UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

FERRAMENTA PARA DETECÇÃO DE FADIGA EM MOTORISTAS BASEADA NO MONITORAMENTO DOS OLHOS

RAFAEL DATTINGER

BLUMENAU
2009

2009/1-16

RAFAEL DATTINGER

FERRAMENTA PARA DETECÇÃO DE FADIGA EM MOTORISTAS BASEADA NO MONITORAMENTO DOS OLHOS

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. Dalton Solano dos Reis, M. Sc. - Orientador

BLUMENAU
2009

2009/1-16

FERRAMENTA PARA DETECÇÃO DE FADIGA EM MOTORISTAS BASEADA NO MONITORAMENTO DOS OLHOS

Por

RAFAEL DATTINGER

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

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

Blumenau, 08 de julho de 2009.

Dedico este trabalho a minha família, pai, mãe, irmã, a minha namorada e claro a todos os amigos, especialmente aqueles que me ajudaram diretamente na realização deste.

AGRADECIMENTOS

À minha família, por acreditar na minha capacidade e pelo apoio recebido durante todo o processo de graduação.

Aos meus amigos e namorada, pela motivação e compreensão nas horas difíceis.

À Israel Damásio Medeiros, pelo esclarecimento de dúvidas em relação ao desenvolvimento da monografia.

Ao professor Dalton Solano dos Reis, pela orientação e apoio no desenvolvimento deste trabalho.

Aos voluntários, por disponibilizar seu tempo e imagens para os testes presente neste projeto.

A Deus, por tudo.

Não existem métodos fáceis para resolver problemas difíceis.

René Descartes

RESUMO

No presente trabalho são apresentados meios para tentar solucionar problemas relativos a acidentes com veículos e a fadiga em motoristas. Utilizando-se de uma *webcam* para o monitoramento dos olhos de modo que em tempo real seja possível a detecção de fadiga do condutor. São reutilizadas as classes e biblioteca do projeto Visage (RESTOM, 2006), onde a maioria destas são modificadas para se atender o objetivo proposto pelo presente trabalho. Também são utilizadas técnicas de processamento de imagens, transformada de Hough, histograma de imagens, binarização de imagens digitais, entre outras. A reutilização dessas classes e a combinação dessas técnicas possibilitaram o desenvolvimento de um estudo para verificar a taxa de acerto em relação à variação do ambiente de captura, por exemplo, diferentes rostos e diferenças nas condições de iluminação. Observou-se que numa situação controlada de iluminação é possível variar os rostos e atingir uma taxa de acerto satisfatória na identificação de fadiga.

Palavras-chave: Fadiga. Acidentes. Imagem digital. Processamento de imagens. Monitoramento dos olhos. Webcam.

ABSTRACT

In the present work are given the means to try to solve problems relating to accidents involving vehicles and fatigue on drivers. By using a webcam to monitor the eyes so that real time is possible to detect fatigue in drivers. Classes and library of the project Visage (RESTOM, 2006) are utilized again, where most of these are modified to meet the objective proposed by this work. Are also used techniques of image processing, Hough transform, histogram of images, digital image binarization, among others. The reuse of these classes and the combination of these techniques allowed the development of a study to determine the rate of success in relation to environmental variation of capture, for example, different faces and differences in lighting conditions. We found that a controlled lighting situation you can change the faces and achieve a satisfactory rate of accuracy in the identification of fatigue.

Key-words: Fatigue. Accidents. Digital image. Image processing. Monitoring of the eyes. Webcam.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

API – *Application Programming Interface*

AMV – Apoio de Máquina Vetor

FPS – *Frames* por Segundo

FRSS – Filtros Retangulares de Seis Segmentos

JMF – *Java Media Framework API*

JUDE – *Java UML Modeling Tool*

RGB – *Red Green Blue*

UML – *Unified Modeling Language*

SUMÁRIO

1 INTRODUÇÃO

Privação de sono e dormir em desordem estão se tornando um problema para os motoristas, numa sociedade em que as pessoas parecem não ter tempo suficiente para realizar todas as atividades que necessitam.

O número de automóveis nas estradas é cada vez maior, desta forma a quantidade de acidentes vem crescendo a cada dia. Pirito (2008, p. 1) afirma que os acidentes em geral são imprevisíveis e difíceis de evitar, e que os acidentes automobilísticos são a terceira principal causa de mortes e ferimentos. Os dois fatores mais reconhecidos como causa dos acidentes de trânsito são a velocidade e o álcool. Entretanto, a desatenção, a fadiga e a sonolência são fatores considerados também como grandes contribuintes.

A fadiga e a sonolência em motoristas são comuns. Estudos mostram que existe um grande número de pessoas que apresentam certo grau de distúrbios do sono. De acordo com a última classificação internacional de sono, sonolência significa a dificuldade em manter um estado de alerta e está dividida em três graus de severidade (PIRITO, 2008, p. 1):

a) sonolência leve: manifesta-se em estado de pausa ou repouso em tarefas que requerem pouca atenção;
b) sonolência moderada: pode comprometer a mais leve atividade física e tarefas que exijam um grau moderado de alerta, como, por exemplo, dirigir;
c) sonolência severa: manifesta-se diariamente, mesmo durante atividades físicas como comer e andar.

Muito tem sido feito nos últimos anos para prover mais segurança no trânsito e nos carros. Porém, também existe o fator humano a ser considerado. Diante do exposto, o presente trabalho propõe a construção de uma ferramenta, mais especificamente, um *software* para ajudar na detecção de casos de fadiga ou início de sonolência. Será utilizada uma câmera de vídeo para monitoramento e análise contínua dos olhos do motorista. No caso de uma detecção de fadiga, será emitido um alerta sonoro.

Estudos relacionados à segurança, itens de segurança e desenvolvimento de novas tecnologias no intuito de ajudar a evitar acidentes são indispensáveis nos dias atuais. Todos gostariam de ter carros mais seguros e mais tranqüilidade no trânsito.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é desenvolver uma ferramenta para detecção de fadiga em motoristas baseada no monitoramento dos olhos utilizando imagens capturadas em uma *Webcam*.

Os objetivos específicos do trabalho são:

a) captar um vídeo de um motorista utilizando uma câmera acoplada a um veículo;
b) detectar e demarcar a face do rosto humano;
c) detectar e demarcar a pupila do olho;
d) perceber alteração no estado dos olhos (abertos ou fechados) para caracterizar um estado de sonolência;
e) alertar o motorista no caso da detecção de um possível estado de sonolência.

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 sono ao se dirigir, fadigas e acidentes, processamento de imagens, visão computacional e sobre o projeto Visage de RESTOM (2006). 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, em seguida a operacionalidade da ferramenta 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. São realizados comentários sobre sono ao se dirigir, a fadiga e os tipos de acidentes que podem ocorrer devido a este problema, também são apresentados técnicas de processamento de imagens, informações sobre visão computacional e o projeto Visage de RESTOM (2006). Por fim, são listados alguns trabalhos correlatos.

2.1 SONO AO DIRIGIR

O sono dá pistas de que está chegando: a pessoa torna-se mais irritada que o normal e há quem fique quieto ou bastante agitado. Outros sintomas de sonolência são distúrbios visuais, bocejos seguidos, dificuldade em se manter alerta e concentrado em tarefas.

É preciso dormir de dois a quatro minutos antes que qualquer lembrança do sono seja possível e a maior parte dos acidentes acontece quando um motorista só dormiu alguns segundos (BONNET; MOORE, 1982, p. 5). Porém um microssono[1] de não mais do que quatro segundos pode ter conseqüências fatais. Segundo Bonnet e Moore (1982, p. 5) neste breve espaço de tempo, um carro a uma velocidade de 88 km por hora percorre mais de 30 metros, praticamente o comprimento de uma quadra de tênis.

Para não sentir sono ao volante, o ideal é dormir de maneira regular e habitual, afirma a Associação Brasileira de Educação de Trânsito (ABETRAN, 2008, p. 1). O próprio ato de dirigir pode ser monótono. Pistas unidirecionais e o conforto do carro ajudam a embalar o motorista. A falta de sono é cumulativa e, dependendo do caso, é preciso até um mês para colocá-lo em dia. Toda vez que o sono é inibido, ao lavar o rosto ou tomar um café, ele volta. A única solução é parar de dirigir.

[1] Episódio com duração mínima de trinta segundos durante o qual não há percepção de estímulos externos. São habitualmente diurnos.

2.2 FADIGA E ACIDENTES

Segundo Levy (2008, p. 1), a fadiga pode ser definida como a impossibilidade de manter uma força ao nível esperado, ou a debilidade para realizar tarefas rotineiras, sendo que o paciente tem a sensação da necessidade de aumentar o esforço continuadamente para manter a mesma força.

Existem vários tipos de fadiga, podendo ser (LEVY, 2008, p. 1):

a) muscular: acontecem em braços ou pernas após exercícios repetitivos, como andar longas distâncias, fazendo com que a perna falhe e também uma sensação de fraqueza. Isso é causado por um bloqueio do impulso nervoso;
b) por falta de condicionamento físico: ocorre quando os músculos são pouco utilizados;
c) causada por distúrbios do sono: acontece pelo fato do corpo estar exausto e não ter o tempo necessário para a recuperação durante o sono;
d) induzida por medicação: utilização de medicamentos que diminuem o nível de potássio nos pacientes.

Papanikolopoulos (1999, p. 1) afirma que há muitos indicadores que a fadiga está por acontecer, alguns dos quais são possíveis de detectar com o uso de uma câmera.

Já no caso de acidentes, Pirito (2008, p. 1) comenta que o acidente de trânsito é uma das principais causas de morte no mundo atualmente, principalmente na faixa etária dos 15 aos 40 anos. A fadiga, sonolência ou desatenção contribuem para o aumento de acidentes com vítimas fatais. Em geral, as pessoas não avaliam a influência que a sonolência e a fadiga extrema exercem sobre o ato de dirigir. A sonolência compromete funções cognitivas, a memória, a concentração e, principalmente, o estado de alerta, implicando em riscos de acidentes. Motoristas com distúrbios do sono correm duas a três vezes mais riscos de se envolverem em acidentes. Quando estes motoristas são tratados, a redução de acidentes é de 70% (ABETRAN, 2008, p. 1).

No Brasil, em 2006, 35.156 pessoas morreram em acidentes de trânsito, segundo o Ambrósio e Gieb (2008, p. 8). Aplicando a média mundial, cerca de 6.600 indivíduos perderam a vida por causa do sono ao volante. Conforme Abetran (2008, p. 1), o risco de acidentes aumenta não no horário de maior trânsito, mas naquele em que o ser humano tem um declínio na temperatura corporal. Isso ocorre entre 12h30 e 14h e das 22h às 6h da manhã, sendo que o período crítico fica entre 3h30 e 5h30. Com a temperatura corporal baixa,

começamos a liberar melatonina, que induz ao sono.

2.3 PROCESSAMENTO DE IMAGENS

O processamento de imagens são técnicas aplicadas a uma imagem, tendo como entrada uma imagem digital a fim de modificá-la, gerando como saída uma imagem melhorada. Pode-se citar como exemplo, a remoção de ruídos ou o melhoramento do contraste da imagem. Segundo Gonzalez e Woods (2000, p. 1), uma das primeiras aplicações de técnicas de processamento de imagens foi o melhoramento de imagens digitalizadas para jornais.

As técnicas de processamento de imagens e suas aplicações vêm crescendo vigorosamente, sendo bastante utilizadas nas áreas de pesquisa espacial e na medicina. Ambas necessitam de métodos capazes de melhorar as informações visuais, tanto para análise, como para interpretação humana.

Quanto a estas técnicas, as de processamento de imagens, associadas com o avanço no poder de processamento dos computadores, permitem obter-se grandes vantagens. Como exemplo, Paciornik (2007, p. 16) afirma que, através da análise digital de uma imagem se realiza medidas milhares de vezes mais rápidas, muito mais acuradas e medidas que seriam impossíveis de se obter manualmente.

2.3.1 Imagem digital

A imagem digital é a materialização de grande parte dos processos da computação gráfica. 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 (GOMEZ; VELHO, 2003, p. 146).

A representação de uma imagem digital é dada por modelos matemáticos, de forma que o armazenamento é obtido através de uma matriz bidimensional, levando-se em consideração o aspecto de representação espacial e de cor.

2.3.2 Segmentação de Imagem

Gonzalez e Woods (2000, p. 295) explicam que a segmentação é um processo no qual se subdivide uma imagem em suas partes ou objetos constituintes e que o nível até a qual esta subdivisão deve ser realizada depende do problema a ser resolvido. Em geral esta é a etapa mais crítica, pois todo o processamento que será realizado na imagem será feito sobre esta região previamente selecionada.

Algoritmos de segmentação dividem-se em dois grupos principais de detecção: descontinuidades e similaridades. Dentre métodos de segmentação por detecção de similaridades existe o limiar. Pode-se dizer que é o método mais utilizado devido à sua natureza intuitiva e simplicidade de implementação (GONZALES; WOODS, 2000, p. 296).

2.3.3 Histogramas e projeção

O histograma de uma imagem fornece informação útil para fazer realce e análise da imagem. É o histograma de cor, que associa a cada intensidade de cor presente na imagem a sua freqüência de ocorrência, ou seja, é quantidade de *pixels* da imagem que utilizam a cor. O histograma de uma imagem revela a distribuição dos níveis de cinza da imagem. É representado por um gráfico que dá o número de *pixels* na imagem para cada nível de cinza.

No caso de projeção, segundo Gonzalez e Woods (2000, p. 197), esta pode ser representada em uma linha vertical, horizontal ou diagonal e é obtida através do número de *pixels* que são projetados nesta linha. São representações compactas de imagens. No entanto, mais de uma imagem pode ter a mesma projeção. Podem também ser usadas em reconhecimento de objetos.

2.4 VISÃO COMPUTACIONAL

Por definição pode-se dizer que visão computacional é uma área da ciência da computação que tem por objetivo modelar o mundo real ou reconhecer objetos, transformando-os em imagens digitais (SONKA; HLAVAC; BOYLE, 1998, p. 17). O

objetivo da área de visão computacional é a determinação de características dos objetos representados em uma imagem.

A palavra visão está relacionada à capacidade que um ser tem de ver e entender o mundo em que habita. A imagem é formada na mente através das organizações físicas, químicas e biológicas dos olhos. A partir da visão, o cérebro realiza diversas funções. Por exemplo, em um movimento para se pegar um copo, a visão tem papel fundamental junto com a coordenação motora. Através da visão guiamos a nossa mão até o copo, encaixando os dedos no seu corpo cilíndrico. Diariamente utilizamos nossa visão para reconhecer objetos, pessoas, calcular distâncias, verificar se há buracos nas ruas, nuvens cinzas no céu, encontrar uma vaga livre no estacionamento e muito mais.

