UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

SOFTWARE DE DETECÇÃO DE ESTRABISMO PARA DISPOSITIVOS MÓVEIS

RENATO MARTINS LORENZI

BLUMENAU
2014

2014/1-20

RENATO MARTINS LORENZI

SOFTWARE DE DETECÇÃO DE ESTRABISMO PARA DISPOSITIVOS MÓVEIS

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 Ciência da Computação — Bacharelado.

Prof. Marcel Hugo, Mestre – Orientador

BLUMENAU
2014

2014/1-20

SOFTWARE DE DETECÇÃO DE ESTRABISMO PARA DISPOSITIVOS MÓVEIS

Por

RENATO MARTINS LORENZI

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

Presidente:	Prof. Marcel Hugo, Mestre – Orientador, FURB
Membro:	Prof. Aurélio Faustino Hoppe, Mestre – FURB
Membro:	Prof. Dalton Solano dos Reis, M.Sc. – FURB

Blumenau, dia 9 de Julho de 2014

Dedico este trabalho a minha família, principalmente a minha esposa Ellen pela paciência e compreensão em todas as horas passadas em frente ao computador para realização do trabalho. Aos amigos que ajudaram direta e indiretamente na realização deste.

AGRADECIMENTOS

A Deus pela força nos momentos difíceis.

À minha família pela compreensão.

À minha esposa que serviu como principal paciente para testes.

Aos meus amigos, pela força e apoio com amostras de suas faces para os testes.

Ao meu orientador pelo apoio durante todo o desenvolvimento do trabalho.

Que os vossos esforços desafiem as impossibilidades, lembrai-vos de que as grandes coisas do homem foram conquistadas do que parecia impossível.

Charles Chaplin

RESUMO

No presente trabalho são apresentadas as técnicas, bem como a especificação e implementação de um software de detecção de estrabismo para dispositivos móveis, especificamente para a plataforma Android. O estrabismo é uma patologia que afeta o paralelismo entre os dois olhos, que apontam para direções diferentes; podendo surgir nos primeiros meses de vida e nos adultos por diferentes razões (VARELLA, 2014). Para efetuar a detecção desta patologia, um dos métodos utilizados é o teste de Hirshberg, que no presente trabalho foi implementado a execução para a detecção do estrabismo. Para tal, foram utilizadas técnicas de processamento de imagem. Como técnicas principais pode-se destacar o classificador em cascata Haar e a transformada de Hough, aplicada a detecção de círculos. O classificador em cascata Haar foi usado para detectar a região da face e dos olhos e a transformada teve como objetivo detectar a região do limbo e do brilho dos olhos. Os resultados obtidos pela metodologia proposta obtiveram 85.71% de acerto nos pacientes analisados.

Palavras-chave: Estrabismo. Teste de Hirshberg. Transformada de Hough. Processamento de imagens. Classificador em cascata Haar.

ABSTRACT

This paper presents techniques as well as software specification and implementation to detect strabismus for mobile devices, in particular for the Android platform. Strabismus is a condition that affects the parallelism between the two eyes that point in different directions; may arise in babies adults for different reasons (VARELLA, 2014). To make the detection of this disease, one of the methods used is to Hirshberg test, which in this work the implementation for the detection of strabismus. To achieve this goal, image processing techniques are used. As main techniques were used the Haar feature-based cascade classifier and Hough transform. The Haar feature-based cascade classifier was used to detect the face region and eyes and Hough transform aimed to detect the region of the limbus and brightness of the eyes. The results obtained by the proposed methodology obtained 85.71% of accuracy in the analyzed patients.

Key-words: Strabismus. Hirshberg test. Hough transform. Image processing. Haar feature-based cascade classifier.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SIGLAS

API – *Application Programming Interface*

IDE - *Integrated Development Environment*

OpenCV - *Open Source Computer Vision Library*

SUMÁRIO

1 INTRODUÇÃO

Estrabismo é uma anomalia dos olhos que faz com que eles percam o paralelismo entre si e é chamado popularmente de vesgueira. A verdade sobre o estrabismo é que ele não é simplesmente um problema estético - atrás do desvio dos olhos existem problemas sensoriais e cerebrais na área da visão. Pode estar presente no início da vida ou ainda na infância. Pode também aparecer casos em adultos, geralmente causado por alguma doença física não ocular, como diabete e doenças neurológicas, ou devido a um traumatismo na cabeça (DIAS, 2013).

Atualmente aproximadamente 7% dos brasileiros sofrem de alguma deficiência visual, sendo que 20% a 30% são crianças (IBGE, 2000). Especificamente sobre estrabismo, 2% das crianças no mundo sofrem com algum tipo desta patologia (VAUGHAN; ASBURY, 1990, p. 216). Estima-se uma prevalência de 1 a 2,5% de ambliopias nas crianças, na maioria das vezes causadas por um estrabismo não tratado (DÓRIA, 2006).

Segundo Dória (2006), os testes para detecção de estrabismo devem ser efetuados a partir dos 6 meses de vida e depois aos 2 e 5 anos. Na idade escolar, depois dos 6 anos, uma grande parte dos problemas sensoriais ligados ao desenvolvimento da visão é mais dificilmente tratável e frequentemente não se consegue a recuperação total. Por isso a importância da detecção precoce, a qual pode ser realizada através de métodos como o método de Hirshberg e o método de cobertura (DÓRIA, 2006). Estes métodos baseiam-se no reconhecimento visual de determinadas características presentes em olhos saudáveis e não saudáveis.

Reconhecer características em imagens é o objeto de estudo do processamento de imagens, uma subárea do reconhecimento automático de padrões. O processamento de imagens digitais, especialmente na medicina, tem evoluído ao longo dos anos (DUNCAN; AYACHE, 2000) e com a sempre crescente capacidade de processamento dos computadores, limites são ultrapassados e aplicações novas são disponibilizadas ao público em geral.

Para colaborar com a prevenção desta doença, o presente trabalho desenvolve um aplicativo para dispositivos móveis, que serve para efetuar uma triagem, sem a necessidade de um médico, onde destaca os possíveis casos da doença para posterior confirmação do diagnóstico e tratamento com o especialista. Pretendendo ser um aplicativo de fácil uso, pode ser utilizado por pessoas leigas em ambiente comum, como por exemplo, em creches onde os próprios professores poderão efetuar o teste com as crianças de forma periódica, acompanhando assim a possibilidade da ocorrência da doença.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é o desenvolvimento de um aplicativo para dispositivos móveis que usam a plataforma Android, para realizar a detecção do estrabismo utilizando-se de técnicas de processamento de imagem e visão computacional, alinhado com as técnicas utilizadas por oftalmologistas para detecção desta patologia.

Os objetivos especificos são:

a) capturar as imagens utilizando a câmera do dispositivo móvel;
b) determinar a técnica de detecção de estrabismo e de processamento de imagem mais adequadas às limitações tecnológicas de um dispositivo móvel;
c) realizar a detecção do estrabismo através das imagens capturadas;
d) apresentar o resultado da detecção informando o tipo de desvio.

1.2 ESTRUTURA DO TRABALHO

O presente trabalho é dividido em quatro capítulos, sendo que no primeiro capítulo é apresentada uma introdução ao tema abordado, os objetivos e a estrutura deste trabalho. O segundo capítulo contém a fundamentação teórica necessária para o entendimento do mesmo.

No terceiro capítulo é apresentado o desenvolvimento do trabalho. São expostos os requisitos a serem atendidos, a especificação, sua implementação, técnicas e ferramentas utilizadas e a sua operacionalidade. No final do capítulo é apresentado um comparativo com os trabalhos correlatos.

No quarto e ultimo capítulo, são apresentadas as conclusões do presente trabalho, apresentando também sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

A seguir na seção 2.1, são apresentados conceitos sobre visão binocular, que é fundamental para o entendimento da seção seguinte. Na seção 2.2 é apresentado o que é estrabismo junto com suas causas, os métodos utilizados para diagnosticá-lo e seu tratamento.

Dentro da seção 2.2 ainda é apresentado uma subseção que tem como objetivo fundamentar o teste de Hirshberg. Na seção 2.3 são apresentadas as técnicas de processamento de imagem. Nessa seção é apresentada a transformada de Hough, parte fundamental para o desenvolvimento do trabalho.

Na seção 2.4 são apresentados fundamentos sobre o desenvolvimento para a plataforma Android. Nesta seção ainda é apresentado a biblioteca OpenCV e as principais funcionalidades utilizadas. Por fim, são apresentados os trabalhos correlatos na seção 2.5.

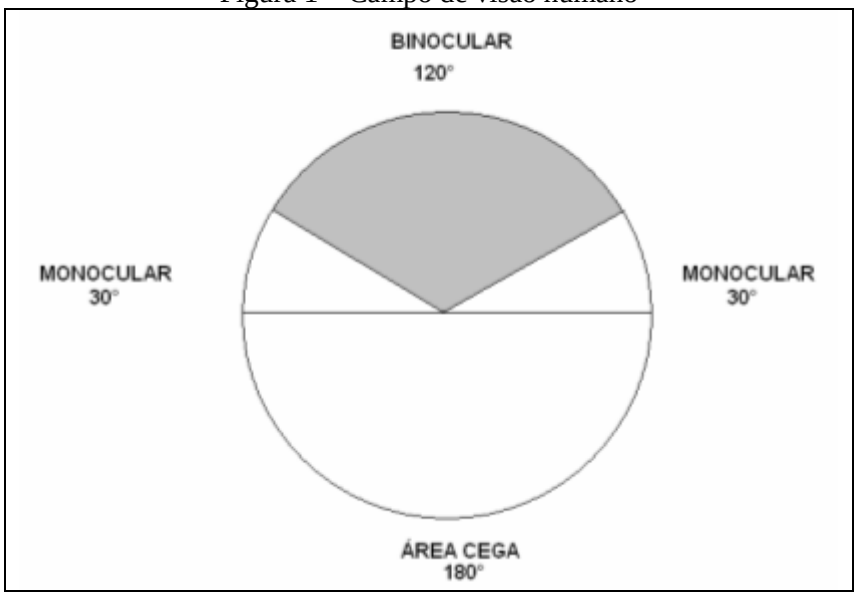
2.1 VISÃO BINOCULAR

Em cada olho tudo que é projetado é visto como estando na frente do observador. Assim, se dois objetos diferentes forem projetados, os dois objetos serão vistos superpostos, porém as diferenças impedirão a fusão em uma única imagem. Como cada olho encontra-se em uma posição espacial diferente, a imagem capturada de cada um deles é ligeiramente diferente, caracterizando a visão binocular (VAUGHAN; ASBURY, 1990, p. 216).

Segundo Vaughan e Asbury (1990, p. 217), a fusão sensorial e a estereopsia são os dois processos fisiológicos responsáveis pela visão binocular. A fusão sensorial é o processo que faz com que as diferenças entre as imagens apreciadas não sejam percebidas.

Enquanto a fusão sensorial é possível devido às diferenças sutis entre as duas imagens, a estereopsia é possível devido ao reconhecimento destas mesmas diferenças. Quando pontos diferentes da retina são estimulados ao mesmo tempo, a fusão de imagens diferentes resulta na percepção de um objeto em três dimensões. Na figura 1 é apresentado em graus o campo visual humano e em qual parte dele é caracterizado a visão binocular.

Figura 1 – Campo de visão humano



Fonte: Couto (2014).

2.2 ESTRABISMO

Em condição de visão binocular normal, a imagem de um objeto fixado incide simultaneamente sobre cada olho e os olhos estão alinhados. Em uma pessoa com estrabismo qualquer um dos olhos pode estar desalinhado, de tal maneira que somente um olho de cada vez veja o objeto fixado. O desvio do alinhamento perfeito, que caracteriza o estrabismo, pode ocorrer em qualquer direção (para dentro, para fora, para cima, para baixo, ou em uma direção rotatória ao redor do eixo visual) (VAUGHAN; ASBURY, 1990, p. 211).

O movimento dos olhos é controlado por seis pares de músculos comandados pelos nervos cranianos que estão conectados ao sistema nervoso central. Esses músculos precisam agir em perfeito equilíbrio e sincronia para que os olhos permaneçam alinhados (VARELLA, 2014).

