8F * raio);
        int ai[][] = new int[2][i2];
        for(int j2 = 0; j2 < i2; j2++) {
            double d1 = (6.2831853071795862D * (double)j2) / (double)i2;
            int k1 = (int)Math.round((double)raio * Math.cos(d1));
            int l1 = (int)Math.round((double)raio * Math.sin(d1));
            if((i == 0) | (k1 != ai[0][i]) & (l1 != ai[1][i])) {
                ai[0][i] = k1;
                ai[1][i] = l1;
                i++;
            }
        }

        double d;

        for(int y = j; y < l; y++) {
            for(int x = j; x < k; x++) {
                if( imageValues[(x+offx)+(y+offy)*offset] == 0 )
                    d = 0;
                else
                    d = 1;
                if(d != 0.0D) {
                    for(int i3 = 0; i3 < i; i3++) {
                        int i1 = y + ai[0][i3];
                        int j1 = x + ai[1][i3];
                        if((i1 >= 0) & (i1 < altura) & (j1 >= 0) & (j1 < largura)) {
                            houghValues[j1][i1] += d;
                        }
                    }
                }
            }
        }
    }

    private void createHoughPixels(byte houghPixels[]) {
        double d = -1D;
        for(int j = 0; j < altura; j++) {
            for(int k = 0; k < largura; k++)
                if(houghValues[k][j] > d) {
                    d = houghValues[k][j];
                }
        }
        for(int l = 0; l < altura; l++) {
            for(int i1 = 0; i1 < largura; i1++) {
                houghPixels[i1 + l * largura] =
```

```
                (byte) Math.round((houghValues[i1][l] * 255D) / d);
            }

        }
    }

    private void getCenterPoints(int i) {
        centerPoint = new Point[maxCircles];
        double d2 = raio / 2.0F;
        double d3 = d2 * d2;
        int j1 = 0;
        int k1 = 0;
        for(int l1 = 0; l1 < i; l1++) {
            double d1 = -1;
            for(int i2 = 0; i2 < altura; i2++) {
                for(int j2 = 0; j2 < largura; j2++)
                    if(houghValues[j2][i2] > d1) {
                        d1 = houghValues[j2][i2];
                        j1 = j2;
                        k1 = i2;
                    }
            }
            centerPoint[l1] = new Point(j1, k1);
            int j = (int)Math.floor((double)k1 - d2);
            if(j < 0)
                j = 0;
            int k = (int)Math.ceil((double)k1 + d2) + 1;
            if(k > altura)
                k = altura;
            int l = (int)Math.floor((double)j1 - d2);
            if(l < 0)
                l = 0;
            int i1 = (int)Math.ceil((double)j1 + d2) + 1;
            if(i1 > largura)
                i1 = largura;
            for(int k2 = j; k2 < k; k2++) {
                for(int l2 = l; l2 < i1; l2++)
                    if(Math.pow(l2 - j1, 2D) + Math.pow(k2 - k1, 2D) < d3)
                        houghValues[l2][k2] = 0.0D;