O mesmo se sucede na visão computacional, que ao invés dos olhos, câmeras são utilizadas para se obter imagens digitais. Em cima destas imagens são aplicadas técnicas computacionais para extrair informações desejadas do mundo tridimensional. Estas informações variam muito de natureza e podem ser empregadas para variadas aplicações em diferentes áreas. Na medicina, imagens de órgãos e células são utilizadas nos diagnósticos e tratamento de doenças. Na robótica a contribuição é ampla com identificação de objetos, localização, inspeção e locomoção. Há também aplicações militares para guiar mísseis e veículos através de trajetórias pré-definidas. A astronomia utiliza imagens de satélites e telescópios para identificar a composição química de planetas e criar um modelo da sua superfície, enquanto que a indústria automobilística utiliza imagens digitais em sistemas de inspeção industrial.

Segundo Sonka, Hlavac e Boyle (1998, p. 26), computação gráfica e processamento de imagens são duas áreas correlacionadas à visão computacional. De uma maneira geral, a computação gráfica utiliza informações, como grafos de cena, tipos de materiais, geometria dos objetos, tipo de projeção utilizada para gerar imagens, entre outras. A visão computacional faz o caminho contrário, procurando obter informações de cenas a partir de imagens digitais previamente capturadas. Para isso quase sempre faz uso de métodos de processamento de imagens. O interesse nestes métodos surgiu da necessidade de melhorar a qualidade da informação pictórica[2] para a interpretação humana. Hoje em dia, os métodos de processamentos de imagens são também utilizados em aplicações de visão computacional de maneira que as duas áreas se confundem.

[2] Definição ambígua, fragmentada, imprecisa da cor e do contorno.

2.4.1 Aquisição de imagens

De acordo com Forsyth e Ponce (2003, p. 75), no procedimento de aquisição da imagem, existem dois elementos relevantes que fazem parte de um sistema de visão computacional. O primeiro se refere aos equipamentos que compõem o ambiente (*hardware*), tais como câmeras, computadores e sistemas de iluminação. E o segundo elemento é o programa (*software*) que processa as imagens e gerencia as ações a serem realizadas. Um sistema de aquisição de imagem consiste usualmente de uma câmera CCD (*Charge Coupled Device*), um monitor de vídeo e a placa digitalizadora de vídeo. A câmera coleta a imagem e a envia para a placa digitalizadora. Um *software* específico acessa os dados digitalizados da placa e extrai as informações relevantes naquele instante para serem processadas e utilizadas.

2.4.2 Identificação de objetos

A identificação e extração de objetos de imagens são necessárias em muitos casos. Por exemplo, um ambiente de montagem requer que peças diferentes possam ser identificadas para que uma dada ação subseqüente possa ser realizada. Outro caso é a junção de partes para formar o produto final. Devido a uma variedade de razões, os dados de imagens usados na entrada de um sistema de visão, nem sempre são perfeitos. Os problemas que freqüentemente ocorrem estão relacionados com a oclusão, onde um objeto pode estar parcialmente escondido atrás de outro objeto, ou dois objetos compondo a mesma imagem. Segundo Sonka, Hlavac e Boyle (1998, p. 103), de forma análoga, a perda de informações ou deformações, pode ser ocasionada por ruídos na imagem devido a condições anormais de iluminação, defeitos de digitalização, e de resultados ineficientes de algoritmos de segmentação.

2.4.3 Reconhecimento de padrões

O processamento da imagem é realizado por ambientes computacionais que conseguem operar com as informações obtidas das imagens. A extração de atributos, a exemplo da detecção de borda, constitui-se numa operação que permite definir quais elementos dos objetos nas imagens podem ser separados de outros objetos presentes na

mesma imagem (SONKA; HLAVAC; BOYLE, 1998, p. 93). As técnicas de reconhecimento de padrões tratam da identificação de partes da imagem que possuem semelhanças. Uma grande quantidade de ferramentas matemáticas e computacionais tem sido desenvolvida para permitir que objetos possam ser extraídos e agrupados em classes específicas de informações. Uma boa representação da forma do objeto gera facilidades para que ele seja armazenado, transmitido, comparado, reconhecido ou mesmo entendido. A representação deve ser gerada de acordo com regras simples e precisas. Geralmente uma forma é descrita em termos de número de componentes, primitivas, e relacionamentos entre estes componentes.

2.4.4 Problemas típicos

O problema clássico da visão computacional e do processamento de imagens é determinar se uma imagem contém ou não um dado objeto, uma dada característica ou uma dada atividade. Conforme Forsyth e Ponce (2003, p. 109), tal tarefa pode ser resolvida de forma robusta e sem esforço humano, mas ainda não foi resolvida satisfatoriamente para o caso geral, objetos arbitrários em situações arbitrárias. Os métodos atuais conseguem, no máximo, resolver para objetos específicos, como poliedros, faces humanas, letras escritas à mão ou veículos e também em situações específicas, como iluminação bem definida, fundo fixo e pose dos objetos bem definida.

Alguns exemplos de problemas são descritos a seguir:

a) reconhecimento: uma ou várias classes pré-definidas ou aprendidas de objetos podem ser reconhecidas, geralmente em conjunto com sua posição em imagens bidimensionais ou com sua pose em imagens tridimensionais;
b) identificação: uma instância individual de um objeto pode ser reconhecida, como a identificação de uma face ou de impressão digital, ou até mesmo a identificação de um veículo;
c) detecção: imagem é digitalizada para uma condição específica, como a detecção de células ou tecidos anormais.

2.5 PROJETO VISAGE

O projeto de RESTOM (2006) visa apresentar uma aplicação que é capaz de substituir os tradicionais *mouses* existentes atualmente por uma interface mais moderna, utilizando o rosto humano como uma nova forma de interagir com o computador. Características faciais como ponta do nariz, olhos e sobrancelhas são detectados e monitorados em tempo real, utilizando-se de uma *webcam* para tal. As coordenadas captadas são utilizadas para fazer o controle do *mouse*. Apertos dos botões do *mouse* são feitos através da detecção da piscada do olho, onde o olho esquerdo equivale ao aperto do botão esquerdo e o direito equivale ao botão direito. As seções a seguir são referentes ao projeto Visage (RESTOM ,2006).

2.5.1 Detecção de face

Alguns métodos de detecção de face surgiram em torno do ano de 1970, onde simples heurísticas eram aplicadas às imagens captadas, porém com algumas restrições. Por exemplo, fundo liso, condição de luz especial e vista frontal. Estes métodos, no entanto, melhoraram ao longo do tempo e tornarem-se mais robustos. Apesar do grande número de métodos de detecção facial existentes atualmente, estes podem ser organizados em duas categorias principais: métodos baseados em características e em imagem.

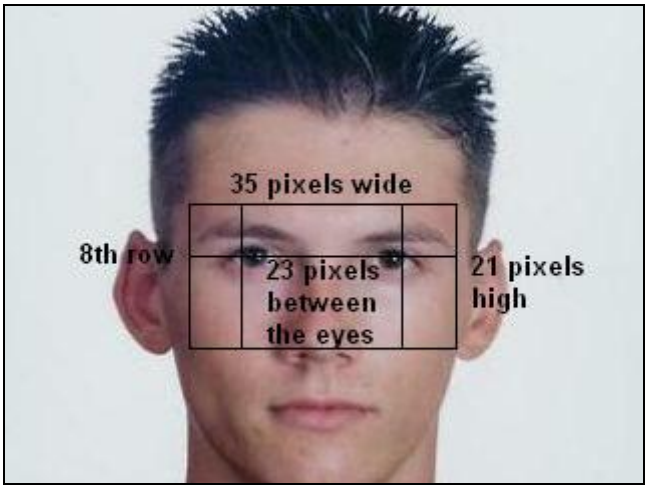
Os métodos baseados em características envolvem encontrar características faciais como, por exemplo, rastros do nariz, olhos, sobrancelhas, lábios, e no fim verificar a sua autenticidade geométrica através de análise das suas posições, tamanhos, e as distâncias de cada um dos outros. Este recurso baseado em características acaba por levar à localização do rosto e as características que ele contém.

Já os métodos baseados em imagem procuram pela região de interesse na imagem com uma área específica onde pode se procurar pela face ou rosto em todas as escalas e localizações possíveis. Esta categoria de detecção de face implica no reconhecimento através de métodos simples, comparando-se a imagem com um modelo correspondente ou com técnicas mais avançadas, tais como redes neurais e Apoio de Máquinas Vetor (AMV, do inglês *Support Vector Machines*).

2.5.2 Apoio de Máquinas Vetor (AMV)

O AMV é um novo tipo de margem máxima de classificação, uma espécie de teoria de aprendizagem. Há um teorema afirmando que para atingir a classificação mínima de erro é necessário separar amostras positivas das amostras negativas. O AMV é uma forma de aprendizado baseada em amostras de dados e classificada por atributos como sendo positivas ou negativas, ou seja, ele verifica as amostras da região entre os olhos treinando essas amostras seguindo as seguintes regras:

a) são extraídas amostras contendo 35 *pixels* de largura com 21 *pixels* de altura, conforme mostra na Figura 1;
b) região da testa e da boca não são consideradas no treinamento das amostras para evitar a influência de diferentes tipos de cabelos, barbas e bigodes;
c) o tamanho das amostras são de 735 *pixels* (35*21);
d) são treinadas no AMV cerca de 6.759 amostras, sendo metade delas exemplos positivos e a outra metade negativos;
e) as amostras são retiradas de diferentes rostos contidos no banco de dados.



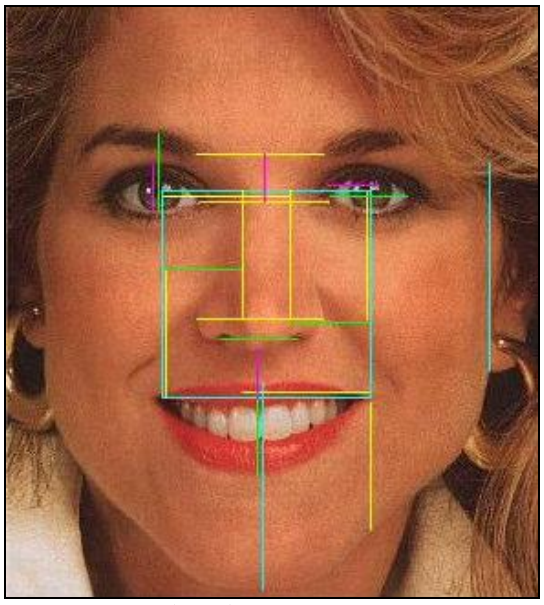
Fonte: Restom (2006).

Figura 1 – Amostra usada no treinamento

2.5.3 Proporções da face humana

O rosto humano é regido pelas proporções que definem os diferentes tamanhos e distâncias entre as suas características faciais. A face é dividida em porções de medidas exatamente iguais, delimitadas por linhas que vão das projeções mais laterais da cabeça aos

cantos internos e externos das pálpebras. A Figura 2 apresenta uma foto mostrando essas proporções onde cada cor na imagem representa uma proporção no rosto humano. Essa divisão ajuda a analisar, através de cálculos, as proporções do rosto humano e é possível usar essas proporções para melhorar a heurística de detecção facial e monitoramento.

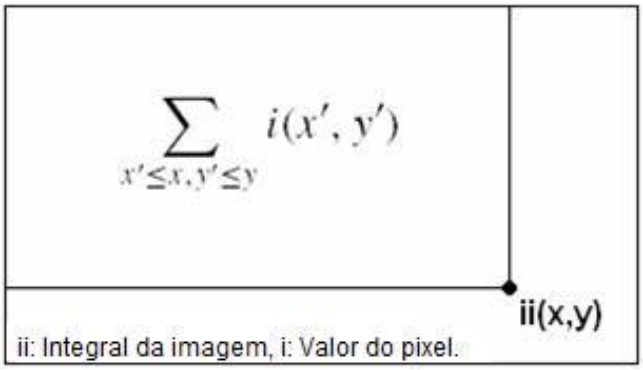


Fonte: Restom (2006).

Figura 2 – Proporções do rosto humano

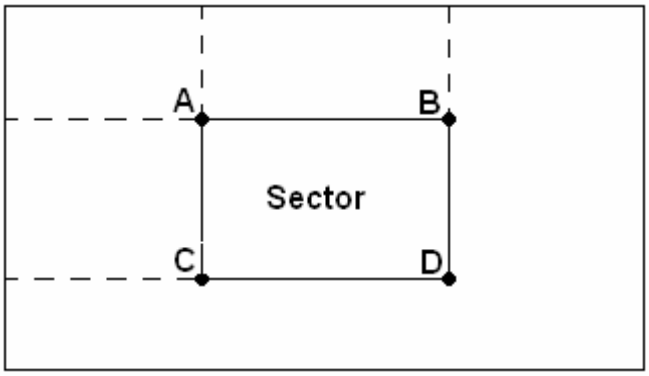
2.5.4 Procura por possíveis rostos

Para a procura por possíveis rostos é utilizado o método baseado em característica, desta forma são reduzidas as áreas de procura do rosto e também o tempo de procura. São utilizados Filtros Retangulares de Seis Segmentos (FRSS, do inglês *Six Segmented Rectangular filters*) que utiliza uma equação chamada de integral da imagem para calcular a as seis regiões do filtro, onde a representação do local da integral da imagem `x,y` contém a soma dos *pixels* que estão acima e a esquerda do *pixel* `x,y` (ver Figura 3). Não importa o tamanho que o setor será, serão necessários apenas 3 operações aritméticas para calcular a soma dos *pixels* que pertencem a ele. Por exemplo, S=D-B-C+A, o filtro precisará de 6*3 operações para calcular (ver Figura 4). Já na Figura 5 é mostrada a localização ideal do FRSS onde considerou o centro como uma possível face humana.



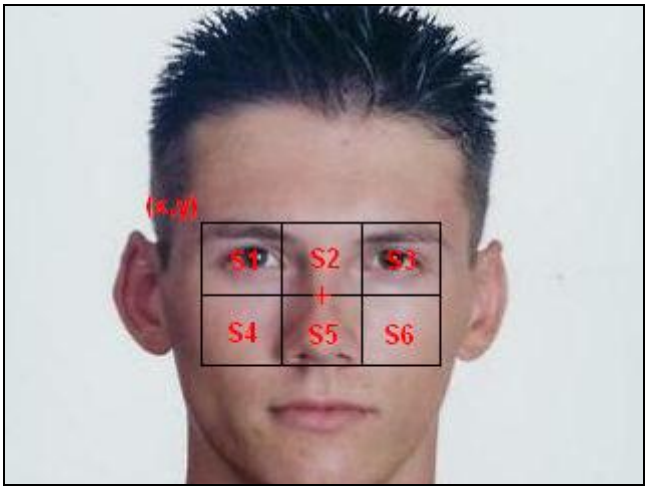
Fonte: Restom (2006).