Alguns fatores podem comprometer esse funcionamento e provocar o estrabismo. Entre as causas prováveis, destacam-se: dificuldade motora para coordenar o movimento dos olhos; grau elevado de hipermetropia, que obriga forçar a aproximação dos olhos para compensar a dificuldade de visão; baixa visão em um dos olhos; doenças neurológicas (acidente vascular cerebral-AVC, paralisia cerebral e traumas), genética (síndrome de Down), oculares (catarata congênita), infecciosas (meningite, encefalite), da tireoide, diabetes e hereditariedade (VARELLA, 2014).

Os sinais do estrabismo variam de acordo com a idade. Nos primeiros anos de vida, não há referência ao principal sintoma: a visão dupla ou diplopia. Ele não aparece nas

crianças, porque elas desenvolvem um mecanismo de supressão e apagam a imagem formada pelo olho que sofreu o desvio (VARELLA, 2014).

Como consequência por desuso, não se desenvolve a região do cérebro responsável pela visão desse olho, e a imagem vai ficando cada vez mais fraca até desaparecer por completo. Na figura 2 um exemplo manifesto da doença.

Figura 2 – Exemplo de paciente estrábico

Fonte: Brum (2014).

O tratamento do estrabismo tem como objetivo melhorar o alinhamento ocular, a fim de possibilitar que os dois olhos enxerguem na mesma direção (visão binocular). O tratamento pode envolver óculos, oclusor, prismas ou cirurgias nos músculos oculares. A cirurgia de estrabismo em adultos é restauradora. Ainda que a visão não possa ser recuperada, o aspecto estético pode ser recuperado com o alinhamento dos olhos (BRUM, 2014).

O estrabismo diagnosticado quando criança, onde a visão está ainda em formação, é extremamente importante para detecção de ambliopias e outras afecções, que, se não forem diagnosticadas precocemente, poderão deixar sequelas graves, como por exemplo, a redução e até a perda da visão no olho estrábico (BICAS, 2011, p. 129).

Há diferentes métodos para detecção de estrabismo: como o teste de cobertura, teste de Hirshberg, entre outros. O método de Hirshberg será abordado a seguir por ter sido o método proposto para implementação nesse trabalho.

2.2.1 Método de Hirshberg

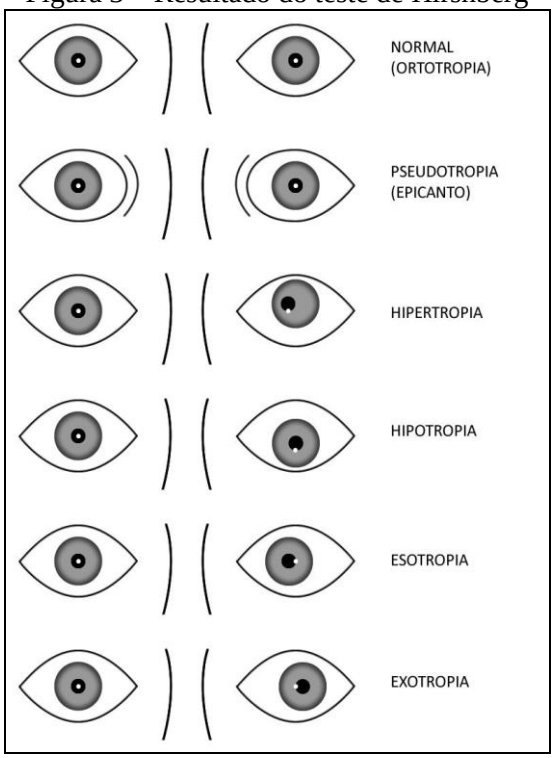
O método de Hirshberg é um teste de reflexo corneano com objetivo de realizar a detecção do estrabismo. Consiste no paciente fixar o olhar em um foco de luz a uma distância de aproximadamente 33 cm. A descentralização da reflexão da luz é percebida no olho que desvia. Com este método é possível saber inclusive qual é a direção e ângulo do desvio (BICAS, 2011, p. 133).

Segundo Siva, Ferreira e Pinto (2013, p. 11), a realização do exame pode ser dividida em 3 etapas:

a) preparo: para que seja realizado o exame, o paciente deve posicionar-se adequadamente, permanecendo imóvel, em um ambiente com pouca luz, com a cabeça alinhada ao eixo axial e fixando o olhar no infinito;
a) técnica: avalia-se o paciente iluminando simultaneamente os dois olhos e observando a posição relativa do reflexo corneano;
b) interpretação: analisa-se a posição relativa ao reflexo corneano. Ele pode ser visto no centro (sem desvio) ou na borda pupilar, entre a borda e o limbo ou no limbo.

A figura 3 apresenta os desvios oculares detectados a partir do teste de Hirshberg.

Figura 3 – Resultado do teste de Hirshberg



Fonte: Siva, Ferreira e Pinto (2013, p. 12).

Pode-se observar os prefixos utilizados para nomear cada direção de desvio: hiper para cima, hipo para baixo, eso nasal e exo temporal. Observa-se também que o desvio se caracteriza pela posição do reflexo corneano em diferentes posições da pupila (SIVA; FERREIRA; PINTO, 2013, p. 13).

2.3 PROCESSAMENTO DE IMAGENS

O termo processamento de imagem é dado a um processo que recebe de entrada uma imagem e o retorno é uma imagem modificada (COSTA; CESAR, 2001, p. 216). Porém, segundo Gonzalez e Woods (2008, p. 25), o resultado do processamento da imagem pode ser apenas de atributos da imagem, como por exemplo, a área de interesse da imagem.

Uma imagem digital é uma função bidimensional de intensidade de luz f (x, y), onde x e y são as coordenadas espaciais e o valor de f em qualquer ponto (x, y) é proporcional ao brilho da imagem naquele ponto. Também pode ser considerada como sendo uma matriz cujos índices de linha e colunas identificam um ponto na imagem. Este ponto é chamado de pixel (GONZALEZ; WOODS, 2000, p. 4).

2.3.1 Algoritmo de Canny

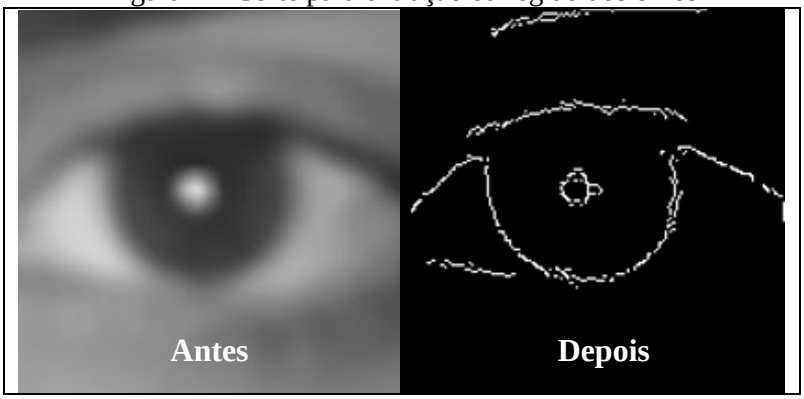
O algoritmo de Canny foi desenvolvido por John F. Canny em 1986, com a proposta de ser um detector ótimo para as bordas do tipo degrau, comum em imagens digitais. Consiste em um operador gaussiano de primeira derivada que suaviza os ruídos e localiza as bordas (VALE; POZ, 2002).

Borda é a fronteira entre duas regiões cujos níveis de cinza predominantes são razoavelmente diferentes. Analogamente, pode-se definir borda de luminosidade como uma descontinuidade na luminosidade. Pode-se ainda, definir borda de textura ou borda de cor, em imagens onde as informações de textura ou cor, respectivamente, são as mais importantes (MARQUES FILHO; VIEIRA NETO, 1999, p. 37).

As bordas constituem informações de alta freqüência e encerram propriedades significativas de uma imagem. Estas propriedades incluem descontinuidades geométricas e as características físicas dos objetos. Estas propriedades são passadas à imagem, pois variações pertinentes ao objeto ocasionam alterações nos tons de cinza (VALE; POZ, 2002).

Segundo Canny (1986), existem três critérios que definem uma detecção ótima da borda: o algoritmo deve marcar tantas bordas quanto possível; as bordas marcadas devem estar tão perto quanto possível da borda real e cada borda na imagem deve ser marcada uma vez, sendo que o ruído da imagem não deve criar bordas falsas. Na figura 4 é apresentado o resultado da aplicação do detector de bordas de Canny.

Figura 4 – Corte para extração da região dos olhos



2.3.2 Transformada de Hough

A transformada de Hough foi desenvolvida por Paul Hough em 1962 e patenteada pela IBM. Foi elaborada para detectar características analiticamente representáveis em imagens, assim como linhas, círculos e elipses. A transformada é aplicada após a imagem sofrer um pré-processamento, comumente a detecção de bordas, como por exemplo, o algoritmo de Canny (JAMUNDÁ, 2000).

Na teoria da transformada de Hough para detecção de círculos qualquer ponto de borda em uma imagem binária pode ser considerado parte de um círculo. O algoritmo consiste em passar em todos os pontos de borda na imagem real, aplicar uma equação com fórmula conhecida e fazer o mapeamento desse ponto para o espaço de parâmetros ou espaço de Hough (LAGANIÈRE, 2011, p. 168).

A equação paramétrica, usada pela transformada de Hough para detecção de círculos, é apresentada na figura 5.

Figura 5 – Equação paramétrica do círculo

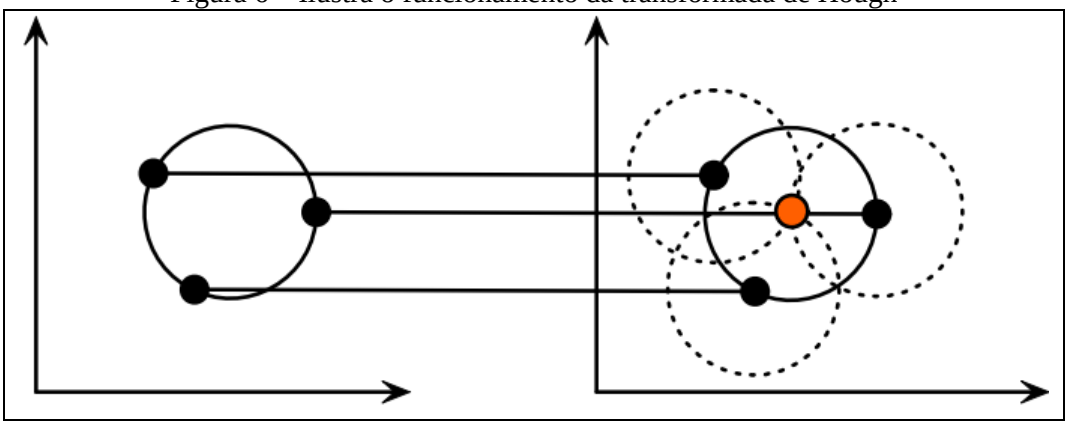
$$x = a + r\cos\theta$$
$$y = b + r\sin\theta$$

Fonte: Rhody (2005, p. 1).

A equação paramétrica do círculo possui três parâmetros, `a` e `b` representando o centro do círculo e `r` representando o raio. Quando o ângulo `θ` passar por todos os 360 graus, os pontos (`x, y`) traçaram o perímetro de um círculo.

O trabalho da transformada de Hough é desenhar o perímetro de um círculo para cada ponto de borda encontrado, no espaço de Hough, como pode ser visto na figura 6.

Figura 6 – Ilustra o funcionamento da transformada de Hough



Fonte: Rhody (2005, p. 2).

Cada ponto de borda no espaço geométrico (lado esquerdo) gera o traço de um círculo no espaço de Hough (lado direito). Os traços de círculos gerados no espaço de Hough tem um

ponto de intersecção que representam (`a`, `b`), que por sua vez representam o centro do círculo na imagem (RHODY, 2005, p. 2).

O espaço de Hough é representado através de uma matriz de acumulação de 3 dimensões devido aos 3 parâmetros da equação como ilustra a figura 5. Com isso a implementação da transformada gera um consumo de memória elevado e um maior tempo de processamento.

Existem várias técnicas para reduzir esse custo e uma delas é fixar o parâmetro `r` que é o raio do círculo. Com isso é possível utilizar uma matriz de 2 dimensões, visto que o raio já é conhecido. Caso para o problema em questão não seja possível fixar o valor do raio, uma técnica comum é ter um intervalo bem definido para o tamanho do raio, com isso reduzindo o tamanho da matriz consideravelmente (RHODY, 2005, p. 2).