            }
        }
    }
}
```

Referências Bibliográficas

[AH] Irfan Essa Antonio Haro. A non-invasive computer vision system for reliable eye tracking.

[AH00] Infan Essa Antonio Haro, Myron Flickner. Detecting and tracking eyes by using their physiological properties, dynamics, and appearance. *IEEE Conference on Intelligent Transportation System*, 2000.

[AH03] Irfan A. Essa Antonio Haro, Gregory Abowd. Perceptual user interfaces using vision-based eye tracking, 2003.

[AJG95] Theo Engell-Nielsen Arne John Glenstrup. Eye controlled media: Present and future state. 1995.

[Bai03] Werner Bailer. Writing imagej plugins tutorial, 2003.

[CC95] S. De Magistris C. Colombo, A. Del Bimbo. Human-computer interaction based on eye movement tracking. *Computer Architectures for Machine Perception - September, 1995 - Italy*, 1995.

[CHM02] Myron Flickner Carlos H. Morimoto, Arnon Armir. Detecting eye position and gaze from a single camera and 2 light sources. *16th International Conference on Pattern Recognition*, 2002.

[DCR04] Michael J. Spivey Daniel C. Richardson. Eye-tracking: Characteristics and methods, 2004.

[DY95] Richard Samuels David Young, Hilary Tunley. Specialised hough transform and active contour methods for real-time eye tracking, 1995.

[HP04] Amaury Antônio de Castro Junior Mauro Conti Pereira e Tania Regina Vilela dos Santos Hemerson Pistori, João José Neto. Sigus - plataforma de apoio ao desenvolvimento de sistemas para inclusão digital de pessoas com necessidades especiais, 2004.

[Hyr97] Aulikki Hyrskykari. Gaze control as an input device, 1997.

[Jac91] Robert J. K. Jacob. The use of eye movements in human-computer interaction techniques: What you look at is what you get, 1991.

[JGW00] Eric Sung Jian-Gang Wang. Gaze determination via images of irises, 2000.

[JGW03] Ronda Venkateswarlu Jian-Gang Wang, Eric Sung. Eye gaze estimation from a single image of one eye. *Ninth International Conference on Computer Vision*, 2003.

[JZ02] Jie Yang Jie Zhu. Subpixel eye gaze tracking, 2002.

[KHT02] Narendra Ahuja Kar-Han Tan, David Kriegman. Appearance-based eye gaze estimation. *Sixth IEEE Workshop on Applications of Computer Vision (WACV'02)*, 2002.

[KNK99] R.S. Ramakrishna Kyung-Nam Kim. Vision-based eye-gaze tracking for human computer interface, 1999.

[LQX98] P. Sheppard L.-Q. Xu, D. Machin. A novel approach to real-time non-intrusive gaze finding. *British Machine Vision Conference*, 1998.

[McL98] Robert McLaughlin. Randomized hough transform: Improved ellipse detection with comparison. Technical Report JP98-01, 1998.

[MJJ98] Tomaso Poggio Michael J. Jones. Multidimensional morphable models, 1998.

[MRMM03] Carlos H. Morimoto Marcio R. M. Mimica. A computer vision framework for eye gaze tracking. *XVI Brazilian Symposium on Computer Graphics and Image Processing*, 2003.

[Pis03] Hemerson Pistori. *Tecnologia Adaptativa em Engenharia de Computação: Estado da Arte e Aplicações*. PhD thesis, Universidade Católica Dom Bosco, 2003.

[Pis04] Hemerson Pistori. Plataforma de apoio ao desenvolvimento de sistemas guiados por sinais visuais, 2004.

[QJ02] Zhiwei Zhu Qiang Ji. Eye and gaze tracking for interactive graphic display. *Int. Symp. on Smart Graphics - June, 2002 - USA*, 2002.

[RB02] M. Mazo R. Barea, L. Boquete. System for assisted mobility using eye movements based on electrooculography. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, pages 209 – 218, 2002.

[ROD72] Peter E. Hart Richard O. Duda. Use of the hough transformation to detect lines and curves in pictures, 1972.

[RS97] A. Waibel R. Stiefelhagen, J. Yang. Tracking eyes and monitoring eye gaze, 1997.

[SA03] John Gowdy Subramanya Amarnag, Raghunandan Kumaran. Real time eye tracking for computer interfaces. *IEEE International Conference on Multimedia and Expo (ICME)*, 2003.

[SB04] Dean Pomerleau Shumeet Baluja. Non-intrusive gaze tracking using artificial neural networks, 2004.

[Sys] Gaze Tracking System. Just blink your eyes: A head-free.

[TR98] M. Jones T. Rikert. Gaze estimation using morphable models, 1998.

[ZZ02a] Kikuo Fujimura Zhiwei Zhu, Qiang Ji. Combining kalman filtering and mean shift for real time eye tracking under active ir ilumination. *16th International Conference on Pattern Recognition*, 2002.

[ZZ02b] Quiang Ji Zhiwei Zhu, Kikuo Fujimura. Real-time eye detection and tracking under various light conditions. *ACM ETRA 2002 - New Orleans*, 2002.