Figura 3 – Equação integral da imagem



Fonte: Restom (2006).

Figura 4 – Calculando a soma de *pixels* no setor



Fonte: Restom (2006).

Figura 5 – Localização ideal do FRSS

São consideradas a posição dos olhos na parte S1 e S3, já a região entre os olhos como sendo a região S2 e por fim a S5 como sendo o nariz na Figura 5. Porém algumas vezes podem ocorrer casos de erro nesta detecção como mostra a Figura 6 e quando isto ocorre são utilizados grupos de possíveis rostos.

Fonte: Restom (2006).

Figura 6 – Erro na detecção do rosto

2.5.5 Grupos de possíveis rostos

Verificar todos os possíveis rostos seria computacionalmente pesado e desnecessário. O que é feito neste caso é encontrar os grupos de possíveis rostos e considerar o centro de cada grupo como o último candidato, depois utilizar o algoritmo de Grupos, presente no trabalho de RESTOM (2006) para filtrar as possíveis áreas aonde o rosto poderá estar localizado.

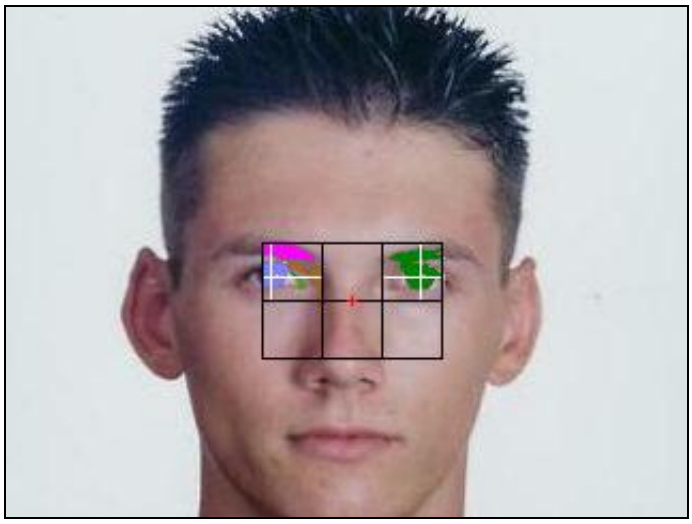
2.5.6 Procura pela pupila do olho

A pupila esquerda e a direita estão localizadas nas regiões S1 e S3 conforme a Figura 5, para cada região são feitas algumas etapas para localizar as pupilas. Por exemplo, a binarização do setor S1 e S3 juntamente com uma taxa de limiarização[3]. Depois de encontrados os setores e já binarizados são seguidos os seguintes passos.

Se o resultado da limiarização produz somente um grupo igual ao setor S3, então é provável que este grupo contenha o olho e também a sobrancelha, conforme mostra a Figura 7. Neste caso é necessário procurar pela pupila um pouco mais abaixo do setor atual. Então é calculada a área da parte do grupo que está na metade inferior do setor, se este for maior que uma determinada taxa de limiarização o centro da parte inferior é considerada como a pupila.

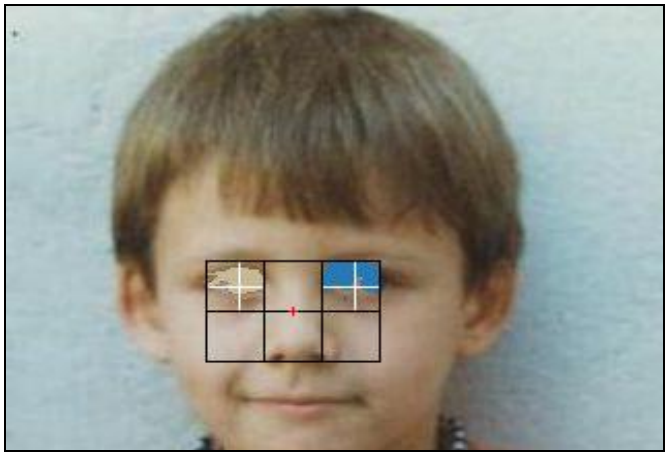
[3] Valor mínimo de alguma quantidade, como exemplo, um dado nível de cinza em uma imagem digital.

Caso não for o mesmo, será aplicado para a metade superior, pois é possível que o grupo que foi encontrado é a grupo da pupila sozinha sem a sobrancelha e ele caiu na metade superior conforme mostra a Figura 8. Em caso de não se encontrar a pupila na metade superior, este setor irá ser desconsiderado e a pupila não é encontrada.



Fonte: Restom (2006).

Figura 7 – Grupo localizado com a sobrancelha



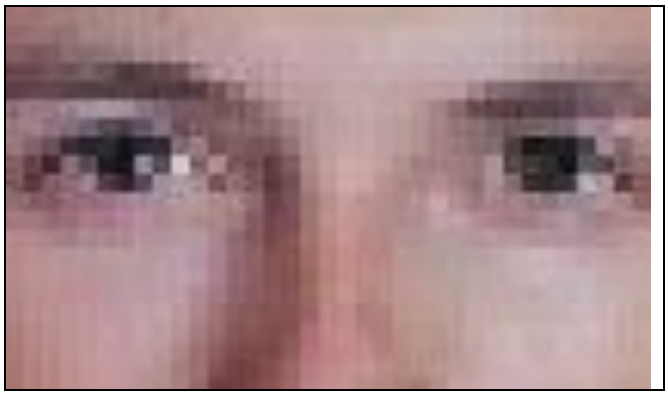
Fonte: Restom (2006).

Figura 8 – Grupo localizado sem a sobrancelha

2.5.7 Extraindo a região entre os olhos

Depois de encontradas as possíveis pupilas, é extraída a região entre os olhos e enviado para o AMV com uma escala de tamanho reduzida, conforme mostra a Figura 9, desta forma é reduzido o tempo de processamento. Para isto é necessário dividir a distância entre a pupila esquerda e a direita por 23, onde 23 é distância entre os olhos e desta forma é obtido à escala necessária. É importante que a região entre os olhos esteja na posição horizontal, caso seja

necessário é feita a rotação na região até que fique nas condições adequadas. Por fim é feita uma classificação pelo AMV a fim de obter melhores resultados e no final do processo são encontrados os olhos. Outros métodos também são utilizados e podem ser verificados no trabalho de RESTOM (2006).



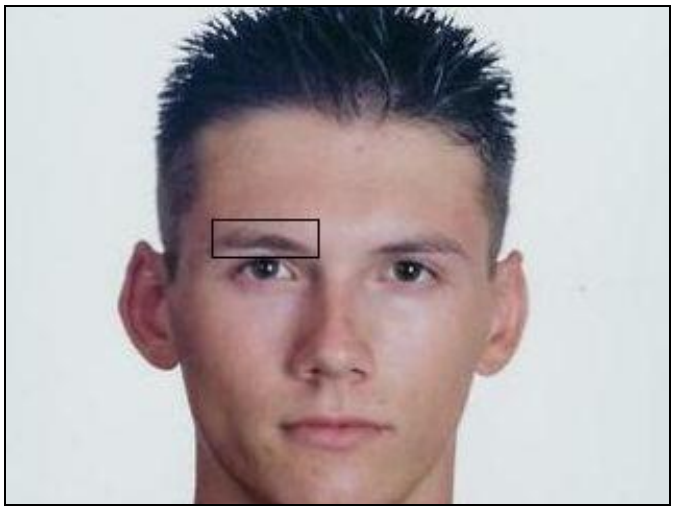
Fonte: Restom (2006).

Figura 9 – Imagem com escala reduzida

2.5.8 Localizando a sobrancelha

Para se detectar a sobrancelha é usada uma pequena região acima da posição que se espera que seja encontrado o olho, conforme mostra a Figura 10. Nesta área é aplicada uma taxa de limiarização, ver Figura 11. Na região acima do olho é encontrada apenas a sobrancelha e um pouco da testa, aplicando a técnica se obtêm alguns pontos que a representam. Caso esta taxa de limiarização não resulte em nenhum ponto, é provável que esta região foi selecionada muito alta, ou seja, acima da sobrancelha e contém somente a testa. Neste caso é necessário utilizar uma região um pouco mais abaixo e repetir todo o processo novamente. Para se encontrar a linha da sobrancelha utilizando os pontos gerados pela técnica anterior é utilizada também a transformada de Hough[4], conforme Figura 12. Algumas vezes a transformada resulta em várias linhas, desta forma é utilizada uma aproximação de onde se suspeita que esteja à linha final que represente a sobrancelha, como pode ser verificado na Figura 13 o resultado final da detecção.

[4] Técnica matemática que realiza a detecção de formas geométricas em imagens digitais.

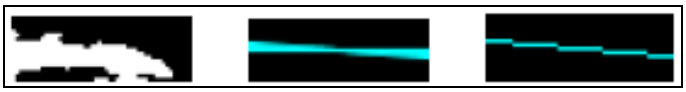


Fonte: Restom (2006).

Figura 10 – Região usada para detectar a sobrancelha

Fonte: Restom (2006).

Figura 11 – Região após o uso da limiarização



Fonte: Restom (2006).

Figura 12 – Região após o uso da transformada de Hough

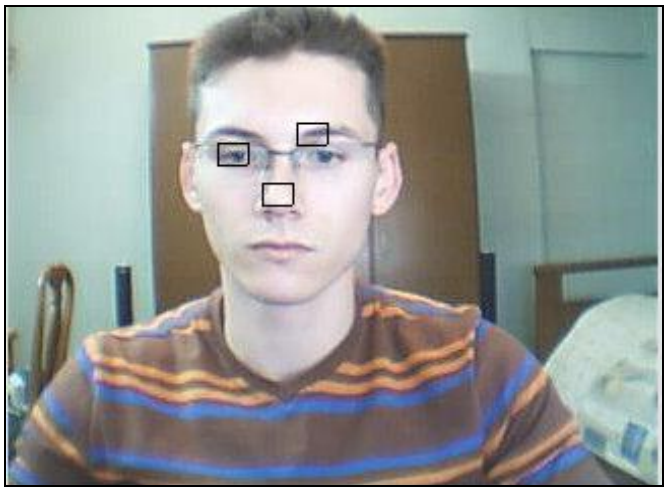
Fonte: Restom (2006).

Figura 13 – Linha da sobrancelha

2.5.9 Monitoramento dos olhos

Monitoramento dos olhos é um pouco diferente da forma como se detecta a região entre os olhos e o nariz. No caso dos olhos estes podem estar se abrindo, se fechando ou até

mesmo piscando no momento da detecção. Para se obter um melhor monitoramento é usada a região entre os olhos como ponto de referência, pois sua distância em relação ao olho é sempre a mesma. Em cada quadro são localizados os olhos e a região entre os olhos para que no quadro seguinte seja comparada sua posição em relação aos olhos novamente e verificar aonde estes poderão estar localizados. Para encontrar a nova região dos olhos são combinados dois métodos. O primeiro é a comparação de imagens, comparando-se uma imagem com a outra através de histograma de imagem e o segundo é a procura por regiões escuras, pois a pupila humana contém uma área mais escura. O problema do segundo método é que este pode detectar a sobrancelha como uma região escura e assumir esta região como sendo a localização do olho, conforme mostra a Figura 14. Por isto antes de calcular a região do olho é calculado a posição da sobrancelha e então é colocada a região de procura sempre abaixo da linha da sobrancelha.



Fonte: Restom (2006).
Figura 14 – Detecção da sobrancelha como sendo o olho

2.6 TRABALHOS CORRELATOS

Dentre os trabalhos pesquisados existem alguns semelhantes ao trabalho proposto, destes foram escolhidos os seguintes: “Construção de uma Ferramenta Voltada à Medicina Preventiva para Diagnosticar Casos de Estrabismo” (MEDEIROS, 2008) e Invisys, “Visão Computacional Aplicada à Análise Automática do Comportamento do Consumidor” (INVISYS, 2007).

2.6.1 Construção de uma ferramenta voltada à medicina preventiva para diagnosticar casos de estrabismo

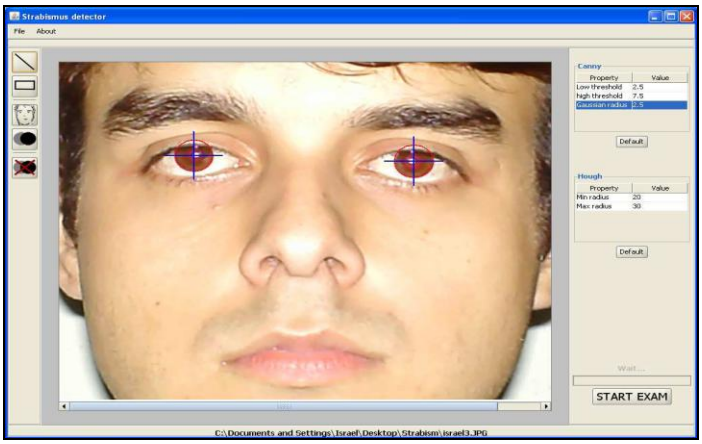
O principal objetivo do trabalho desenvolvido por Medeiros (2008) foi a criação de uma ferramenta voltada à medicina preventiva, mais especificamente o estrabismo, ou seja, é uma ferramenta em forma de *software* que realiza a análise a partir de uma imagem da face humana, capturada por uma câmera digital com *flash*, tendo como base o teste de Hirschberg[5].

São utilizadas *Application Programming Interface*, Java 2D, *Java Advanced Image API* e de técnicas de processamento de imagem digital para o desenvolvimento do projeto. Dentre essas técnicas destacam se o operador de Canny e transformada de Hough, utilizada para a detecção da íris do olho humano e do reflexo ocasionado pelo *flash* da câmera digital.

Com a ferramenta é possível:

a) recuperar uma imagem da face humana através do menu principal;
b) permitir que sejam definidas áreas de interesses para análise da imagem;
c) identificar a íris e o limbo do olho humano na imagem digital;
d) identificar o reflexo do *flash* na córnea ocular na imagem digital;
e) calcular a medida entre o reflexo da córnea e o limbo no sentido nasal e temporal horizontal;
f) calcular o diâmetro da íris;
g) realizar o diagnóstico do estrabismo a partir das medidas estabelecidas;
h) gerar relatório do exame contendo as medidas extraídas;
i) geração do relatório do exame em formato pdf.

A Figura 15 mostra a interface da ferramenta após feito o processo de detecção da íris.



Fonte: Medeiros (2008, p. 59).

Figura 15 - Interface da ferramenta

[5] Visa detectar casos de estrabismo.

2.6.2 Visão computacional aplicada à análise automática do comportamento do consumidor

Segundo Invisys (2007, p. 1) trata-se de uma ferramenta comercial de visão computacional e *marketing* que é capaz de monitorar ambientes.