Como último passo o algoritmo tem que encontrar os círculos na matriz e para isso é utilizado o conceito de votação. Cada vez que é encontrado um ponto de intersecção na imagem real, é dado um voto para o ponto em questão na matriz de acumulação. O algoritmo varre a matriz verificando os pontos com mais votos. Estes pontos por sua vez representam os círculos na imagem real (RHODY, 2005, p. 3).

2.3.3 Haar classificador em cascata

Inicialmente proposto por Viola (2001) e depois melhorado por Lienhart, Kuranov e Pisarevsky (2002). Trata-se de um classificador em cascata que utiliza características de um objeto, conhecidas como *Haar-like features*, para após uma fase de treinamento efetuar uma eficiente detecção do objeto em questão em imagens arbitrárias (CASCADE CLASSIFICATION, 2014).

O classificador é treinado com algumas centenas de imagens de um determinado objeto que se deseja detectar, como por exemplo um carro ou uma face. Essas amostras são chamadas de exemplos positivos. Após isso, é feito o treinamento com amostras negativas que são imagens que não contém o objeto em questão. Todo o processo de treinamento é feito com imagens do mesmo tamanho (CASCADE CLASSIFICATION, 2014).

Depois de o classificador ser treinado, ele pode ser aplicado para uma região de interesse, que deve ser do mesmo tamanho da imagem utilizada no treinamento. O classificador retorna 1 se a região contém o objeto e 0 caso contrário.

Para procurar por objetos em imagens maiores é criada uma espécie de janela de busca que pode ser movida por toda a imagem. Além da opção da janela de busca, o classificador foi desenhado para trabalhar com o redimensionamento do objeto de interesse; com isso pode-se

buscar o objeto em diferentes resoluções, que é mais eficiente do que ficar redimensionando a imagem de interesse (CASCADE CLASSIFICATION, 2014).

A palavra cascata no nome do classificador significa que o classificador resultante consiste em vários classificadores simples, denominados pelo algoritmo de estágio. Cada estágio é aplicado subsequentemente para uma região de interesse até que algum estágio rejeite ou todos os estágios sejam passados (BRADSKI; KAEHLER, 2008, p. 508).

2.4 DESENVOLVIMENTO PARA ANDROID

Android é um sistema operacional, mantido pelo Google, voltado para dispositivos móveis baseado no núcleo do sistema operacional de código aberto Linux. Desenvolvido inicialmente pelo Open Handset Alliance, que é uma aliança feita por grandes empresas como Google, HTC, Dell, Intel, Motorola, Qualcomm, Texas Instruments, Samsung, LG, T-Mobile e Nvidia. A aliança tem como objetivo criar um padrão aberto para *smartphone* (ANDROID DEVELOPERS, 2013).

A ferramenta de código aberto, Android *Developer Tools* (ADT), que é um *plugin* para o ambiente de desenvolvimento Eclipse, possibilita ao desenvolvedor ter um ambiente para construir aplicações para Android. Este *plugin* é integrado com a *Integrated Development Environment* (IDE) Java com recursos de ajuda, teste, depuração e empacotamento dos aplicativos para Android (ANDROID TOOLS, 2013). Em conjunto com o *plugin* é utilizado o *Software Delevopment Kit* (SDK) de desenvolvimento para Android que provê bibliotecas e ferramentas necessárias para desenvolvimento (ANDROID SDK, 2013).

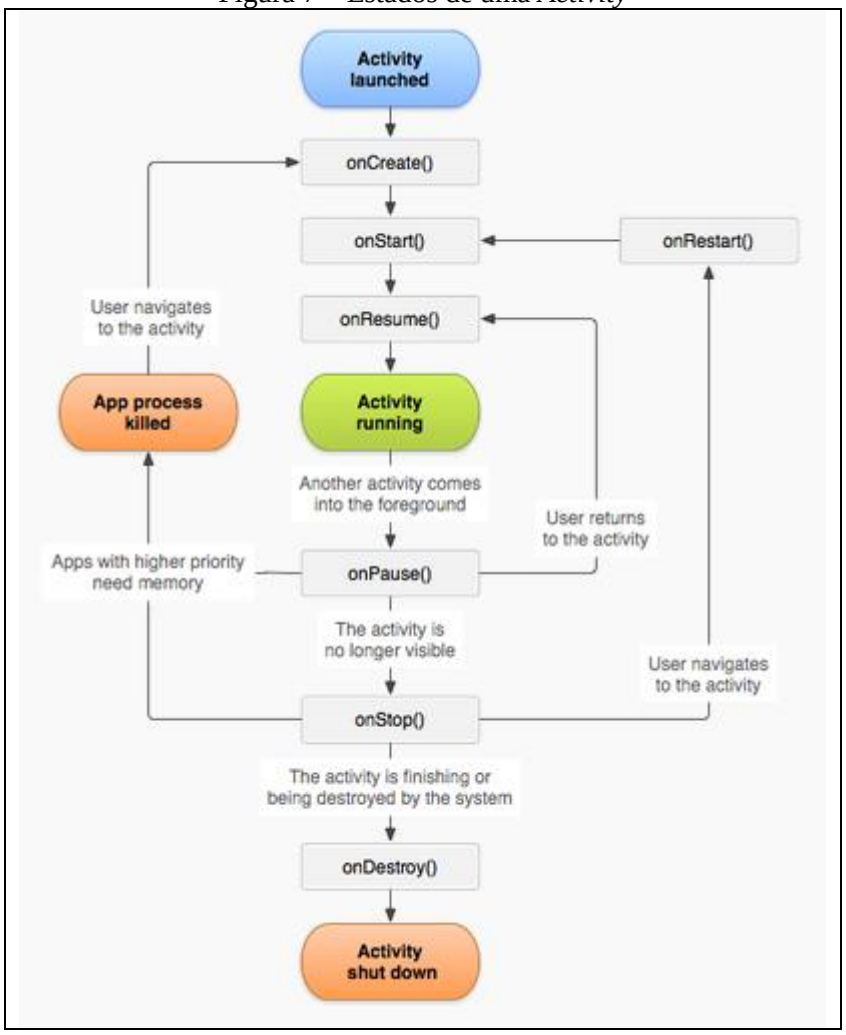
2.4.1 Activity

Uma *Activity* representa uma tela para o usuário interagir com aplicativo. Todas as *activities* possuem interação com usuário, de modo que uma *Activity* tem a responsabilidade de criar uma janela onde serão aprensentados todos os componentes de interface gráfica. Uma aplicação Android é composta por diversas *activities* (ACTIVITY, 2014).

As *activities* possuem um ciclo de vida bem definido. Este ciclo de vida é gerenciado com uma fila de *activities*. Quando uma nova *Activity* é iniciada, a mesma é colocada no topo da fila e seu estado será de executando.

A *Activity* que estava rodando anteriormente, continuará na fila, porém só voltará para o primeiro plano de execução quando a *Activity* que estiver rodando terminar sua execução. O diagrama apresentado na figura 7 é apresentado importantes estados que uma *Activity* pode assumir.

Figura 7 – Estados de uma *Activity*



Fonte: Activity (2014).

Os retângulos representam os métodos de *callback* disponibilizados pela *Activity*. Estes *callbacks* podem ser utilizados para realizar operação entre as mudanças de estado que ocorrem no ciclo de vida da *Activity*.

O ciclo de vida apresentado na figura 7 pode ser divido em 3 macros ciclos. O primeiro é chamado de ciclo de vida completo. Este acontece entre a chamada do primeiro *callback* `onCreate` e termina em uma única chamada ao *callback* `onDestroy`. Este ciclo de vida abrange toda a execução da *Activity*.

O segundo é denominado de ciclo de vida visível. Este acontece entre uma chamada do *callback* `onStart` até a chamada do *callback* `onStop`. Neste tempo o usuário da aplicação poderá ver a *Activity* sobre a tela do dispositivo.

O último é denominado de ciclo de vida de primeiro plano. Este acontece entre a chamada do *callback* `onResume` até a chamada do *callback* `onPause`. Durante esse tempo a *Activity* está na frente de todas as outras *activities* e interagindo com o usuário. Uma *Activity* pode frequentemente entrar e sair deste estado.

Para efetuar a troca de dados de uma *Activity* para outra é feito o uso de um recurso chamado *Intent*. Um *Intent* representa uma mensagem que pode ser usada para requisitar uma ação de uma outra *Activity* e até mesmo de um componente de outra aplicação (INTENT, 2014).

Os principais casos onde deve ser feito o uso de um *Intent* são para iniciar uma *Activity*, para iniciar serviços e para realizar um *broadcast*. Um *broadcast* é uma mensagem que todas as aplicações interessadas podem receber (INTENT, 2014).

2.5 TRABALHOS CORRELATOS

Foram identificados trabalhos que abordam o uso de técnicas de processamento de imagem para implementação de métodos de detecção de estrabismo. No âmbito nacional pode-se citar a "Ferramenta voltada à medicina preventiva para diagnosticar casos de estrabismo" (MEDEIROS, 2008) e a "Metodologia Computacional para Detecção Automática de Estrabismo em Imagens Digitais através do Teste de Hirschberg" (ALMEIDA, 2010). Outro trabalho que se assemelha é o "Protótipo de Software para Autenticação Biométrica baseada na Estrutura da Íris em Dispositivos Móveis" (KRUEGER, 2012).

2.5.1 Ferramenta voltada à medicina preventiva para diagnosticar casos de estrabismo

Este trabalho teve como objetivo desenvolver um protótipo utilizando abordagens de processamento de imagem para realização do teste de Hirschberg. A ideia de Medeiros (2008) foi disponibilizar uma ferramenta para que o usuário pudesse fazer o diagnóstico em sua própria casa.

Como hardware para os testes foi utilizada uma câmera digital Mitsuca DS5028BR de 5 megapixels com o zoom da máquina no máximo (4x), com o flash ligado. Segundo Medeiros (2008, p. 58) "os testes realizados com esta máquina digital tiveram um acerto de 100% para identificar a íris e o reflexo no olho da pessoa. Também foi capaz de extrair com precisão as medidas propostas".

Para a implementação foi feito o uso da biblioteca ImageJ (IMAGEJ, 2013), que é uma ferramenta de código aberto que disponibiliza implementações de algoritmos de processamento de imagem através de seus *plugins*.

Para a detecção foi utilizado o operador de Canny em conjunto com a transformada circular de Hough, que segundo Medeiros (2008, p. 58), "foi o que melhor se adaptou para atingir os objetivos do desenvolvimento da ferramenta". O operador de Canny foi utilizado

com objetivo de extrair as bordas da imagem e a transformada de Hough foi útil para resolver o problema de identificar as íris e os reflexos ocasionados pelo *flash* da câmera digital.

Algumas limitações do trabalho, segundo Medeiros (2008, p. 61), é que a ferramenta não consegue exibir qual o tipo de estrabismo (esotropia ou exotropia) e também que as medidas reais dos elementos avaliados não são precisas, o que seria necessário para que o diagnóstico fosse mais realista.

2.5.2 Metodologia computacional para detecção automática de estrabismo em imagens digitais através do teste de Hirschberg

Almeida (2010) descreve uma abordagem para construção de um algoritmo com o objetivo de detectar o estrabismo através de técnicas de processamento de imagem, visão computacional e reconhecimento de padrões. O trabalho foi dividido em quatro etapas: localização da região dos olhos, localização dos olhos, localização do limbo e do brilho, e identificação automática do estrabismo, que junto com as outras etapas representa o resultado final do trabalho.

Na primeira etapa de localização da região dos olhos, obteve 100% de acerto e a conclusão é que a utilização da filtragem homomórfica contribuiu para o acerto total, sendo que ela ajusta a iluminação, deixando-a mais uniforme na imagem.

Na segunda etapa de localização dos olhos, obteve 95,19% de acerto. Segundo Almeida (2010, p. 91), isso acontece devido à combinação das funções geoestatísticas. Porém, faz-se necessário aumentar a variabilidade das amostras de faces para alcançar uma metodologia mais genérica.

Na terceira etapa de localização do limbo e do brilho, obteve 97,5% de acerto utilizando o método de Canny e a transformada de Hough e concluiu-se que o erro ocorrido foi causado principalmente por reflexos luminosos gerados na aquisição das imagens.

Foram realizados testes para cada uma das quatro etapas para comprovar a eficiência do método proposto de forma isolada, para cada problema a se resolver, com o objetivo de aplicar cada etapa em problemas diferentes, se necessário.