Invisys (2007, p. 1) afirma que o reconhecimento de face é um método não invasivo onde a pessoa pode nem estar ciente que está sendo identificada e é a métrica mais utilizada pelos humanos para realizar a identificação de uma pessoa. São utilizados algoritmos para extração de características e *matching* de faces de alta performance e baixa complexidade computacional que podem ser integrados em sistemas embarcados e em sistemas baseados em computadores pessoais.

O Invisys utiliza uma ou mais câmeras de vídeo para fazer o monitoramento de pessoas. Já para aplicações de contagem de pessoas, o caminho completo que uma pessoa faz é gravado a partir do momento em que ele aparece até esta sair do campo de visão. A contagem é gerada quando a trajetória de um pedestre cruza uma zona de contagem marcada na imagem.

Além disso, é possível estimar o número de pessoas em uma determinada região, bem como o tempo médio que elas permaneceram. Isso pode ser usado para análise de filas de espera em aeroportos, bancos, caixas de supermercados, entre outros.

São utilizados métodos de detecção de faces e detecção da trajetória de pessoas através de algoritmos proprietários desenvolvido pela própria empresa. A seguir é apresentada a tela principal do produto como mostra a Figura 16.

Fonte: Invisys (2007).

Figura 16 - Interface do produto

3 DESENVOLVIMENTO DA FERRAMENTA

Neste capítulo são detalhadas as etapas do desenvolvimento do projeto. São ilustrados os principais requisitos, a especificação, a implementação e por fim são listados resultados e discussão.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

O presente trabalho deverá:

a) capturar um vídeo do motorista (Requisito Funcional - RF);
b) identificar a face do rosto humano (RF);
c) identificar a região dos olhos (RF);
d) verificar se o motorista está entrando num possível estado de fadiga ou sonolência através do monitoramento dos olhos (RF);
e) emitir um alerta no caso de detecção positiva (RF);
f) disponibilizar uma interface para permitir a visualização das imagens do motorista (RF);
g) implementar a ferramenta utilizando a linguagem Java (Requisito Não Funcional - RNF);
h) utilizar o ambiente de programação Eclipse (RNF).

3.2 ESPECIFICAÇÃO

A especificação do sistema apresentado utiliza alguns dos diagramas da *Unified Modeling Language* (UML) em conjunto com a ferramenta Enterprise Architect 7.0.813 e JUDE Community 5.4.1 para a elaboração dos diagramas de casos de uso, de classes e de sequência. Alguns diagramas estão em sua forma resumida para melhor visualização.

3.2.1 Diagrama de casos de uso

A Figura 17 apresenta o diagrama de casos de uso com as principais interações do usuário com o sistema.

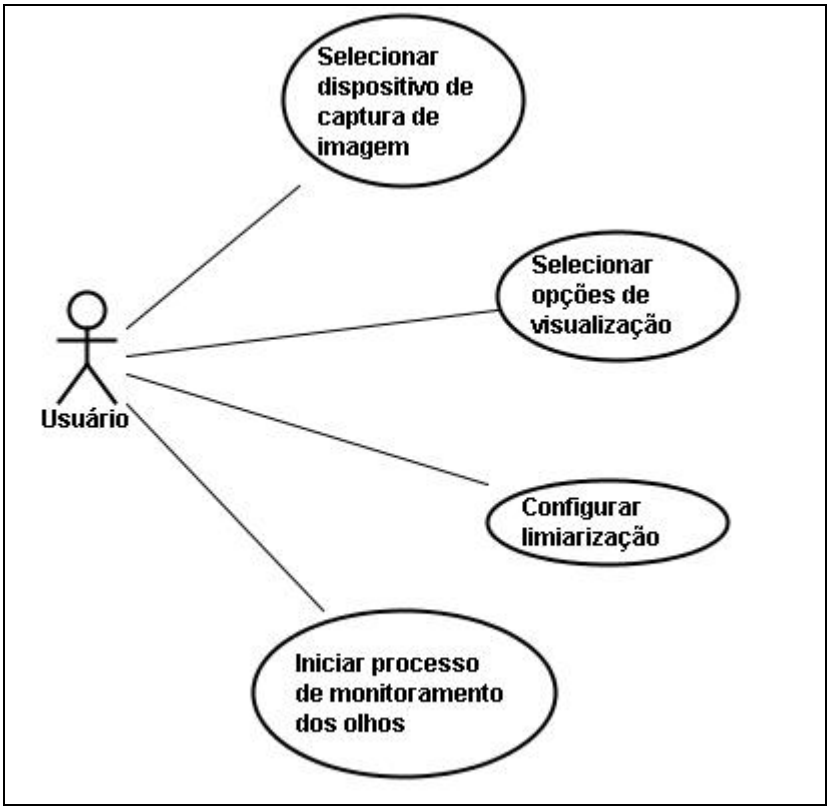


Figura 17 – Diagrama de casos de uso

O caso de uso `Seleciona dispositivo de captura de imagem` (Quadro 1) descreve como o usuário poderá selecionar a *webcam* a ser utilizada. Este caso de uso possui um cenário principal, um alternativo e um de exceção.

Selecionar dispositivo de captura de imagem: possibilita ao usuário selecionar a webcam a ser utilizada.	
Pré-condição	Um dispositivo de captura de imagem deve estar instalado.
Cenário principal	1) O sistema lista os dispositivos de captura de imagem instalados. 2) O usuário seleciona o dispositivo desejado. 3) O sistema valida o dispositivo selecionado.
Fluxo Alternativo 01	No passo 1, caso exista apenas um dispositivo de captura de imagem instalado, este é selecionado automaticamente pelo sistema.
Exceção 01	No passo 1, caso não exista nenhum dispositivo de captura de imagem instalado, o sistema apresenta uma mensagem informando que não foi encontrado nenhum dispositivo instalado.
Pós-condição	O dispositivo é definido com sucesso.

Quadro 1 – Caso de uso seleciona dispositivo de captura de imagem

O segundo caso de uso Seleciona opções de visualização (Quadro 2), explana como o usuário pode determinar quais opções serão mostradas junto na imagem capturada pelo dispositivo de captura de imagem. Além do cenário principal, este caso de uso possui dois fluxos alternativos, o primeiro responsável por informar o que ocorrerá caso o usuário não altere as seleções padrão e o segundo caso ainda não tenha sido iniciado o processo de monitoramento dos olhos.

Seleciona opções de visualização: possibilita ao usuário selecionar as opções de visualizações tendo a possibilidade de deixar as seleções como padrão.	
Pré-condição	Um *streaming* de vídeo deve ser exibido pelo sistema.
Cenário principal	1) O usuário seleciona a opções de visualização. 2) O sistema exibe o resultado junto na imagem capturada pelo dispositivo de captura de imagem.
Fluxo Alternativo 01	No passo 1, caso o usuário não altere as seleções padrões o sistema assume as seleções *default* estabelecidos pelo desenvolvedor.
Fluxo Alternativo 02	No passo 2, caso o processo de monitoramento dos olhos ainda não tenha sido iniciado o sistema apenas armazenas os valores para ser usados após o inicio do processo.
Pós-condição	O sistema exibe os resultados com sucesso.

Quadro 2 – Caso de uso seleciona opções de visualização

O terceiro caso de uso Configura limiarização (Quadro 3), explana como o usuário pode configurar a taxa de limiarização na interface do projeto. Além do cenário principal existe apenas um cenário alternativo, não havendo cenário de exceção.

Configura *limiarização*: possibilita ao usuário configurar as taxas de limiarização tendo a possibilidade de informar os parâmetros de filtragem.	
Pré-condição	Um *streaming* de vídeo deve ser exibido pelo sistema.
Cenário principal	1) O usuário informa os parâmetros de limiarização podendo deixar os valores padrões já estabelecidos pelo desenvolvedor. 2) O sistema utiliza os valores informados.
Fluxo Alternativo 01	No passo 1, caso o usuário não informe os valores da limiarização o sistema assume os valores *default* estabelecidos pelo desenvolvedor.
Pós-condição	O sistema utiliza os valores informados com sucesso.

Quadro 3 – Caso de uso configura limiarização

O quarto e último caso de uso, Inicia processo de monitoramento dos olhos (Quadro 4), trata do momento em que o projeto passa a monitorar o usuário para a detecção de um possível caso de fadiga. Este possui um cenário principal e um alternativo.

Inicia processo de monitoramento: possibilita ao usuário iniciar o processo de monitoramento dos olhos.	
Pré-condição	Um *streaming* de vídeo deve ser exibido pelo sistema.
Cenário principal	1) O usuário clica com o mouse no botão START. 2) O sistema inicia o monitoramento dos olhos.
Fluxo Alternativo 01	No passo 2, caso o sistema não inicie o monitoramento, ele assume que não encontrou os olhos e volta para o passo 1.
Pós-condição	O processo de monitoramento é iniciado com sucesso.

Quadro 4 – Caso de uso inicia processo de monitoramento dos olhos

3.2.2 Diagrama de classe

O diagrama de classes apresentado na Figura 18 exibe as principais classes utilizadas no sistema com os atributos. Algumas classes foram abstraídas para uma melhor visualização do digrama, não comprometendo o entendimento do mesmo.

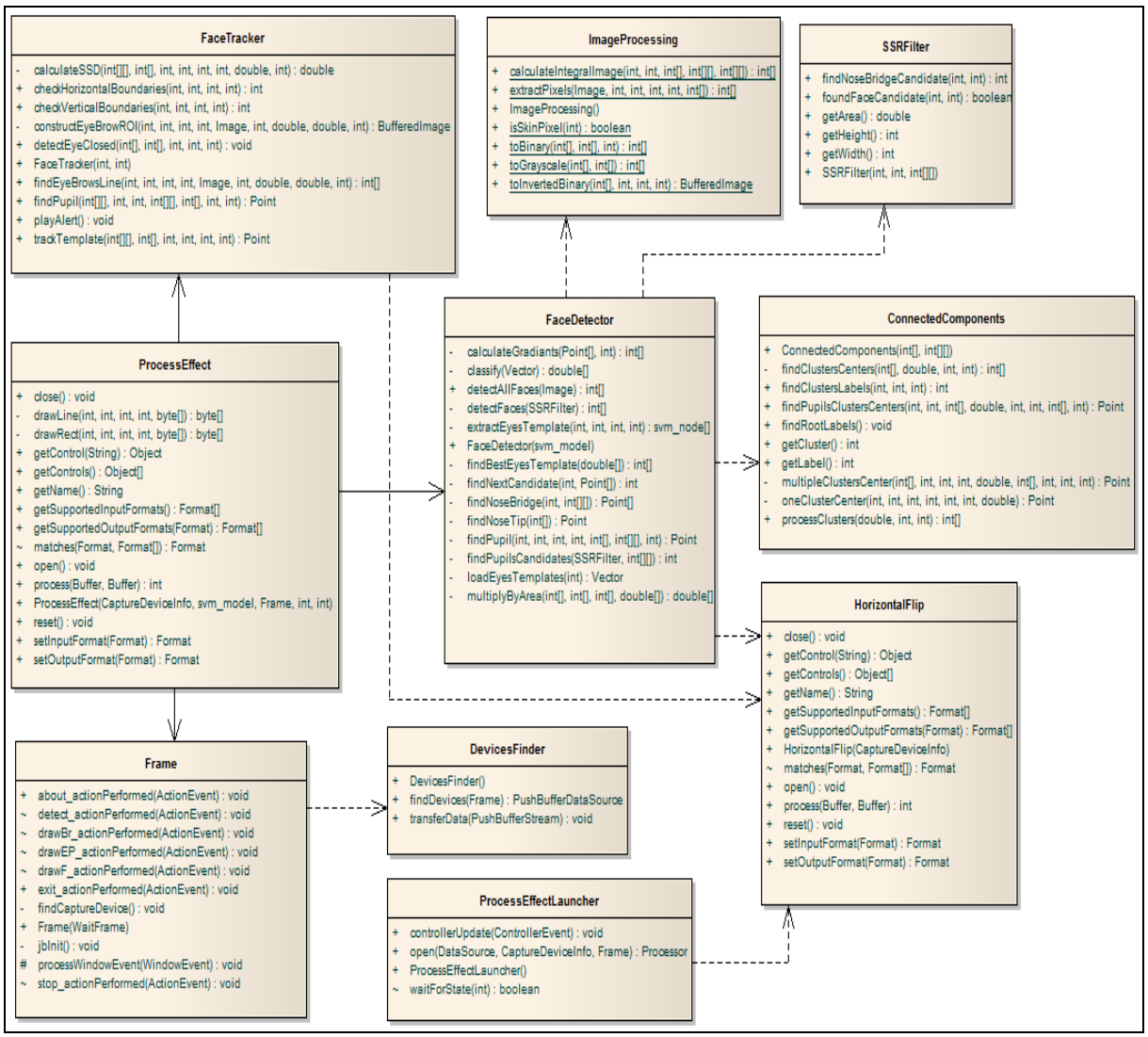


Figura 18 – Diagrama de classes

3.2.2.1 Classe `DevicesFinder`

A classe `DevicesFinder` é responsável pela procura dos dispositivos de captura de imagem tais como *webcams*. Esta classe possui métodos para ler os *frames* gerados pelos dispositivos e identificar os formatos que são suportados por eles. O presente trabalho utiliza o formato de cor RGB e uma taxa de trinta *frames* por segundo (FPS) com uma resolução de 320 por 240 *pixels* para se obter um funcionamento correto do projeto, sendo possível utilizar menos *frames*, porém com resultados não tão precisos. Além disso, a classe contém métodos para se fazer a consistências necessárias para o funcionamento. Pode-se citar como exemplo de validação, caso não seja encontrado nenhum dispositivo de captura ou o formato do vídeo não é adequado, o sistema apresenta uma mensagem para o usuário.

3.2.2.2 Classe `ProcessEffect`

Esta classe processa as imagens obtidas pelo dispositivo de captura com o intuito de realizar todos os processos necessários para a detecção e demarcação do rosto e localização da pupila. Também possui métodos para delimitar as regiões de interesse e demarcá-las, como exemplo temos os métodos `drawRect e drawLine` utilizados para desenhar um retângulo e uma linha nestas regiões. Além disso, é através desta que é feita a chamada do método para a detecção dos olhos fechados que caracteriza o estado de fadiga do motorista.

3.2.2.3 Classe `FaceTracker`

Através da classe `FaceTracker` é realizada a combinação de métodos para se obter um rastreamento do rosto com maior precisão e menor tempo de detecção. É nesta classe que se encontra o método de detecção de olhos fechados (`detectEyeClosed`). Além disso, esta classe também possui o método de reprodução do som de alerta quando ocorre a detecção de fadiga. Outro método presente é o `findEyeBrowsLine` cuja a finalidade é encontrar a linha da sobrancelha através da transformada de Hough (ver seção 2.5.8). Já no método `findPupil` é

feito o processo para se obter a localização da pupila.