As contribuições do trabalho, segundo Almeida (2010, p. 89), foram uma combinação inovadora de técnicas (filtragem homomórfica, transformada de Hough, funções geoestatísticas, análise discriminante *stepwise*, Máquina de Vetores de Suporte (SVM) e a medida de similaridade Erro Médio Absoluto (EMA)) para localizar olhos em faces humanas. Na fase de extração de características, outra contribuição foi a utilização de funções

geoestatísticas com intuito de extrair informações de textura dos olhos que ajuda na descriminação com precisão.

2.5.3 Protótipo de software para autenticação biométrica baseada na estrutura da íris em dispositivos móveis

Segundo Krueger (2012, p. 10), o objetivo do trabalho é desenvolver um protótipo de software para autenticação biométrica baseada na estrutura da íris para a plataforma iOS. Utiliza-se de técnicas de processamento de imagem com o intuito de localizar a região dos olhos e a íris, sendo disponibilizado para dispositivos móveis.

A arquitetura adotada segue os estudos de Daugman (1993) e Wildes (1997). Esta arquitetura foi dividida em cinco etapas, porém serão apresentadas as dois etapas que assemelham-se com o presente trabalho: aquisição da imagem e localização (KRUEGER, 2012, p. 40).

A primeira etapa de aquisição da imagem é dividida em três sub-etapas: aquisição da imagem, recorte da região de interesse e transformação da imagem para tons de cinza. (KRUEGER, 2012, p. 43).

Na segunda etapa de localização dos círculos da íris e da pupila é utilizado o filtro de mediana para suavizar os ruídos e manter os detalhes dos contornos. Após isso é aplicada a implementação do OpenCV da transformada de Hough para detectar os dois círculos, a íris e a pupila, respectivamente (KRUEGER, 2012, p. 44).

O resultado obtido a partir das íris de 10 indivíduos comprovou-se que é possível, em um tempo aceitável, realizar a autenticação biométrica com taxas de 3,2% para falsos negativos e 0% para falsos positivos. As taxas foram alcançadas após análise e ajuste do valor do limiar entre classes para 0,3, garantindo um bom desempenho em termos de autenticação (KRUEGER, 2012, p. 61).

Como principal limitação, segundo Krueger (2012, p. 60), devido a problemas na detecção do limite entre a pupila e a íris de olhos castanhos utilizando a técnica da transformada de círculos de Hough, o protótipo teve de ser limitado ao reconhecimento biométrico de apenas íris claras.

2.5.4 Comparativo entre os trabalhos correlatos

Nesta seção são apresentadas e discutidas as principais características da comparação entre os trabalhos correlatos, as quais estão ilustradas no quadro 1.

Quadro 1 – Quadro comparativo com os trabalhos correlatos

Características	Medeiros (2008)	Almeida (2010)	Krueger (2012)
dispositivos móveis			X
computadores convencionais	X	X	
utilização do OpenCV	X		X
implementação em	Java	C++	Objective-C
reconhecimento de olhos escuros	X	X	
reconhecimento de olhos claros	X	X	X
Quantidade de amostras mais relevantes		X	

No quadro 1 pode ser observado que o trabalho de Medeiros (2008) tem como principais características o desenvolvimento na linguagem de programação Java utilizando-se da biblioteca OpenCV para computadores convencionais. Também se pode destacar o reconhecimento tanto de olhos escuros quanto olhos claros.

Em relação ao trabalho do Almeida (2008) o que se pode destacar é a implementação na linguagem de programação C++ para computadores convencionais o também reconhecimento de olhos claros e escuros. O principal diferencial fica em relação à quantidade de amostras relevantes, onde com apoio de um especialista em um ambiente controlado conseguiu-se um melhor aproveitamento das amostras.

Em relação ao trabalho de Krueger (2012) pode-se destacar a implementação na linguagem Objective-C para dispositivos móveis onde os recursos, tanto de memória quanto de processamento são limitados. Outro ponto a se destacar é o uso da biblioteca OpenCV e como principal limitação segundo Krueger (2012) o não reconhecimento de olhos claros.

3 DESENVOLVIMENTO DO PROTÓTIPO

Neste capítulo são abordadas as etapas envolvidas no desenvolvimento do protótipo proposto. São apresentados os principais requisitos, a especificação, a implementação e, por fim, são listados os resultados e discussões.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

O protótipo foi desenvolvido atendendo aos seguintes requisitos:

a) disponibilizar uma interface para que o usuário possa tirar a foto dos olhos (Requisito Funcional - RF);
b) validar se as imagens capturadas atendem aos requisitos mínimos de luminosidade e enquadramento para que possa se fazer a detecção (RF);
c) detectar indícios de estrabismo na imagem capturada, caso ocorra a patologia (RF);
d) disponibilizar uma interface com o resultado da detecção (RF);
e) disponibilizar uma interface com o histórico das detecções efetuadas pelo dispositivo (RF);
f) utilizar a linguagem de programação Java (Requisito Não-Funcional RNF);
g) extrair a localização da pupila utilizando a transformada de Hough (RNF);
h) construir uma API para todo o processo de detecção do estrabismo que seja independente de plataforma (RNF);
i) ser compatível com a plataforma Android (RNF);
j) utilizar a biblioteca OpenCV para desenvolver a parte de processamento de imagem e visão computacional (RNF);
k) utilizar o ambiente de desenvolvimento Eclipse com o plugin de desenvolvimento para a plataforma Android (RNF).

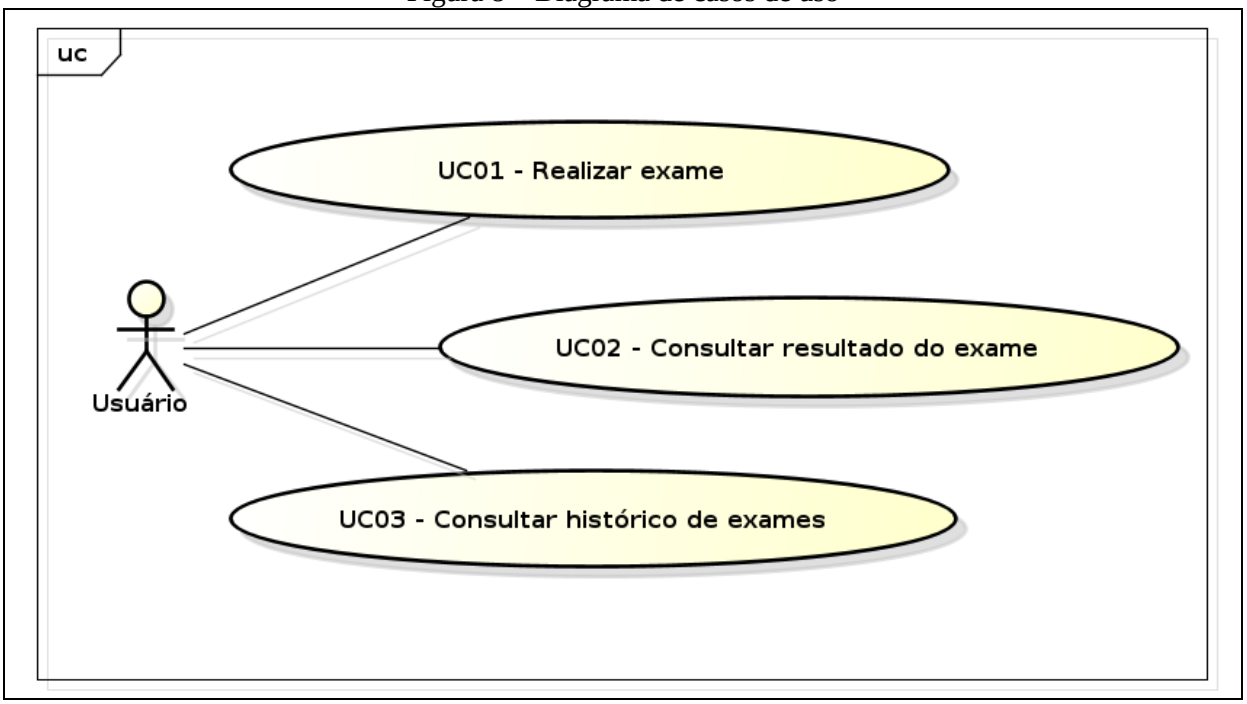
3.2 ESPECIFICAÇÃO

A especificação foi desenvolvida seguindo a análise orientada a objetos, utilizando a *Unified Modeling Language* (UML) em conjunto com a ferramenta Astah Community 6.7, a qual foi utilizada para modelagem e geração dos diagramas de casos de uso, de classes e de sequência.

3.2.1 Casos de uso

Nesta seção são descritos os casos de uso do protótipo, ilustrados na figura 8. Identificou-se apenas um ator, denominado Usuário, o qual faz uso de todas as funcionalidades do protótipo.

Figura 8 – Diagrama de casos de uso



3.2.1.1 Realizar exame

Este caso de uso descreve como o Usuário deverá proceder para realizar o exame através da ferramenta. Detalhes são apresentados no quadro 2.

Quadro 2 – Caso de uso UC01

UC01 – Realizar exame	
Descrição	O usuário entra no sistema para realizar um exame.
Pré-condições	Possuir um dispositivo que possua o recurso de *flash* e com o aplicativo OpenCV Manager instalado.
Fluxo principal	1. O usuário abre aplicativo; 2. O protótipo mostra a visão da câmera do dispositivo e mostra o seguinte aviso: "Toque para iniciar o exame..."; 3. O usuário toca na tela e inicia o exame; 4. O protótipo detecta a face; 5. O protótipo detecta os olhos e mostra a mensagem: "Não se mova. Tirando a foto para o exame..."; 6. O protótipo dispara o *flash* do dispositivo e tira a foto que será usada para realizar o exame; 7. O protótipo mostra uma tela de progresso com o seguinte texto: "Aguarde realizando o exame...". A tela permanece até o fim do exame; 8. O Usuário é redirecionado para tela de consulta de exame.
Fluxo de exceção	No passo 4, o protótipo não consegue encontrar a face na imagem. Com isso mostra a mensagem: "Nenhuma face encontrada". Após isso volta ao passo 4 novamente.
Fluxo de exceção	No passo 5, o protótipo detecta os olhos, porém constata que os mesmos não estão alinhados. Com isso mostra a mensagem: "Os olhos estão desalinhados". Após isso volta ao passo 4 novamente.

3.2.1.2 Consultar resultado do exame

Este caso de uso descreve como o Usuário deverá proceder para consultar o resultado do exame através da ferramenta. Detalhes são apresentados no quadro 3.

Quadro 3 – Caso de uso UC02

UC02 – Consultar resultado do exame	
Descrição	O usuário deseja consultar o resultado de um exame.
Pré-condições	Ter realizado todas as etapas do exame ou possuir algum exame a ser consultado no histórico de exames.
Fluxo principal	1. O usuário entra na tela de consulta do exame; 2. O sistema apresenta a tela de consulta de exame que possui dois botões, “salvar” e “descartar” e os seguintes campos: a) paciente: onde pode ser informado o nome do paciente examinado; b) Resultado do exame: informa ao usuário qual o resultado do exame. Apenas para visualização, não pode ser alterado; 3. O usuário não faz nenhuma alteração no exame em questão e pressiona no botão voltar do dispositivo; 4. O protótipo volta para a tela que fez a chamada para tela de consulta.
Fluxo alternativo	No passo 3, o Usuário altera o nome do paciente e clica no botão salvar. O protótipo atualiza o nome do paciente na base de dados do dispositivo.
Fluxo alternativo	No passo 3, o Usuário altera o nome do paciente e clica no botão voltar do dispositivo. O protótipo não atualiza o nome do paciente na base de dados do dispositivo.
Fluxo alternativo	No passo 3, o Usuário clica no botão “descartar”. O protótipo remove o exame em questão da base de dados do dispositivo.

3.2.1.3 Consultar histórico de exames

Este caso de uso descreve como o Usuário deverá proceder para consultar o histórico de resultados de exames realizados no dispositivo através da ferramenta. Detalhes são apresentados no quadro 4.

Quadro 4 – Caso de uso UC03

UC03 – Realizar exame	
Descrição	O usuário deseja consultar o histórico de exames realizados no dispositivo.
Fluxo principal	1. O usuário entra no sistema e no menu superior clica para expandir as opções e seleciona a opção “Histórico de exames”; 2. O protótipo para de mostrar a visão da câmera e mostra a tela de consulta de histórico. 3. O protótipo consulta a base de dados do dispositivo e apresentada uma lista com todos os exames realizados; 4. O usuário seleciona um exame na lista; 5. O protótipo mostra a tela de consulta de exame.
Fluxo alternativo	No passo 3, o protótipo consulta a base de dados do dispositivo, porém não encontra nenhum exame. A lista é apresentada vazia.
Fluxo alternativo	No passo 4, o Usuário clica no botão voltar do dispositivo. O protótipo volta para tela de realização do exame, ou tela inicial do aplicativo.

3.2.2 Diagrama de classes

Nesta seção são descritas as classes e seus respectivos pacotes que constituem o desenvolvimento do protótipo. Ao longo dessa seção serão abordados cada pacote e suas responsabilidades.

3.2.2.1 Pacote `br.com.rml.strabismusdetector.api`

Este pacote contém toda API responsável por fazer o processamento de imagem envolvido na detecção do estrabismo. Possui também a lógica envolvida na realização do exame em si.

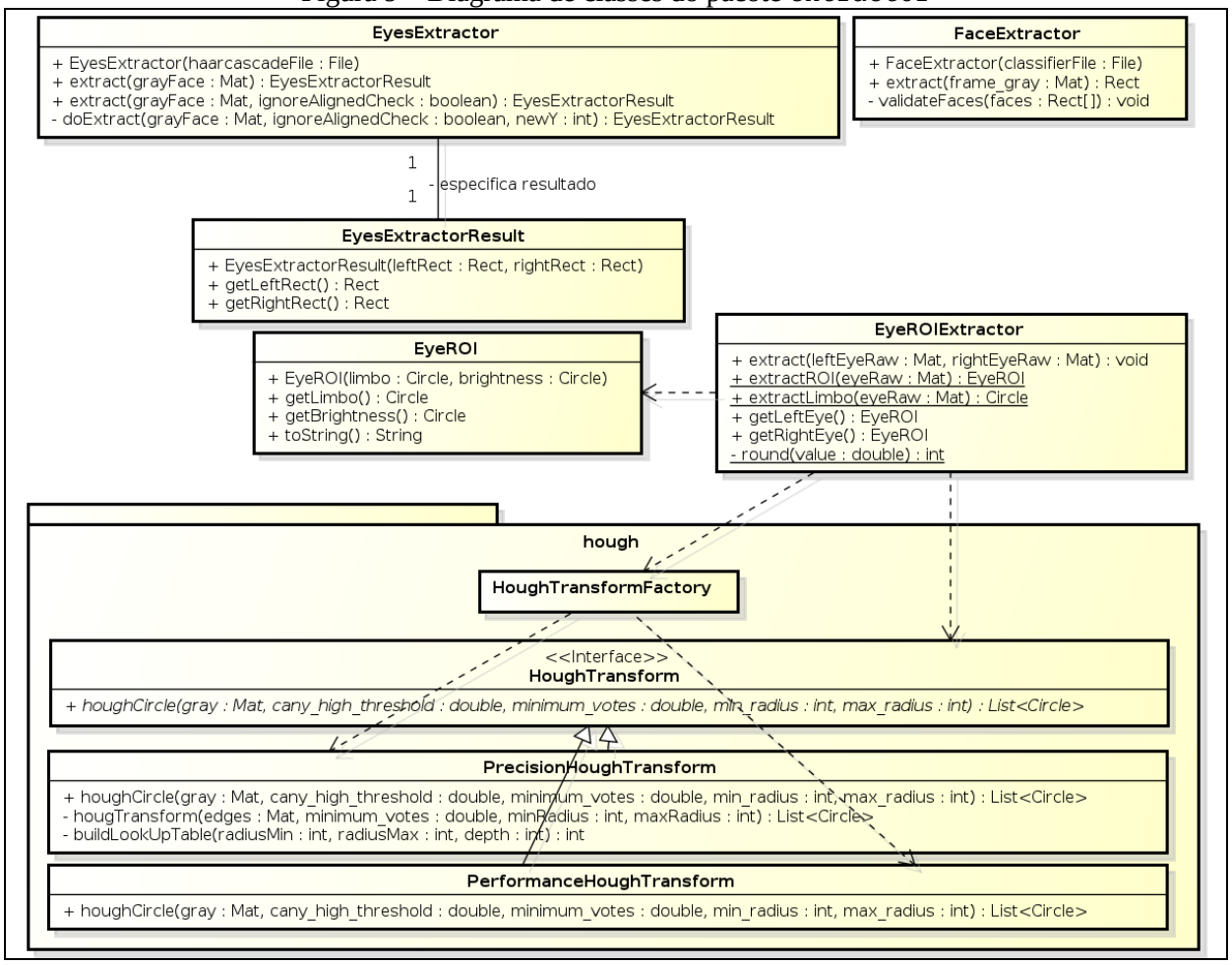
O objetivo do pacote é atender a um requisito não funcional. Este requisito define que o protótipo deve construir uma API para detecção do estrabismo que seja independente de plataforma.

Como o pacote é relativamente grande, para simplificar a explicação o mesmo será dividido em duas partes. Primeiro será apresentada a parte de extração de características, que foi separada em um pacote interno denominado `extractor`. Após isso, serão abordadas as classes envolvidas na interface externa da API e utilitários criados.

3.2.2.1.1 Pacote `extractor`

Este pacote possui todas as classes envolvidas na parte de extração de características utilizadas pelo protótipo. No pacote em questão, existem três classes que fornecem funcionalidade para os outros pacotes. Essas classes são `FaceExtractor`, `EyesExtractor` e `EyeROIExtractor`. O restante das classes serve apenas para auxiliar o trabalho das três classes citadas. Na figura 9 é apresentado o diagrama do pacote com mais detalhes.

Figura 9 – Diagrama de classes do pacote `extractor`



A classe `FaceExtractor`, como próprio nome diz, é responsável por extrair a face de uma imagem passada. Esse trabalho é realizado através do método `extract`, que retorna um retângulo onde foi encontrada a face na imagem.

A classe `EyesExtractor` é a classe responsável por extrair os olhos da imagem passada como parâmetro. A imagem passada deve ser a imagem de uma face para que o algoritmo consiga extrair os olhos com precisão. Para realizar a extração deve ser chamada uma das assinaturas do método `extract`. O algoritmo também pode checar o alinhamento dos olhos caso seja passado false para a opção `ignoreAlignedCheck`, com isso ele lançará uma exceção do tipo `UnalignedHeadException`, caso os olhos não estiverem na mesma linha.

Como retorno do processo de extração de olhos da imagem é utilizado o objeto `EyesExtractorResult`. Este objeto consiste em dois retângulos que indicam a posição onde os olhos foram encontrados na imagem.

Por último há a classe `EyeROIExtractor` que tem como objetivo, a partir de duas imagens de entrada, uma de cada olho, extrair as regiões de interesse para a execução do exame. As regiões de interesse são o limbo e o brilho de cada olho. Esta região de interesse é representada pelo objeto `EyeROI`. Este objeto consiste em dois círculos que representam o limbo e o brilho.

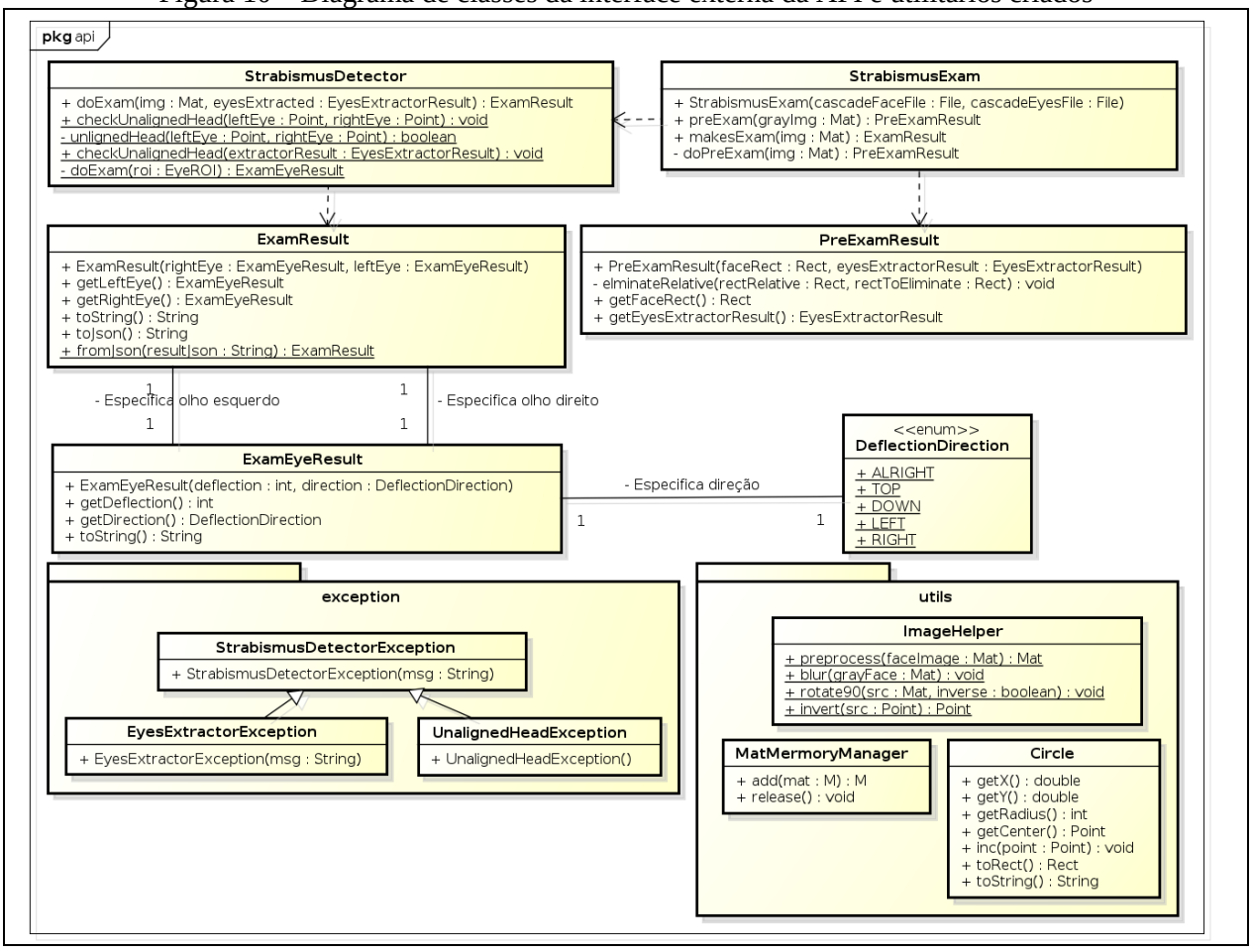
Para efetuar a extração das regiões de interesse é utilizada a transformada de Hough. Para tal extração são utilizadas duas implementações da transformada de Hough. Uma que tem como característica ser mais eficiente e outra que possui mais precisão, porém com um desempenho consideravelmente inferior.

A criação do objeto que representa a transformada de Hough é gerenciada por uma fábrica chamada `HoughTransformFactory`. Esta classe é responsável por saber qual implementação tem mais desempenho e qual que tem mais precisão.

3.2.2.1.2 Interface externa da API e utilitários criados

Nessa seção serão abordadas as classes que representam a interface entre a API e algumas classes utilitárias criadas para facilitar o trabalho. Na figura 10 é apresentado o diagrama de classes do pacote com mais detalhes.

Figura 10 – Diagrama de classes da interface externa da API e utilitários criados



A classe `StrabismusExam` é a classe que representa por onde o restante do código se comunica com a API de detecção de estrabismo. Essa classe permite que sejam realizadas duas operações, que são representadas pelos métodos `preExam` e `makeExam`.

O método `preExam` realiza uma prévia do que será feito no exame, com o objetivo principal de dar um *feedback* se a imagem é válida ou não, em um curto espaço de tempo. Este método não passa por todas as etapas do exame, por consequência acaba tendo um melhor desempenho do que o método `makeExam`.

O retorno do método `preExam` é representado pela classe `PreExamResult`. Esta classe consiste no resultado da extração dos olhos, representado pela classe `EyesExtractorResult` já comentado na seção anterior, e o retângulo que representa a localização da face na imagem. Caso exista algum problema no pré-exame é retornado uma exceção.