3.2.2.4 Classe `FaceDetector`

A classe `FaceDetector` contém métodos para localizar a pupila e o rosto. Um dos métodos utilizados é `findPupil` responsável por localizar a pupila. Outros métodos como o `detectAllFaces` e `detectFaces` cujo o objetivo é detectar o rosto utilizando filtros FRSS (presentes na classe `SSRFilter`) juntamente com um grupo de imagens dos possíveis rostos, conhecidos como *clusters*. Desta forma é feita uma classificação através dos métodos `classify` e `findBestEyesTemplate` para se obter melhores resultados que serão utilizados posteriormente por outras classes.

3.2.2.5 Classes `SSRFilter` e `HorizontalFlip`

A classe `SSRFilter` representa o filtro FRSS onde esta possui o método `foundFaceCandidate` responsável por dividir a imagem em seis partes iguais que representam as partes do rosto, tais como o olho direito e esquerdo, região entre os olhos, nariz e sobrancelhas como pode ser visto na seção 2.5.4. Figura 5.

Já a classe `HorizontalFlip` é responsável por fazer a adequação da imagem, como por exemplo, rotacionar a imagem até que fique na posição horizontal que é usada pelo AMV. Além disso esta classe contém o método `setOutputFormat` cujo objetivo é adequar o formato da imagem para 320 *pixels* de largura por 240 *pixels* de altura, o tamanho utilizado pelo projeto.

3.2.2.6 Classes `Frame` e `ProcessEffectLauncher`

A classe `Frame` contém os componentes responsáveis pela interface da aplicação de modo que o usuário possa interagir com os recursos presentes na ferramenta. Esta classe

possui componentes do tipo `JPanel`, utilizados para a visualização dos resultados obtidos pela ferramenta. Também possui componentes do tipo `JSpinner` e `JCheckBox`, onde são configurados os parâmetros da ferramenta, entre outros.

Através da classe `ProcessEffectLauncher` é feito o controle do *stream* de vídeo capturado pelo dispositivo de captura, como por exemplo, o sincronismo e a adequação do formato.

3.2.2.7 `ImageProcessing` e `ConnectedComponents`

Na classe `ImageProcessing` estão presentes os métodos de processamento de imagem da ferramenta, como o `extractPixels` cuja função é extrair as informações presentes em uma determinada posição da imagem. Também contém o método `toGrayscale` e `toBinary` responsáveis por transformar a imagem em escala de cinza ou binarizada.

Já a classe `ConnectedComponents` possui métodos para demarcar quais os locais de procura pelos possíveis rostos serão armazenados no *cluster*, a fim de evitar o processamento desnecessário com a procura de todos os locais encontrados pelo filtro FRSS.

3.2.3 Diagrama de sequência

A Figura 19 mostra o diagrama de seqüência, que permite visualizar o caso de uso `Inicia processo de monitoramento dos olhos` de forma resumida para uma melhor visualização. A partir do momento em que o usuário aperta o botão `OK`, o processo de detecção é iniciado, o método `findCaptureDevice` é executado fazendo a busca por dispositivos instalados. Em seguida a classe `Frame` executa o método `process` passando os parâmetros do tipo *Buffer* para a classe `ProcessEffect`. Logo após os `checkHorizontalBoundaries` e `checkVerticalBoundaries` são disparados definindo dessa forma uma área de limite aonde será procurada a pupila ou as sobrancelhas, em seguida o `trackTemplate` é executado com os parâmetros necessários.

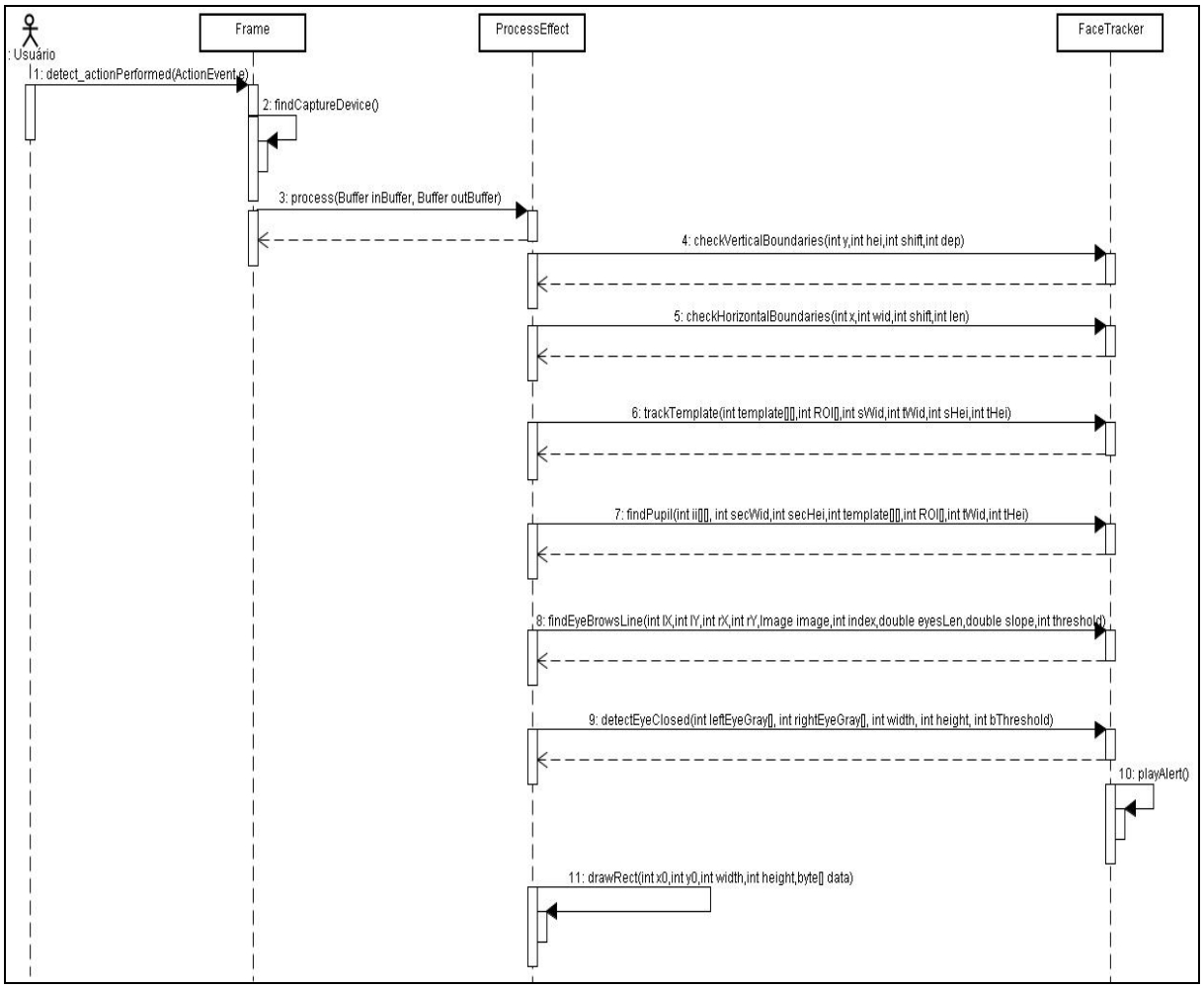


Figura 19 – Diagrama de seqüência para o caso de uso `Inicia processo de monitoramento dos olhos`

Os próximos métodos chamados são `findPupil` e `findEyeBrowsLine`, são disparados na tentativa de localizar a pupila e a sobrancelha, respectivamente. Após estes serem localizados, o `detectEyeClosed` faz o processo de verificação se os olhos estão fechados, caracterizando um estado de fadiga e em caso positivo o `playAlert` é acionado reproduzindo um sinal sonoro na tentativa de alertar o usuário. Por fim, é chamado o método de desenho `drawRect` responsável por desenhar os retângulos nas regiões de interesses como a pupila e rosto.

3.3 IMPLEMENTAÇÃO

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

3.3.1 Técnicas e ferramentas utilizadas

Para a implementação da ferramenta foi utilizada a linguagem de programação Java juntamente com *Java Media Framework API* (JMF). O ambiente de desenvolvimento escolhido foi o Eclipse SDK Version: 3.4.0. A biblioteca Visage V3.2 e classes utilizadas no projeto foram obtidas do trabalho de RESTOM (2006).

3.3.1.1 Detecção dos olhos

A detecção dos olhos através de um dispositivo de captura é realizada principalmente pelas classes `FaceTracker` e `FaceDetector` que foram modificadas do trabalho de RESTOM (2006). Nestas classes encontram-se os métodos necessários para a localização da pupila, como o `findPupil`, que pode ser observado o seu código fonte no Quadro 5.

```
  private  Point findPupil(int x0,int y0,int width,int height,int labels[],int
pupilsMembers[][],int limit)
  {
    int label = 0;
    for (int y = 0; y < height; y++)
      for (int x = 0; x < width; x++)
        pupilsMembers[x][y] = 0;
    ConnectedComponents CC = new ConnectedComponents(labels,pupilsMembers);
    for (int y = 0; y < height; y++)
      for (int x = 0; x < width; x++)
        if( binaryPixels[ (y + y0) * fWidth + (x + x0)] == 1 )
          label = CC.findClustersLabels(x, y, width);
    if( label != 0 )
    {
      CC.findRootLabels();
      return
CC.findPupilsClustersCenters(x0,y0,CC.clusters,limit,width,height,grayPixels,fWidth);
    }
    else{
      return null; } }
```

Quadro 5 – Código fonte do método `findPupil`

São passados sete parâmetros necessários para a localização do ponto que representa a pupila, entre eles o tamanho e largura que será feita a procura. É realizada a binarização do setor e os resultados encontrados são armazenados em um *cluster* para serem classificados pelo método `findPupilsCandidates` presente nesta classe (`FaceTracker`), obtendo-se assim os melhores resultados na detecção da pupila.

No Quadro 6 encontra-se o código fonte do método `findPupilsCandidates` que é proveniente da classe `FaceTracker`. São passados como parâmetros os filtros `SSRFilter` descritos na seção 3.2.2.5. Em todos os possíveis rostos armazenados no *cluster* é feita a

procura da pupila esquerda no setor S1 e da pupila direita no setor S3 (mais detalhes na seção 3.3.1.4).

```
  private int findPupilsCandidates(SSRFilter filter,int centers[][])
  {
    eyes = new int[cluster][5];
    int width = filter.getWidth()/3;
    int height = filter.getHeight()/2;
    int labels[] = new int[width*height];
    int pupilsMembers[][] = new int[width][height];
    int limit = (int)filter.getArea()/(144*6);
    int halfW, halfH, sixthW, counter = -1;
    halfW = filter.getWidth() / 2;
    halfH = filter.getHeight() / 2;
    sixthW = filter.getWidth() / 6;
    Point lp1,rp1;
    int x, y;
    for (int i = 1; centers[i][0] != Integer.MAX_VALUE; i++)//pass all
clusters
    {
      x = centers[i][0];
      y = centers[i][1];
      //find left pupil in sector s1
      lp1 = findPupil(x-halfW,y-
halfH,width,height,labels,pupilsMembers,limit);
      if( lp1 == null )
        continue;

      //find right pupil in sector s3
      rp1 = findPupil(x+sixthW,y-
halfH,width,height,labels,pupilsMembers,limit);
      if( rp1 == null )
        continue;

      counter++;
      eyes[counter][0] = (int)lp1.getX()+x-halfW;
      eyes[counter][1] = (int)lp1.getY()+y-halfH;
      eyes[counter][2] = (int)rp1.getX()+x+sixthW;
      eyes[counter][3] = (int)rp1.getY()+y-halfH;
      eyes[counter][4] = i;
    }

    return counter;
  }
```

Quadro 6 – Código fonte do método `findPupilsCandidates`

3.3.1.2 Detecção da sobrancelha

A detecção da sobrancelha é realizada pela classe `FaceTracker` que foi modificada do trabalho de RESTOM (2006), que tem como método principal o `findEyeBrowsLine` que pode ser visto no Quadro 7.

```
  public int[] findEyeBrowsLine(int lX,int lY,int rX,int rY,Image
image,int index,double eyesLen,double slope,int threshold)
  {
    int coords[] = new int[6];
    int sx = 0,sy = 0,ex = 0,ey = 0;
    boolean found;
    BufferedImage ROI =
constructEyeBrowROI(lX,lY,rX,rY,image,index,eyesLen,slope,threshold);
    if( ROI == null )
      return null;
    BufferedImage temp = new
BufferedImage(ROI.getWidth(),ROI.getHeight(),BufferedImage.TYPE_INT_RGB);
    hough.run(ROI);
    ROI = hough.getSuperimposed(temp,1d);
    found = false;
    for(int x=0 ; x<ROI.getWidth()/2 && !found ; x++)
      for(int y=0 ; y<ROI.getHeight() ; y++)
        if(ROI.getRGB(x,y) != Color.black.getRGB())
        {
          sx = x; sy = y;
          found = true;
          break;
        }
    if( !found )
      return null;
    found = false;
    for(int x=ROI.getWidth()-1 ; x>=ROI.getWidth()/2 && !found ; x--)
      for(int y=ROI.getHeight()-1 ; y>=0 ; y--)
        if(ROI.getRGB(x,y) != Color.black.getRGB())
        {
          ex = x; ey = y;
          found = true;
          break;
        }
    if( !found )
      return null;
    coords[0] = sx; coords[1] = sy; coords[2] = ex; coords[3] = ey;
    if( index == 0 )
    {
      coords[4] = this.slx; coords[5] = this.sly;
    }
    else
    {
      coords[4] = this.srx;
      coords[5] = this.sry;
    }
    return coords;
  }
}
```

Quadro 7 – Código fonte do método `findEyeBrowsLine`

São necessários os métodos de limiarização e a transformada de Hough (seção 2.5.8) para encontrar a linha da sobrancelha. Se a região de interesse contém apenas a sobrancelha a limiarização irá trazer os pontos que representam as sobrancelhas. Caso não seja encontrado nenhum ponto, é provável que a região foi colocada muito alta, trazendo apenas a testa. Desta forma é repetido o processo com uma região de interesse mais abaixo, este processo é feito mais de duas vezes para cada sobrancelha a fim de garantir um melhor resultado. Já na

transformada de Hough, esta às vezes pode trazer mais de uma linha, neste caso é desenhada a linha que mais se aproxima do resultado final.