O método `makeExam` realiza o exame de fato. O resultado do exame é representado pela classe `ExamResult`. Esta classe consiste em dois objetos, um para o olho esquerdo e outro para o olho direito do tipo `ExamEyeResult`. Este objeto por sua vez consiste na quantidade de

pixels encontrada no desvio e numa enumeração `DeflectionDirection`, que representa a direção em que o desvio foi encontrado.

A classe `StrabismusDetector` é a classe responsável por, através de um objeto do tipo `EyesExtractorResult`, extrair da imagem original os olhos e fazer o processo de extrair as regiões de interesse. Este processo é realizado utilizando a classe `EyeROIExtractor`, já comentada na seção anterior.

Após as regiões de interesse serem extraídas, que são o limbo e brilho de cada olho, realiza a checagem para verificar se existe o estrabismo em algum dos olhos ou até mesmo nos dois. O resultado é representado `ExamResult`, já mencionada anteriormente. Como pode ser visto na figura 10, a classe `StrabismusExam`, mencionada anteriormente, utiliza-se dessa classe para realizar o exame.

Toda a exceção conhecida, lançada pela a API será uma extensão da classe `StrabismusDetectorException`. Atualmente existem apenas duas extensões específicas. Estas extensões estão representadas pelas classes `EyesExtractorException`, que pode ser lançada caso seja encontrado algum problema no processo de extração dos olhos que é realizado pela classe `EyesExtractor`; e `UnalignedHeadException` que pode ser lançada quando a API detectar um desalinhamento dos olhos durante o processo.

No pacote `utils` existem três utilitários. A classe `ImageHelper` tem o objetivo de auxiliar o usuário da API a fazer o pré-processamento da imagem através do método `preProcess` e possui mais alguns métodos utilitários.

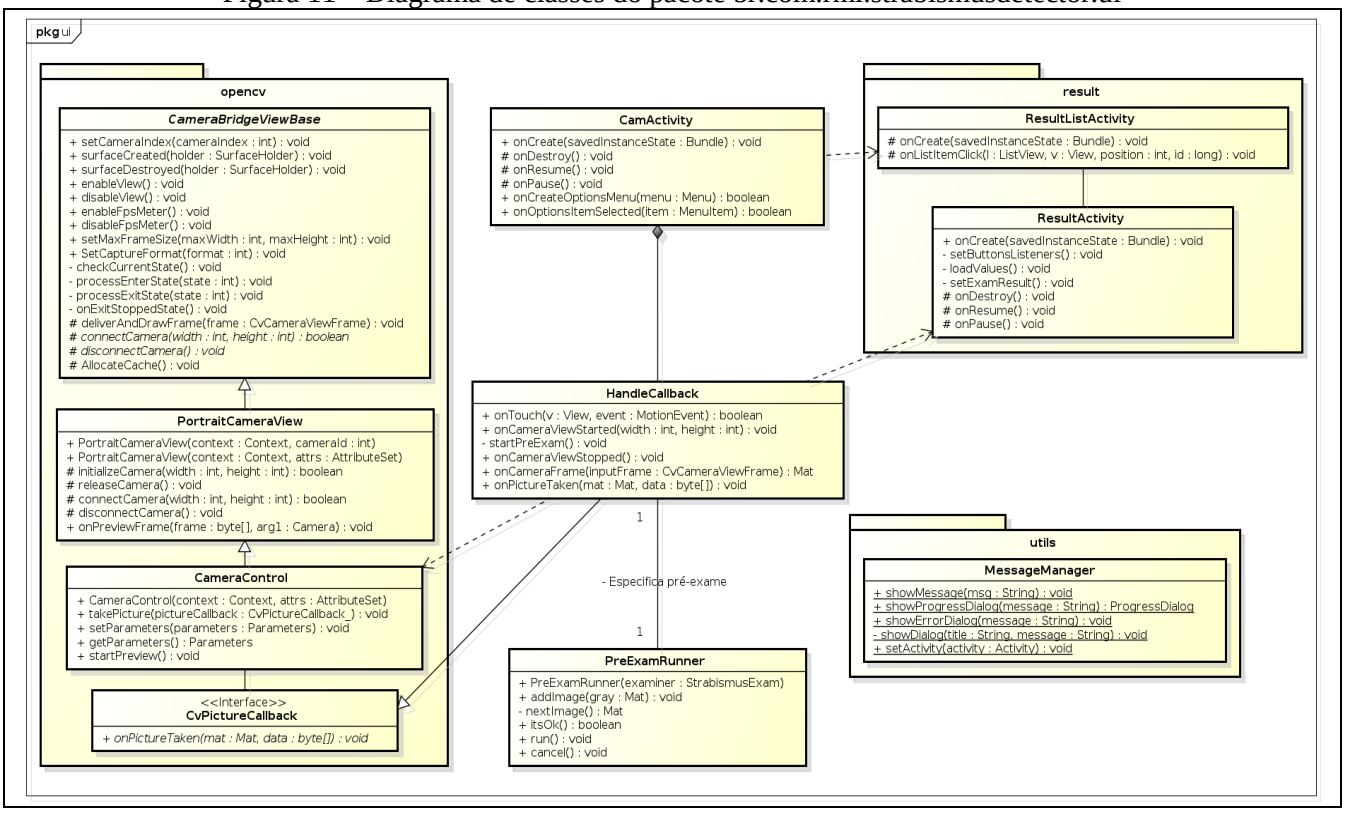
A classe `MatMermoryManager` é um gerenciador de memória para simplificar a liberação de memória dos objetos da classe `Mat` do OpenCV. Essa classe representa uma imagem em memória e está vinculada ao um objeto nativo que não é liberado com a ação do *Garbage Colector* do Java.

A classe `Circle`, como o próprio nome diz, representa um círculo. Tem como atributos o raio do círculo e um ponto, cordenadas na imagem `x` e `y`. Esse utilitário é utilizado pela classe `EyeROI`, para dois de seus atributos limbo e brilho.

3.2.2.2 Pacote `br.com.rml.strabismusdetector.ui`

Este pacote contém todas as classes responsáveis por fazer a parte de interface com o usuário. Na figura 11 é apresentado o diagrama do pacote com mais detalhes.

Figura 11 – Diagrama de classes do pacote br.com.rml.strabismusdetector.ui



A interface gráfica do protótipo é composta por três *Activities*. Estas são representadas pelas as classes `CamActivity`, `ResultListActivity` e `ResultActivity`. A classe `ResultListActivity` representa a lista de exames realizados pelo dispositivo e a classe `ResultActivity` representa o resultado de um exame.

A classe `CamActivity` representa a *Activity* principal do protótipo. A classe em si é utilizada apenas como provedor da interface gráfica. Todas as funcionalidades contempladas por ela estão divididas entre a classe `HandleCallback` e `PreExamRunner`.

A classe `HandleCallback` é responsável por gerenciar todos os eventos relacionados a câmera e a *Activity*. Os principais eventos da câmera são representados pelos métodos `onPictureTaken` e `onCameraFrame`. O evento `onPictureTaken` é chamado após o aplicativo tirar uma foto. Neste evento são recebidos os *bytes* da imagem que foi capturada ao tirar a foto.

O evento `onCameraFrame` é chamado para cada imagem recebida após o início da pré-visualização da câmera. Nesse momento é feito o uso da classe responsável por realizar o pré-exame, denominada `PreExamRunner`. Cada imagem recebida é repassada a uma fila que existe na classe de pré-exame. Esta classe se trata de uma *thread* que fica consultando a fila de imagens e realizando o pré-exame, utilizando-se do recurso de pré-exame disponibilizado pela classe `StrabismusExam`, já mencionada anteriormente.

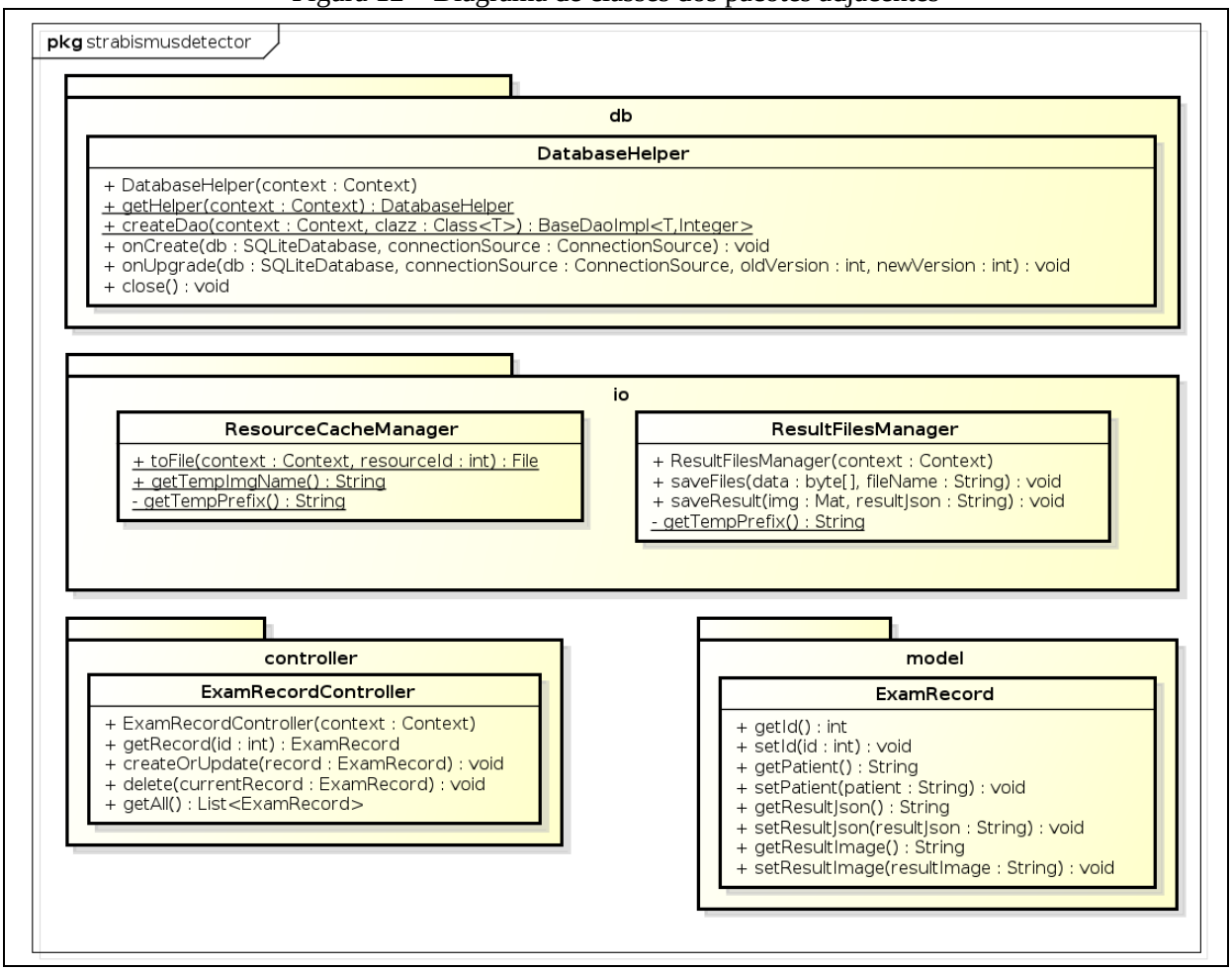
O conjunto de classes dentro do sub-pacote `opencv` são uma hierarquia que representam um componente visual. Essa hierarquia é responsável por fazer a renderização do conteúdo recebido da câmera, através do método `onCameraFrame` para a tela do dispositivo.

No pacote `utils` há apenas uma classe denominada `MessageManager`. Esta classe é responsável por fazer o gerenciamento das mensagens ou diálogos, que são telas que mostram informações ao usuário durante o uso da aplicação. Esta classe tem dois papéis fundamentais. O primeiro é sincronizar as mensagens, que podem vir de várias *threads*. O segundo é garantir que todas as mensagens sejam renderizadas na *thread* principal, que se trata de uma exigência da plataforma (ANDROID THREADS, 2014).

3.2.2.3 Pacotes adjacentes

Estes pacotes implementam funcionalidades ao negócio principal da aplicação. Esses pacotes representam a integração com o banco de dados e recursos do sistema, como arquivos. Na figura 12 é apresentado o diagrama do pacote com mais detalhes.