3.3.1.3 Detecção de olhos fechados

O código fonte do método `detectEyeClosed` pode ser visto no Quadro 8 e sua explicação em seguida.

```
  Public void detectEyeClosed(int leftEyeGray[], int rightEyeGray[], int
width, int height, int bThreshold)
  {
      int binaryPixelsL[] = new int[width * height];
      int binaryPixelsR[] = new int[width * height];
      int countL = 0; int countR = 0;
      binaryPixelsL =
ImageProcessing.toBinary(leftEyeGray,binaryPixelsL,bThreshold);
      binaryPixelsR =
ImageProcessing.toBinary(rightEyeGray,binaryPixelsR,bThreshold);

      for (int y = 0; y < height; y++)
            for (int x = 0; x < width; x++){
                  if(binaryPixelsL[y * width + x] == 0){
                        countL++;
                  }
                  if(binaryPixelsR[y * width + x] == 0){
                        countR++;
                  }
            }
      if (System.currentTimeMillis() - lt > (int) (200)){
            lt = System.currentTimeMillis();
            if ((countL > 14) || (countR > 14)){
                  countAuxL++;
                  if(countAuxL > 3){
                        textStatus = ("ALERT... FATIGUE DETECTED !!!");
                        playAlert();
                  }
            }else{
                  countAuxL =0;
                  textStatus = ("STATUS... NORMAL !!!");
            }
      }
}
```

Quadro 8 – Código fonte do método `detectEyeClosed`

Para a detecção dos olhos fechados utilizado na detecção da fadiga foi adicionado o método `detectEyeClosed` na classe `FaceTracker` que foi aproveitada do trabalho de RESTOM (2006). É passado como parâmetro uma matriz de 4x4 *pixels* na forma de *grayscale*, que representa o olho esquerdo e o direito, também é passado uma taxa de limiarização. Com essa matriz é realizado o processo de binarização, logo em seguida é feito a contagem de *pixels* pretos na imagem, caso esta matriz possuir uma quantidade maior que

quatorze *pixels* (mais detalhes na seção 3.4), então é armazenado num contador e se este contador atingir o valor maior que três, é considerado como um caso de fadiga e o método `playAlert` que reproduz um som é chamado para alertar o motorista.

3.3.1.4 Filtro FRSS

O filtro FRSS presente na classe `SSRFilter` aproveitada do trabalho de RESTOM (2006) contém o método `foundFaceCandidate` onde pode ser visto o seu código fonte no Quadro 9. É responsável por diminuir o tempo de procura do rosto, utiliza o cálculo da integral da imagem (ver seção 2.5.4) a fim de separar a imagem em seis setores e classificá-los por regiões, sendo a região S1 o olho esquerdo, S2 a região entre os olhos, S3 o olho direito e S5 o nariz.

```
  public boolean foundFaceCandidate(int x,int y)
  {
    int s1,s2,s3,s4,s6;

    //Calculate sectors
    s1 = ii[x+sectorWidth][y+sectorHeight] - ii[x+sectorWidth][y] -
ii[x][y+sectorHeight] + ii[x][y];
    s2 = ii[x+2*sectorWidth][y+sectorHeight] - ii[x+2*sectorWidth][y] -
ii[x+sectorWidth][y+sectorHeight] + ii[x+sectorWidth][y];
    s3 = ii[x+3*sectorWidth][y+sectorHeight] - ii[x+3*sectorWidth][y] -
ii[x+2*sectorWidth][y+sectorHeight] + ii[x+2*sectorWidth][y];
    s4 = ii[x+sectorWidth][y+2*sectorHeight] -
ii[x+sectorWidth][y+sectorHeight] - ii[x][y+2*sectorHeight] +
ii[x][y+sectorHeight];
    s6 = ii[x+3*sectorWidth][y+2*sectorHeight] -
ii[x+3*sectorWidth][y+sectorHeight] -
ii[x+2*sectorWidth][y+2*sectorHeight] +
ii[x+2*sectorWidth][y+sectorHeight];

    //Face candidate conditions
    if( (s1<s2) && (s1<s4) && (s3<s2) && (s3<s6) )
      return true;
    else
      return false;
  }
```

Quadro 9 – Código fonte do método `foundFaceCandidate`

3.3.1.5 Desenho retangular

O desenho retangular utilizado para desenhar as regiões de interesse como os olhos e o rosto está presente na classe `ProcessEffect` aproveitada do trabalho de RESTOM (2006). O

método `drawRect` responsável por desenhar quatro linhas, sendo duas horizontais e duas verticais, necessárias para o desenho do retângulo. É passado como parâmetros o local inicial de `x`, `y` e a partir deste local quantos *pixels* para `x` e `y` de distância serão desenhados no retângulo. Uma parte do código fonte do método `drawRect` pode ser visto no Quadro 10.

```
    private byte[] drawRect(int x0,int y0,int width,int height,byte[]
data)
    {
      int pos;
      //draw first horizontal line
      for(int x=x0 ; x<x0+width ; x++)
      {
        pos = (240-y0)*960+(x)*3;
        if( (pos>=0) && (pos+2<data.length) )
        {
          data[pos] = blue;
          pos++;
          data[pos] = green;
          pos++;
          data[pos] = red;
        }
        else
          break;
      }
      //draw second horizontal line
...
      }
      //draw first vertical line
...
      }
      //draw second vertical line
      for(int y=y0 ; y<y0+height ; y++)
      {
        pos = (240-y)*960+(x0+width)*3;
        if( (pos>=0) && (pos+2<data.length) )
        {
          data[pos] = blue;
          pos++;
          data[pos] = green;
          pos++;
          data[pos] = red;
        }
        else
          break;
      }
      return data;
    }
```

Quadro 10 – Código fonte do método `drawRect`

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 as principais funcionalidades da ferramenta.

3.3.2.1 Escolhendo um dispositivo de captura de imagem

Ao se iniciar o projeto, caso o usuário possuir mais de um dispositivo de captura de imagem instalado no seu computador, a primeira tela que é apresentada para o usuário é a `tela selecionar dispositivo de captura de imagem` como pode ser visto na Figura 20.

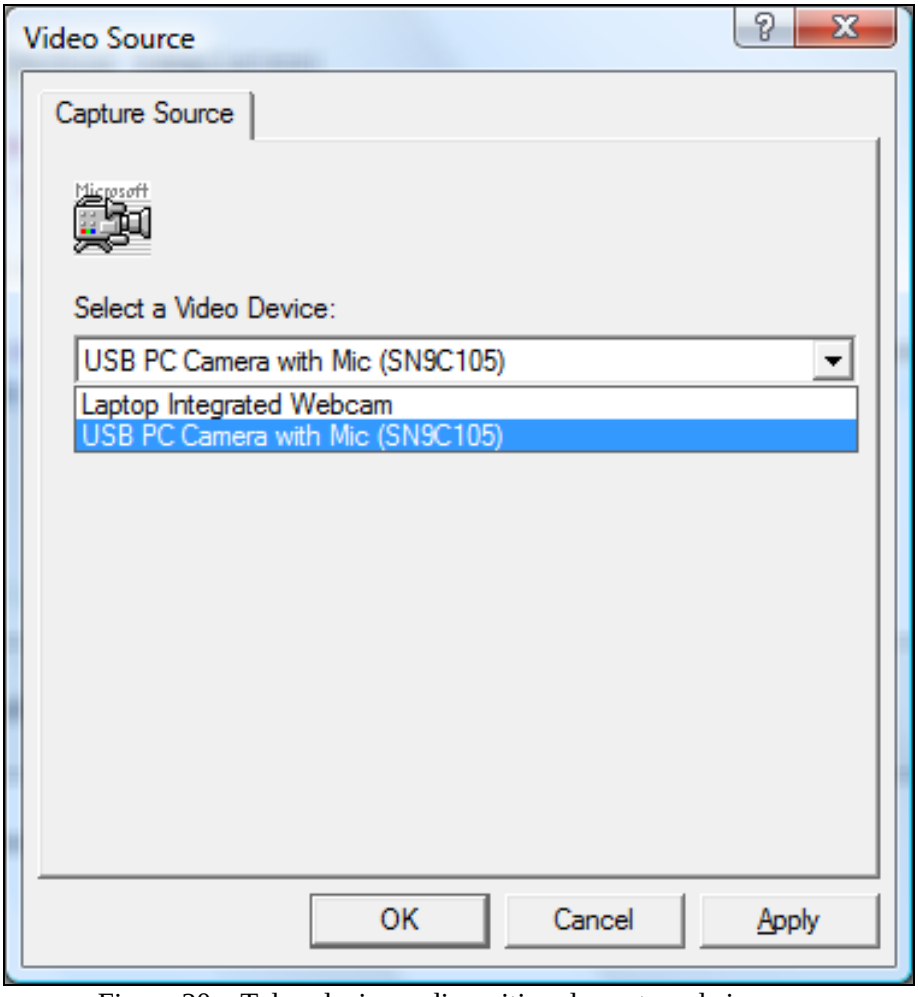


Figura 20 – Tela selecionar dispositivo de captura de imagem

Ao se selecionar o botão `OK` na interface, ou se o usuário possuir somente um dispositivo de captura de imagem instalado no seu computador. O sistema seleciona este

dispositivo e se tudo ocorrer normalmente deverá ser apresentado à tela principal do projeto, conforme mostra a Figura 21 contendo a imagem do usuário.

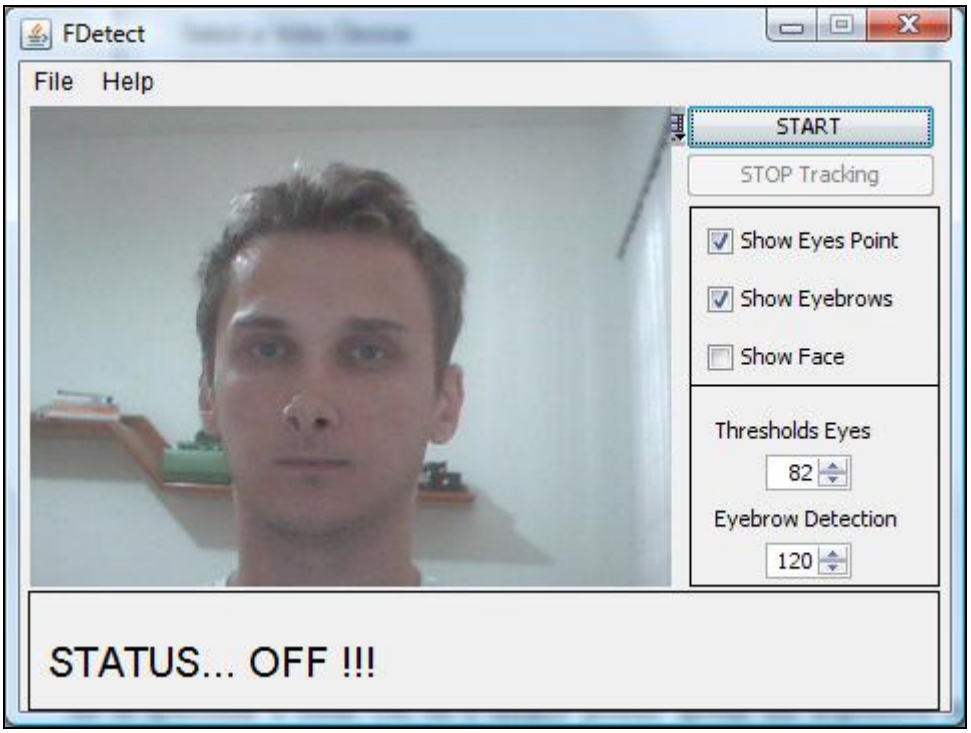


Figura 21 – Tela principal do projeto

Caso nenhum dispositivo for encontrado ou o dispositivo não for adequado para o funcionamento correto da ferramenta o sistema apresenta uma mensagem informando ao usuário, como pode ser visto na Figura 22.

Figura 22 – Tela de informação de nenhum dispositivo encontrado

3.3.2.2 Recursos da ferramenta

A ferramenta disponibiliza na sua interface dois botões, um sendo o `START` e outro o `STOP Tracking` (Figura 23), utilizados para iniciar e finalizar o processo de detecção. Três `JCheckBox` (Figura 24), onde se escolhe o que deseja ser mostrado na tela para o usuário.

Figura 23 – Botões da interface

Figura 24 – `JCheckBox` da interface

Dois `JSpinner` (Figura 25), usados para definir os valores da limiarização que será usado na detecção dos olhos e da sobrancelha. Um menu (Figura 26) `File` onde pode-se fechar o programa e o `Help` onde obtém-se informações a respeito da versão e o nome do autor do projeto.

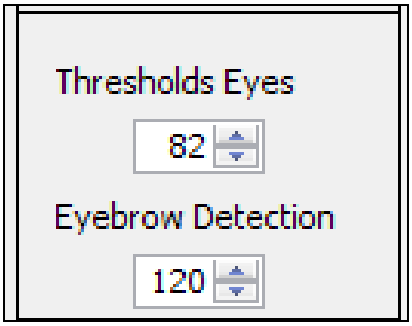


Figura 25 – `JSpinner` da interface

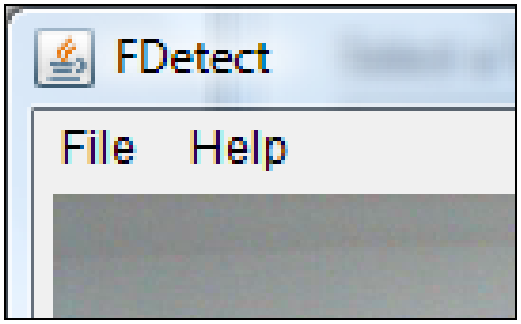


Figura 26 – Menu da interface

Uma área onde é mostrado o *status* da ferramenta (Figura 27) e por fim um `JPanel` (Figura 28) com dimensões de 320 por 240 *pixels* utilizado para mostrar a imagem que está sendo captada pelo dispositivo e os seus resultados.

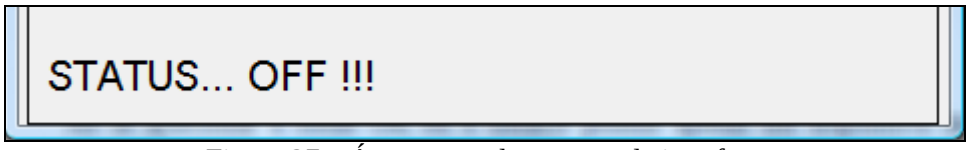


Figura 27 – Área contendo o *status* da interface

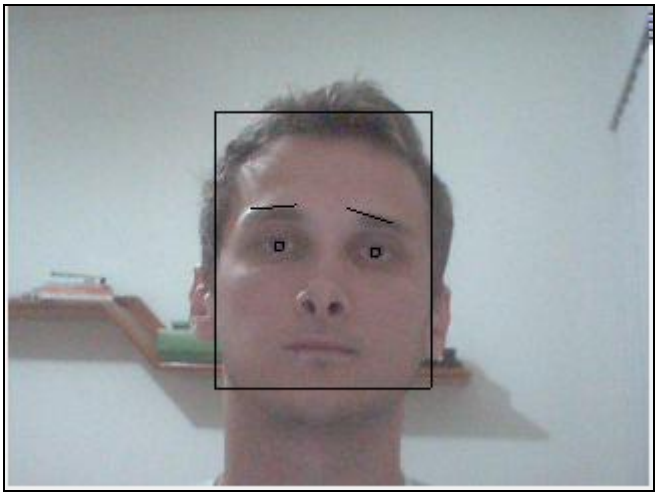


Figura 28 – `JPanel` da interface mostrando os resultados da detecção

3.3.2.3 Funcionamento da ferramenta

Após a tela principal (Figura 21) ser apresentada para o usuário, este pode iniciar o processo de detecção, para isto deve-se apertar o botão `START` na interface do projeto, desta maneira serão utilizados os valores padrões especificados pelo desenvolvedor. Caso não seja iniciado o processo de detecção deve-se observar as condições de iluminação ou quantidades de *frames* por segundo disponíveis pelo dispositivo de captura de maneira que estas estejam dentro das especificações mínimas fornecidas pelo desenvolvedor (mais detalhes, ver seção 3.4).

Após a detecção ser iniciada corretamente o sistema passa a monitorar constantemente os olhos do usuário a fim de detectar um possível estado de fadiga. Caso este estado se torne presente o sistema emite um aviso sonoro na tentativa de alertá-lo conforme mostra a Figura 29.

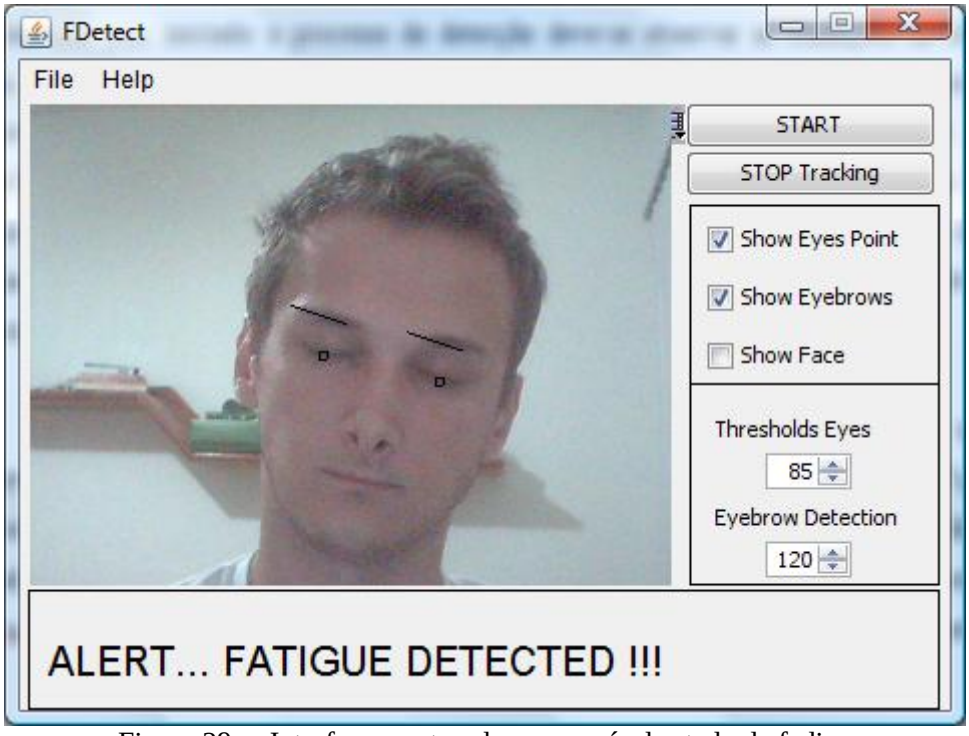


Figura 29 – Interface mostrando um possível estado de fadiga

Também é possível configurar quais informações irão aparecer na tela, tais como detecção dos olhos, sobrancelhas ou o rosto, utilizando-se das opções do menu (Figura 24). Pode-se ainda alterar os valores da limiarização (Figura 25) caso a detecção dos olhos ou de fadiga não esteja adequada. Para cessar o rastreamento pode-se apertar o botão `STOP Tracking` (Figura 23), desta forma o processo de detecção é interrompido.

3.4 RESULTADOS E DISCUSSÃO

No presente trabalho temos o desenvolvimento de uma ferramenta capaz de monitorar os olhos do motorista a fim de detectar um possível estado de fadiga. As classes presentes neste trabalho foram reutilizadas do projeto Visage (RESTOM, 2006). Algumas classes, como por exemplo: `DevicesFinder`, `HorizontalFlip`, `Preview` e `WaitFrame` não receberam alterações, mantendo seus formatos originais. As demais classes presente no trabalho receberam de alguma forma certas modificações. A classe `Frame`, teve que ser reformulada a fim de se adequar a interface da ferramenta, algumas classes receberam novos métodos, como no caso da classe `FaceTracker`, onde foi adicionado os métodos `detectEyeClosed` e `playAlert`. Já outras classes, como por exemplo, na `ProcessEffect`, optou-se por remover alguns dos seus métodos, pois não seriam mais necessários para o projeto em questão. Também foram adicionado a esta classe, a chamada do método `faceTracker .detectEyeClosed` e a criação de novas variáveis, como por exemplo, `leftEyeGray[]` e `rightEyeGray[]`, necessárias para o funcionamento da ferramenta. As demais classes tiveram seus métodos ou atributos modificados afim de uma melhor adequação ao objetivo proposto.

Ao se iniciar o projeto notou-se que a velocidade de processamento era um dos fatores críticos a ser tratado. Nos testes foram utilizados um notebook com processador Core2Duo Intel de 2.0Ghz e uma *webcam* LG modelo LIC-300 (WEBPRO2, 2009).

Após a conclusão da ferramenta iniciou-se a fase de testes de desempenho. Utilizando a função `System.currentTimeMillis()` nativa do Java, pode-se observar que o tempo necessário para realizar todo o processo de detecção e monitoramento foi realizado em menos de 175 milissegundos, tornando o projeto viável para o objetivo proposto pelo trabalho.

Outro fator a ser explorado nos testes foram as condições de luminosidade e posicionamento da luz no ambiente. Para os testes foi utilizada uma luminária de mesa Decorlux modelo LM-9260 (DECORLUX, 2009), com uma lâmpada modelo PL0096-9W / 6500K / F07 (EMPALUX, 2009). Um fluxo luminoso[6] acima de 550lm é fundamental para o funcionamento correto da ferramenta, abaixo desses valores a detecção se torna imprecisa e difícil de ocorrer. Quanto à questão do posicionamento da luz, a Figura 30 mostra a visão de cima e os posicionamentos da luz que foram utilizadas nos testes.

[6] Radiação total emitida em todas as direções por uma fonte luminosa ou fonte de luz que pode produzir estímulo visual cuja unidade de medida é o lumen (lm).

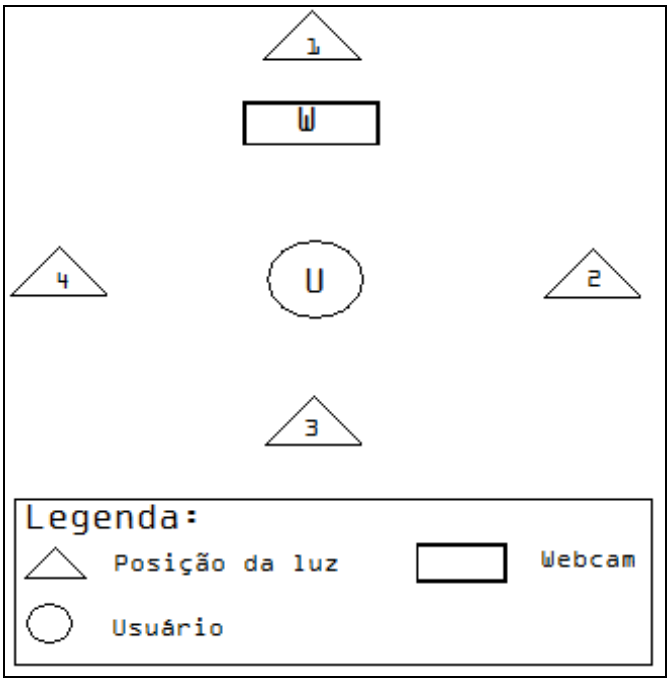


Figura 30 – Posicionamento da luz

Com a luz posicionada a frente do usuário e representada pelo triângulo número 1 na Figura 30, nota-se que foi o que obteve o melhor desempenho nos testes realizados. Já com a luz posicionada nas laterais do usuário, representada pelos triângulos número 2 e 4, houve uma perda bastante significativa na capacidade de detecção, influenciado pelo posicionamento da luz. Quando a luz foi posicionada na parte de trás do usuário (triângulo 3 na Figura 30) a detecção teve um desempenho mediano, devido a falta de luminosidade necessária para o funcionamento da ferramenta. O comparativo do teste pode ser visto no Quadro 11 que mostra a porcentagem de acerto na detecção dos olhos dependendo do posicionamento da luz no ambiente.

Posição da luz número	**Taxa limiarização olhos**	**Taxa limiarização sobrancelha**	**Porcentagem de acerto na detecção**
1 (Frente)	93	120	100%
2 (Lateral)	93	120	30%
3 (Atrás)	93	120	60%
4 (Lateral)	93	120	30%

Quadro 11 – Resultados do teste de posicionamento da luz

Para a detecção dos olhos fechados foram criados testes através de um protótipo a fim de definir qual o tamanho ideal a ser utilizado na matriz que representa os olhos. Como

resultado final foi utilizado uma matriz de 4x4 *pixels*, pois uma matriz maior observou-se que o tempo necessário para percorrer todos os *pixels* era maior e os resultados eram iguais aos que se obteve utilizando uma matriz menor. Desta forma pode-se reduzir a matriz e o tempo necessário para realizar o processo de detecção.

Foi criado o método `detectEyeClosed` no qual faz a contagem de *pixels* pretos presentes na matriz após realizado o processo de binarização desta. Quando se obtém um número maior do que quatorze *pixels* pretos na matriz, assume-se que o usuário está com os olhos fechados e desta forma podemos detectar um estado de fadiga no motorista. Para se chegar ao valor de quatorze *pixels* pretos, foram realizados testes no qual se pode observar que após feita a configuração correta da taxa de limiarização, o usuário fechava os olhos e se obtinha uma matriz com uma quantidade de *pixels* maior do que quando se estava com os olhos abertos. Em outras palavras, quando se tem um valor acima de quatorze *pixels* preto o usuário está com os olhos fechados e abaixo deste valor os olhos estão abertos. A contagem de *pixels* pode ser vista no Quadro 12.

Caso de teste	**Taxa limiarização olhos**	**Taxa limiarização sobrancelha**	**Quantidade de pixels pretos**	**Estado dos olhos**
Teste 1	93	120	10	Aberto
Teste 2	93	120	9	Aberto
Teste 3	93	120	7	Aberto
Teste 4	93	120	16	Fechado
Teste 5	93	120	15	Fechado
Teste 6	93	120	16	Fechado

Quadro 12 – Resultados da contagem de *pixels* pretos

Para que o processo de monitoramento dos olhos seja realizado de forma mais eficiente, o ângulo máximo que o usuário pode atingir com sua cabeça é de 30 graus para os lados, considerando 0 graus o ângulo em relação ao usuário e o dispositivo de captura. Outro fator a ser considerado é taxa de FPS gerados pelo dispositivo de captura, onde este deve ser maior do que 20.

Com o objetivo de testar o comportamento da ferramenta com diversas pessoas, foi organizado um grupo de oito pessoas com diferentes características, tais como: cabelo

comprido e curto, indivíduos com pela clara e mais escura, indivíduos do sexo masculino e feminino, entre outras características únicas de cada pessoa. Os testes foram realizados no interior de um veículo Gol GV (VOLKSWAGEN, 2009) durante o dia e consistem em 10 tentativas de detecção dos olhos em cada usuário. As porcentagens de acerto nestes testes podem ser vistos na Quadro 13.

Usuário	**Características**	**Sexo do usuário**	**Porcentagem de acerto na detecção**
Usuário 1	Pele negra	Masculino	70%
Usuário 2	Pele clara	Masculino	40%
Usuário 3	Cabelos compridos	Feminino	80%
Usuário 4	Pele escura	Feminino	60%
Usuário 5	Um pouco de barba	Masculino	70%
Usuário 6	Rosto mais cheio	Masculino	80%
Usuário 7	Rosto mais fino	Masculino	80%
Usuário 8	Provida de óculo	Feminino	60%

Quadro 13 – Resultados da detecção em um grupo de pessoas

Após realizados os testes, pode-se observar que o usuário número um, com cor de pele negra, obteve uma taxa de acerto significativa, tendo atingido 70% das tentativas de detecção. O usuário número dois, com pele mais clara e cabelos curtos, foi o que obteve o pior resultado, tendo acertado apenas 40% das tentativas. Cerca de três usuários (3, 6 e 7) atingiram uma taxa de acerto muito alta, em torno de 80%. Já com o usuário número oito a ferramenta mostrou-se capaz de detectar os olhos em usuários do sexo feminino, mesmo estes estando providos de óculos de grau. Neste caso, os óculos possuem lentes bem grandes e anti-reflexivas, também possui uma armação muito fina, quase imperceptível, tornando viável a detecção através da ferramenta.

Já no caso da Figura 31, esta mostra o teste sendo realizado com um indivíduo do sexo masculino utilizando óculos com armação mais larga e na cor preta, onde neste caso, não foi possível fazer a detecção corretamente do usuário devido à armação dos óculos estarem ocultando a linha da sobrancelha, dificultando o processo de detecção.

Figura 31 – Usuário com óculo de armação larga

Os demais resultados das detecções realizados em vários indivíduos podem ser vistos na Figura 32 e Figura 33. Cada linha nas imagens é representada por um usuário diferente após feito o processo de detecção dos olhos e das sobrancelhas. Outros testes gravados em vídeos e realizados com pessoas e horários diferentes dos testes já apresentados neste trabalho podem ser vistos em (VÍDEOS, 2009).