Figura 12 – Diagrama de classes dos pacotes adjacentes



A classe `DatabaseHelper` contida dentro do pacote `db` é a responsável por gerenciar a

conexão com o banco de dados. Nessa classe fica o controle de criação, onde é realizada através do método `onCreate` e atualização do banco de dados entre versões, onde é realizada através do método `onUpgrade`.

No pacote `io` estão as classes responsáveis por manter o acesso a arquivos da aplicação. A classe `ResourceCacheManager` é responsável por realizar um *cache* dos arquivos compactados dentro aplicativo em cada dispositivo. A classe `ResultFilesManager` é responsável por realizar o controle dos arquivos de resultado do exame. Em modo de depuração o aplicativo armazena as imagens dos exames realizados em uma pasta no dispositivo.

Para realizar a manipulação da base de dados foi utilizada a arquitetura de *Model View Controller* (MVC). Esta arquitetura tem como principal objetivo separar a lógica de negócio, denominado *model*, da camada de interface com o usuário, denominado v*iew*, do fluxo da aplicação denominado c*ontroller* (MVC, 2014). Com isso foram criados os pacotes `model` e `controller`.

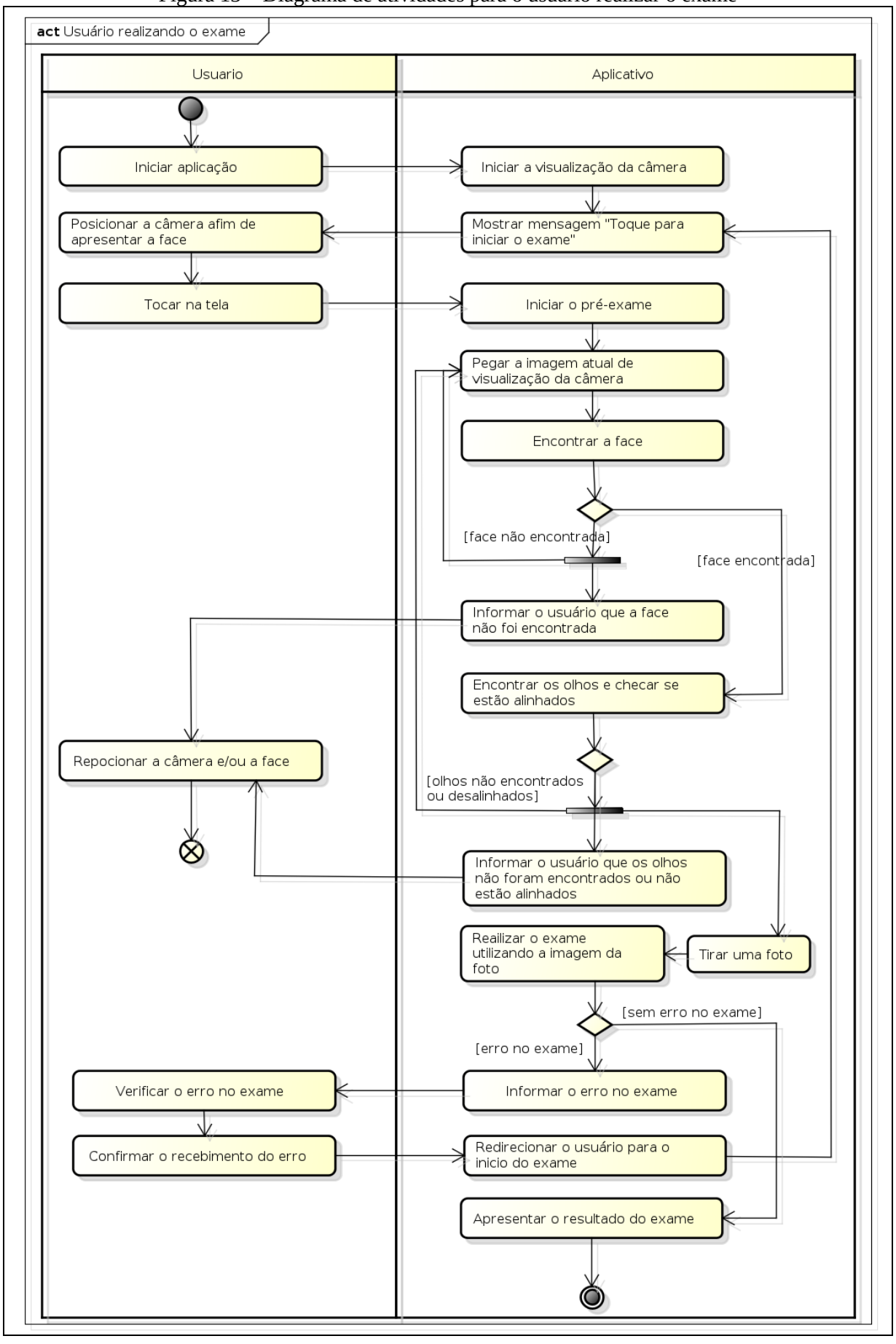
A classe `ExamRecord` representa o modelo de negócio que são os dados persistidos na base de dados. Esta mantém alguns dados sobre o exame que são o nome do paciente e o resultado do exame. A classe `ExamRecordController` tem como objetivo prover as informações da forma que a camada de *view* representada pela classe `ResultActivity` consiga apresentar as informações sem ter que fazer acesso direto ao banco de dados.

3.2.3 Diagrama de atividades

Nesta seção será apresentado o diagrama de atividades para o usuário realizar um exame através do protótipo. O diagrama abrange desde a atividade do usuário de abrir a aplicação até a visualização do resultado do exame. Foram contemplados todos os cenários, desde o cenário principal até os cenários de exceção.

Na figura 13 é apresentado o diagrama de atividades. Nele são abordadas as atividades que o usuário pode executar no software com o objetivo de realizar um exame. Também são abordadas as atividades onde podem ocorrer erros que são apresentados ao usuário com o objetivo de auxiliá-lo a realizar o exame. Por fim, a apresentação do resultado do exame.

Figura 13 – Diagrama de atividades para o usuário realizar o exame



3.3 IMPLEMENTAÇÃO

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

3.3.1 Técnicas e ferramentas utilizadas

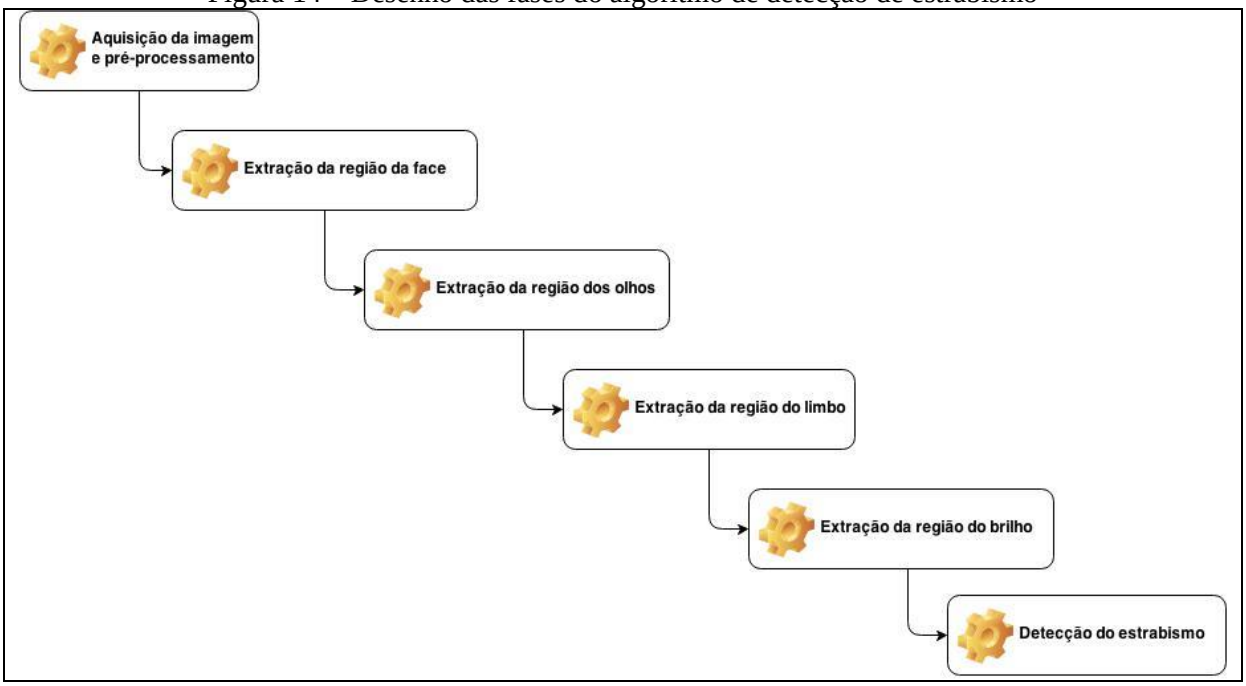
Para o desenvolvimento do protótipo foi utilizado a linguagem de programação Java sobre o sistema operacional Arch Linux. Foram utilizadas as seguintes ferramentas:

a) Android *Developer Tools* versão 22 sobre a plataforma Eclipse na versão Juno;
b) OpenCV na versão 2.4.7, como ferramenta para manipulação e processamento de imagens. Utilizado a versão para Android e a versão Java para *desktop*;
c) OpenCV *Manager* versão 4.4.2, que deve ser instalado no dispositivo para o funcionamento da biblioteca OpenCV para a Android;
d) Gson versão 2.2.4, como biblioteca criada pelo Google com o objetivo de fazer a serialização e deserialização de objetos para o formato Json;
e) OrmLite versão 4.48, como biblioteca para a acesso a banco de dados com suporte a *Object Relational Mapping* (ORM) para Android.

3.3.2 Fases do algoritmo de detecção de estrabismo

Nesta seção serão apresentadas todas as fases do algoritmo de detecção de estrabismo. O algoritmo foi dividido em fases justamente para que cada fase possa ser reaproveitada de forma independente. Na figura 14 são exemplificadas todas as fases, em ordem de execução.

Figura 14 – Desenho das fases do algoritmo de detecção de estrabismo



3.3.2.1 Aquisição da imagem

Nesta etapa é feito a interface entre as bibliotecas da plataforma Android com a *API* de detecção de estrabismo. As principais responsabilidades desta etapa são capturar a imagem, fazer as conversões necessárias para adaptar a imagem recebida da plataforma para a biblioteca OpenCV e preparar a imagem para a detecção.

A captura da imagem é realizada utilizando-se da classe `Camera` disponibilizada pela biblioteca da plataforma. Esta classe é responsável por concentrar toda a integração dos aplicativos com os hardwares de câmera do dispositivo. No quadro 5 são apresentadas de forma sintetizada todas as configurações realizadas na câmera.

Quadro 5 – Configurações da câmera

```
1    //Abre a camera para uso
2    Camera camera = Camera.open();
3    Parameters parameters = camera.getParameters();
4    //Liga o flash ao tirar a foto
5    parameters.setFlashMode(Parameters.FLASH_MODE_ON);
6    //Rotaciona a imagem em 90 graus
7    parameters.setRotation(90);
8    //Configura a câmera para tirar foto
9    parameters.setRecordingHint(false);
```

Na primeira linha é efetuada a inicialização da câmera para uso. A chamada do método `open` sem nenhum parâmetro indica que se deseja utilizar a câmera traseira do dispositivo. Na quinta linha é definido o *flash mode* para ligado. Isso indica que se quer ligar o *flash* do dispositivo apenas quando for tirada uma foto (não na pré-visualização).

Na linha sete, comanda-se à câmera para rotacionar em 90 graus a imagem capturada ao tirar uma foto. Isso se faz necessário, pois por padrão a imagem sempre virá em formato paisagem e todo o processamento é realizado com a imagem no formato retrato.

Na linha nove, informa-se à câmera que será utilizada com o objetivo de tirar fotos e não gravar um vídeo. Sem esta configuração não é possível obter uma boa resolução nas fotos tiradas.

Depois de efetuado as configurações necessárias a próxima etapa é executar a etapa de pré-exame. Esta etapa consiste em analisar cada imagem capturada pela câmera em modo de pré-visualização e verificar se a mesma está apta para realizar o exame.

Na quadro 6 é apresentado o código que realiza o pré-exame. Esse código roda em uma *thread* separada para não reduzir o *frames per second* (FPS) na fase de pré-visualização.