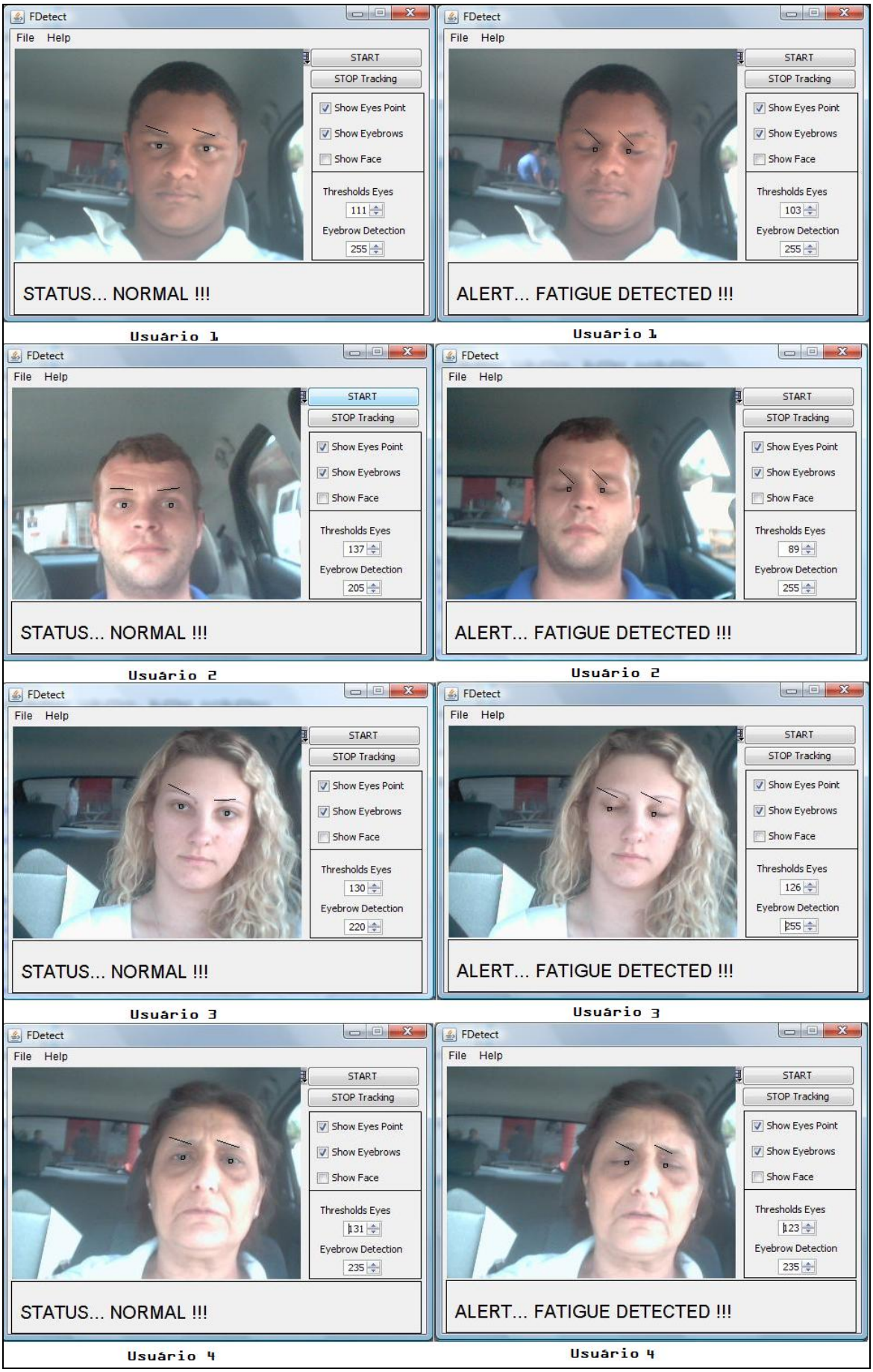


Figura 32 – Resultados dos testes de detecção em diferentes usuários parte 1

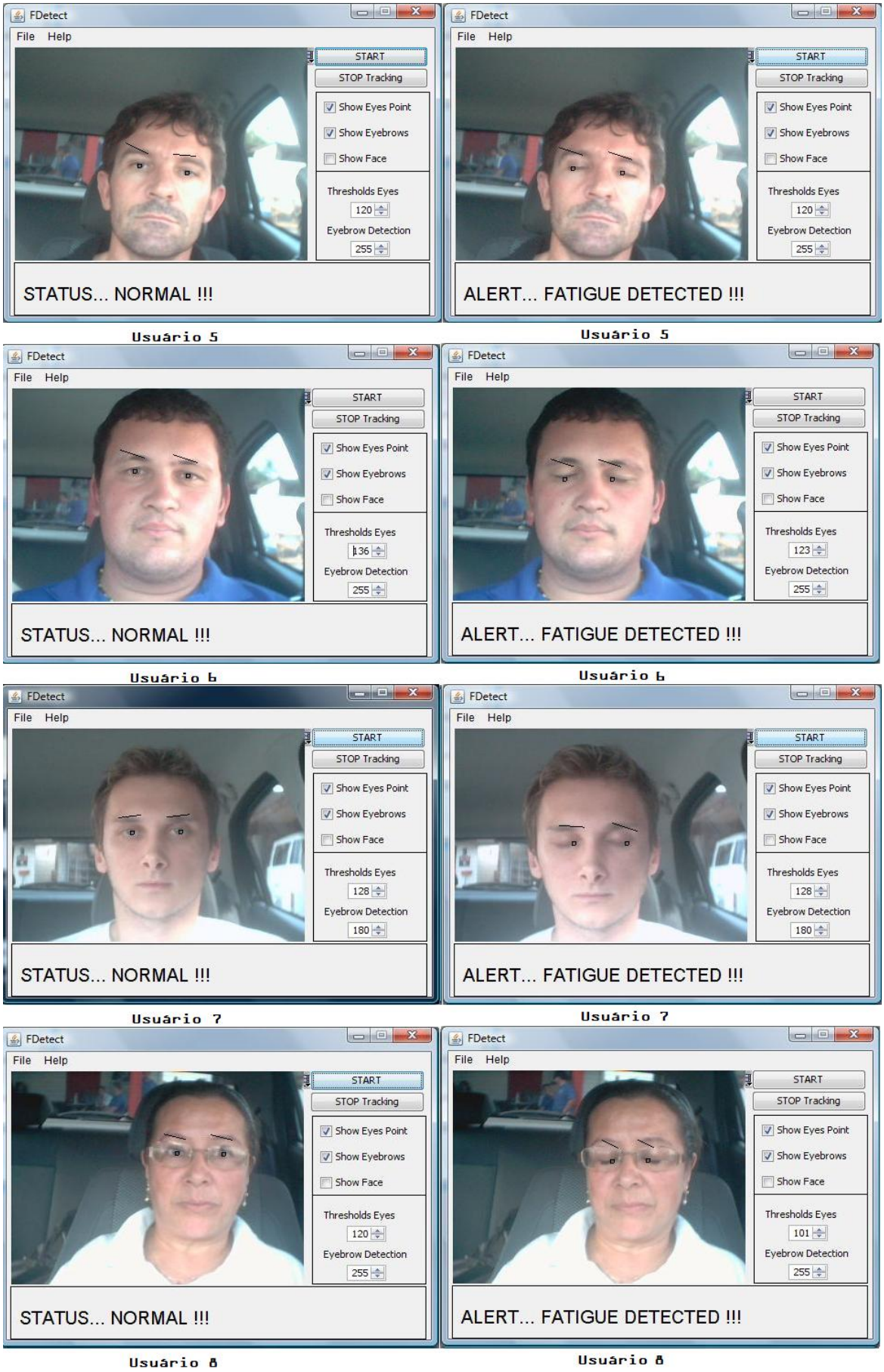


Figura 33 – Resultados dos testes de detecção em diferentes usuários parte 2

4 CONCLUSÕES

O presente trabalho possibilitou desenvolver uma ferramenta capaz de monitorar o motorista de forma segura e de baixo custo a fim de prevenir acidentes ocorridos por um dos principais fatores causadores de acidentes com automóveis, a fadiga. Para o funcionamento utilizou-se um dispositivo de captura de imagem, como por exemplo, uma *webcam* já presente na maioria dos computadores atuais.

A implementação da ferramenta teve como base o trabalho de RESTOM (2006). Métodos como os de limiarização e transformada de Hough foram cruciais para a detecção do rosto e da pupila. A utilização do filtro FRSS e de heurística presente no AMV mostraram-se de grande importância para otimização do processo de detecção, tornando viável a detecção e monitoramento da pupila em tempo real. Outros métodos como, por exemplo, a rotação da imagem e diminuição da escala também contribuíram para um melhor desempenho da ferramenta.

A utilização da biblioteca e classes presentes no trabalho de RESTOM (2006) juntamente com a linguagem de programação Java, que possui vários métodos e rotinas de exemplo bem documentadas foi de grande ajuda para a elaboração da ferramenta de forma que muitos métodos puderam ser reutilizados, poupando tempo e tornando o projeto viável.

O uso do conjunto de ferramentas Architect 7.0.813 e JUDE Community 5.4.1 para a elaboração das especificações do trabalho se mostraram ágeis e fáceis de usar, não apresentando problemas durante sua execução, atendendo desta forma a todas as necessidades do projeto.

A ferramenta se mostrou eficaz quando as condições mínimas de operação são atendidas, entre elas o ângulo do rosto não apresentar um grau de rotação maior que trinta graus, e o fato de o motorista não estiver usando óculos ou outro acessório que dificulte o rastreamento do rosto. A distância entre o dispositivo de captura e o motorista deve estar próxima de trinta e cinco centímetros e a taxa de FPS deve ser igual ou maior que vinte. A fonte de luz deve estar posicionada na parte da frente do motorista e deve ser maior do que 550lm. Porém, nem sempre estas condições são atingidas, tornando a ferramenta vulnerável a possíveis erros de detecção e falsos alertas. Não é uma taxa de 100% de acerto, existe muita influência do ambiente, mais testes devem ser feitos para ajustar melhor os parâmetros da ferramenta ou mesmo aprimorar os algoritmos utilizados.

4.1 EXTENSÕES

A ferramenta presente no seguinte trabalho possibilita a continuação do projeto de forma a melhorar sua precisão. Mais testes podem ser realizados para o controle da luz ambiente através do uso de aparelhos específicos para a medição da luminosidade do ambiente.

Já que uma das grandes dificuldades no projeto é o controle da luz no ambiente, como sugestão, a eliminação da necessidade da presença de luz seria de grande impacto, possibilitando a ferramenta funcionar à noite ou em qualquer situação. Com a diminuição dos custos e melhoria dos dispositivos de captura de imagem o uso de câmeras com infravermelho ou visão noturna se tornaria viável.

Além disso, o projeto pode ser melhorado a fim de detectar de forma mais precisa o rosto e a pupila do motorista, mesmo que este esteja provido de acessórios do tipo: óculos, chapéus, bonés, entre outros.

Outro fator que pode ser melhorado é a ausência da necessidade do usuário em configurar a taxa de limiarização fazendo que este processo seja realizado de forma implícita, desta forma facilitando a sua futura utilização diretamente no interior de veículos.

REFERÊNCIAS BIBLIOGRÁFICAS

ABETRAN. **Sono ao volante**. [S.l.], 2008. Disponível em: <http://abetran.org.br/index.php?option=com_content&task=view&id=2865&Itemid=2>. Acesso em: 16 set. 2008.

ALECRIM, E. **Redes neurais artificiais**. [S.l.], 2004. Disponível em: <http://www.infowester.com/redesneurais.php>. Acesso em: 03 set. 2008.

AMBRÓSIO, P.; GEIB, L. T. C. Sonolência excessiva diurna em condutores de ambulância da macrorregião norte do estado do Rio Grande do Sul, Brasil. **Epidemiologia e Serviços de Saúde**, Brasília, v. 17, n. 1, p. 21-31, mar. 2008. Disponível em: <http://portal.saude.gov.br/portal/arquivos/pdf/2artigo_sonolencia_diurna.pdf>. Acesso em: 14 set. 2008.

BOONET, M.; MOORE, S. **The threshold of sleep:** perception of sleep as a function of time asleep and auditory threshold. [S.l.], 1982. Disponível em: <http://www.cosic.org/caffeine-and-behaviour/sleep>. Acesso em: 14 set. 2008.

DECORLUX. [S.l.], 2009. Disponível em: <http://www.decorlux.com.br/newsite/?page_id=235&preview=true>. Acesso em: 30 abr. 2009.

EMPALUX. [S.l.], 2009. Disponível em: <http://www.empalux.com.br/dadostecnicos/index.htm#f3>. Acesso em: 30 abr. 2009.

FORSYTH, A. D.; PONCE, J. **Computer vision, a modern approach**. Texas: Prentice Hall, 2003.

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

GONZALEZ, R. C.; WOODS, R. E. **Processamento de imagens digitais.** Tradução de Roberto Marcondes Cesar Junior, Luciano da Fontoura Costa. São Paulo: Edgard Blücher, 2000.

INVISYS. **Visão computacional aplicada à análise automática do comportamento do consumidor**. [S.l.], 2007. Disponível em: <http://www.invisys.com.br/vcbi.html>. Acesso em: 03 set. 2008.

LEVY, A. J. **Fadiga**. [S.l.], 2008. Disponível em: <http://www.emedix.com.br/doe/neu012_1f_fadiga.php>. Acesso em: 31 ago. 2008.

MEDEIROS, I. D. **Construção de uma ferramenta voltada à medicina preventiva para diagnosticar casos de estrabismo**. 2008. 66 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

PACIORNIK, S. **Processamento digital de imagens**. [Rio de Janeiro], 2007. Disponível em: <http://www.dcmm.puc-rio.br/cursos/ipdi>. Acesso em: 31 ago. 2008.

PAPANIKOLOPOULOS, N. P. **Vision-based detection of driver fatigue**. [Minneapolis], 1999. Disponível em: <http://www-users.cs.umn.edu/~sasingh/research.htm>. Acesso em: 03 set. 2008.

PIRITO, A. M. **Sonolência e acidentes de trânsito**. [S.l.], 2008. Disponível em: <http://www.wlad.com.br/sites/cipa/artigos/sonoacitransito.html>. Acesso em: 01 set. 2008.

RESTOM, A. **Visage, using the face as a mouse**. [S.l.], 2006. Disponível em: <http://www.mirrorservice.org/sites/download.sourceforge.net/pub/sourceforge/v/vi/visage-hci/>. Acesso em: 19 jan. 2009.

SONKA, M.; HLAVAC, V.; BOYLE, R. **Image processing analysis and machine vision**. Boston: PWS, 1998.

VÍDEOS. [S.l.], 2009. Disponível em: <http://www.inf.furb.br/~rafaeld/>. Acesso em: 30 abr. 2009.

VOLKSWAGEN. [S.l.], 2009. Disponível em: <http://www.volkswagen.com/br/pt/modelos/novo_gol.html>. Acesso em: 30 abr. 2009.

WEBPRO2. [S.l.], 2009. Disponível em: <http://compare.buscape.com.br/prod_ficha?idu=90590>. Acesso em: 30 abr. 2009.