Quadro 6 – Implementação do método `preExam`

```
1 public PreExamResult preExam(Mat grayImg) throws StrabismusDetectorException {
2     Rect faceRect = faceExtractor.extract(grayImg);
3     EyesExtractorResult extractorResult;
4     Mat face = grayImg.submat(faceRect);
5     extractorResult = eyesExtractor.extract(face);
6
7     if (unalignedHead(extractorResult)) {
8         throw new UnalignedHeadException(new PreExamResult(faceRect, extractorResult));
9
10        PreExamResult ret = new PreExamResult(faceRect, extractorResult);
11    return ret;
12}
```

Este código é fundamental para auxiliar o usuário na captura da imagem correta para realização do exame. Na linha 2 é realizada a extração da região da face da imagem. Caso essa região não seja encontrada na imagem é lançada uma exceção.

Na linha 5 é extraída a região dos olhos da região da face extraída anteriormente. Caso essa região não seja encontrada na imagem é lançada uma exceção. Na linha 7, é verificado o alinhamento da região dos olhos.

Consiste em verificar se os retângulos, um para cada olho, gerados na etapa anterior estão alinhados. Com isso consegue-se validar se a cabeça está no ângulo correto para realização do exame.

Caso toda a validação realizada no método exposto no quadro 6 estiver de acordo, o software automaticamente irá capturar a imagem para a realização do exame. Este processo é feito através da implementação de uma *interface* de *callback* chamada `PictureCallback`. Esta *interface* possui o método `onPictureTaken` que é chamado quando tirado a foto. No quadro 7 é apresentado a implementação do método `onPictureTaken`.

Quadro 7 – Implementação do método `onPictureTaken`

```
1  @Override
2  public void onPictureTaken(final byte[] data, Camera camera) {
3      new Thread() {
4
5          @Override
6          public void run() {
7              final String resultStr;
8              ProgressDialog dialog = showProgressDialog("Aguarde realizando o exame...");
9              Mat img = ImageHelper.bitmapBytesToMat(data);
10             try {
11                 try {
12                     ExamResult result = examiner.makesExam(img);
13                     dialog.cancel();
14                     resultStr = result.toJson();
15                 } catch (Exception e) {
16                     dialog.cancel();
17                     showErrorDialog(e.getMessage());
18                     return;
19                 }
20             } finally {
21                 img.release();
22             }
23             // Action insert in ResultActivity
24             Intent intent = new Intent(activity, ResultActivity.class);
25             intent.setAction(Action.INSERT.name());
26             intent.putExtra(ResultActivity.EXAM_RESULT_IN, resultStr);
27             activity.startActivity(intent);
28         }
29     }.start();
30 }
31
```

Na linha 3 é criada uma *thread* onde será executado toda a lógica de captura de foto. Isso se faz necessário devido a câmera efetuar a chamada do método `onPictureTaken` da *thread* principal que é a responsável por efetuar a renderização da interface com usuário (ANDROID THREADS, 2014). Caso este código não execute em uma nova *thread* não é possível apresentar as mensagens devidas ao usuário.

Na linha 9 é convertido a imagem do formato *bitmap* para o formato nativo do OpenCV. Nas linhas 11 até a linha 14 é realizado o exame na imagem convertida e é capturado o resultado do mesmo.

Nas linhas 24 a 27 é criada e configurada a *Intent*. O objetivo dela é fazer a chamada a tela que apresenta o resultado, repassando as informações do exame.

3.3.2.2 Pré-processamento da imagem

Nesta etapa é realizado o pré-processamento da imagem vinda da etapa de aquisição. Tem como principal objetivo tornar a imagem processável computacionalmente e suavizar as bordas da imagem para facilitar a detecção das mesmas. No quadro 8 é apresentado o código que realiza o pré-processamento.

Quadro 8 – Código que realiza o pré-processamento

```
1 //Converte a imagem para tons de cinza
2 cvtColor(faceImage, grayFace, COLOR_RGB2GRAY);
3 //Suaviza as bordas
4 Imgproc.blur(grayFace, grayFace, new Size(11, 11));
```

Na segunda linha é feita a conversão da imagem para tons de cinza. O formato inicial da imagem vinda da etapa de aquisição é *Red Green Blue* (RGB). Neste formato a imagem possui 3 canais, um para cada cor, sendo que cada canal desse possui 8 bits, na imagem em questão.

O custo computacional para efetuar o processamento de imagens no formato *RGB* se torna maior visto que, em tons de cinza, a imagem possui apenas um canal de 8 bits. Outro ponto que faz necessário o uso de imagens em tons de cinza é que os métodos do OpenCV que são utilizados no presente trabalho apenas suportam imagens em tons de cinza.

Na quarta linha, há a chamada ao método `blur` do OpenCV. Este método se trata da implementação mais simplificada de um filtro de *low-pass* ou filtro de passa-baixa. Este filtro tem como objetivo reduzir a amplitude da variação dos pixels na imagem (LAGANIÈRE, 2011, p. 142).

O método `blur` em específico tem como objetivo suavizar a imagem substituindo cada pixel pelo valor médio. Este valor é calculado sobre um retângulo gerado sobre o pixel em questão e seus vizinhos (LAGANIÈRE, 2011, p. 143).

Esse retângulo, também chamado de *kernel*, no presente trabalho foi configurado com um tamanho de 11 por 11 pixels, como pode ser visto na linha 4 do quadro 10. Na figura 15 é apresentado o resultado do uso do método `blur` com um *kernel* de 30 por 30 para evidenciar os efeitos gerados.

Figura 15 – Resultado do uso do método `blur`

(a) Imagem original (b) Imagem passada no filtro

O uso da suavização da imagem se trata de um passo importante para posteriormente ser feito a extração das bordas. Existem outros filtros de passa-baixa mais elaborados como, por exemplo, o filtro gaussiano e o filtro de mediana (LAGANIÈRE, 2011, p. 145). Porém no presente trabalho após a realização de testes o filtro simples (`blur`) teve um melhor desempenho.

3.3.2.3 Extração da região da face e dos olhos

Nesta etapa é realizada a extração das regiões de interesse para o exame diretamente da imagem bruta. Tal extração, tanto da face quanto dos olhos, é realizada através da classe `CascadeClassifier` do OpenCV. Esta classe implementa um classificador em cascata Haar.

Toda a extração começa carregando os pesos que são obtidos através de um arquivo XML que é gerado através de uma fase de treinamento. No presente trabalho foram utilizados os arquivos previamente treinados disponibilizados pela biblioteca.

Os arquivos com os pesos utilizados foram `haarcascade_eye.xml` e `lbpcascade_frontalface.xml`. O primeiro possui os pesos para realizar a detecção da região dos olhos e o segundo possui os pesos para detectar a região da face. O segundo trabalha com números inteiros e possui um desempenho melhor.

A primeira etapa desta fase é extrair a face da imagem bruta. No quadro 9 é apresentado o código que realiza a extração.

Quadro 9 – Código que realiza a extração da região da face

```
1 public Rect extract(Mat frame_gray) {
2     MatOfRect rects = new MatOfRect();
3     int sizeFaceInImage = Math.round(frame_gray.width() * 0.5f);
4     classifier.detectMultiScale(frame_gray,
5                                 rects,
6                                 1.1,
7                                 2,
8                                 2,
9                                 new Size(sizeFaceInImage, sizeFaceInImage),
10                                new Size());
11
12    Rect[] faces = rects.toArray();
13    validateFaces(faces);
14    return faces[0];
15 }
```

Na linha 2 é criada uma matriz de retângulos que irá armazenar o resultado da detecção. Na linha 3 é calculado o tamanho mínimo esperado que a face tenha na imagem, que é de 50% da largura.

Na linha 4 é realizado a chamada a função `detectMultiScale` que é responsável por realizar a detecção da face. Das linhas 4 a linha 9 são passados os parâmetros para a função. O primeiro parâmetro é a imagem onde se deseja detectar a face. Esta imagem obrigatoriamente deve estar em tons de cinza e possuindo apenas um canal. O segundo parâmetro a matriz de retângulo criada anteriormente.

O terceiro parâmetro especifica o quanto o tamanho da imagem deve ser reduzido em cada escala. O quarto parâmetro especifica quantos vizinhos cada retângulo precisa ter para ser considerado válido. O quinto parâmetro é apenas para compatibilidade com versões antigas e não tem efeito algum. Para o valor dos parâmetros do terceiro ao quinto foram utilizados valores padrões fornecidos pela biblioteca.

Os dois últimos parâmetros da função `detectMultiScale` definem o tamanho da face que se deseja encontrar na imagem. Como tamanho mínimo definiu-se que pelo menos 50% da imagem deve ser composta pela face. Como tamanho máximo deixou-se em aberto para trazer qualquer tamanho que seja acima do mínimo definido.

Na linha 13 é realizado a chamada ao método `validadeFaces`. Este método tem como objetivo validar que o retorno do método `detectMultiScale` seja apenas uma face. Por fim, é retornado o retângulo que representa a região da face encontrada.

Depois de realizada a extração da região da face tem-se início a última etapa desta fase que é a extração da região dos olhos. Esta etapa tem como entrada o recorte da imagem da face que foi encontrada na etapa anterior. Com isso, além de ter um significativo ganho de desempenho visto que a imagem da face é menor que a imagem bruta, tem-se a certeza que se está procurando olhos em uma face.

A extração da região dos olhos também é realizada através da chamada do método `detectMultiScale` da classe `CascadeClassifier`. No quadro 10 é apresentado o código que realiza tal extração.

Quadro 10 – Código que inicia a extração da região dos olhos

```
1 public EyesExtractorResult extract(Mat grayFace) {
2     int newY = (int) (grayFace.height() * 0.25);
3     int newHeight = (int) (grayFace.height() * 0.25);
4     grayFace = grayFace.submat(new Rect(0, newY, grayFace.width(), newHeight));
5     return doExtract(grayFace, newY);
6 }
```

Nas linhas 2 e 3 é recalculado um novo tamanho de altura para imagem. Como o objetivo é extrair a região dos olhos é realizado um corte na imagem da face com objetivo de deixar apenas a região dos olhos. Na figura 16 é apresentado um exemplo da região que será analisada pelo classificador extraída diretamente do algoritmo.

Figura 16 – Corte para extração da região dos olhos

Depois de realizar o corte na imagem, apresentado na figura 16, é realizada a chamada do método `doExtract`. Este método é o responsável por realizar a extração de fato dos olhos do recorte da face feito anteriormente. No quadro 11 é apresentado o código do método `doExtract` de forma sintetizada.

Quadro 11 – Código que realiza a extração da região dos olhos

```
 1 private EyesExtractorResult doExtract(Mat grayFace, int newY) {
 2     double minVal = grayFace.height() * 0.25;
 3     MatOfRect eyes = new MatOfRect();
 4     Rect[] rects;
 5     classificator.detectMultiScale(grayFace,
 6                                    eyes,
 7                                    1.1,
 8                                    2,
 9                                    2,
10                                    new Size(minVal, minVal), new Size());
11
12     rects = eyes.toArray();
13     if (rects.length < 2) {
14         throw new EyesExtractorException("Eyes not found");
15     }
16
17     Rect leftRect;
18     Rect rightRect;
19     if (rects[0].x > rects[1].x) {
20         leftRect = rects[0];
21         rightRect = rects[1];
22     } else {
23         leftRect = rects[1];
24         rightRect = rects[0];
25     }
26
27     Rect realRightRect = rightRect.clone();
28     realRightRect.y += newY;
29
30     Rect realLeftRect = leftRect.clone();
31     realLeftRect.y += newY;
32
33     return new EyesExtractorResult(realLeftRect, realRightRect);
34 }
```

Na linha 2 é realizado o cálculo para identificar o tamanho mínimo para região dos olhos. Este tamanho deve ser de 25% da altura da imagem. Na linha 3 é criada uma matriz de retângulos que irá armazenar o resultado da detecção.

Na linha 5 é realizada a chamada ao método `detectMultiScale`. O primeiro parâmetro é a imagem onde se deseja detectar os olhos, já o segundo parâmetro é a matriz de retângulo criada anteriormente.

O terceiro parâmetro especifica o quanto o tamanho da imagem deve ser reduzido em cada escala. O quarto parâmetro especifica quantos vizinhos cada retângulo precisa ter para ser considerado válido. O quinto parâmetro é apenas para compatibilidade com versões antigas e não tem efeito algum. Para o valor dos parâmetros do terceiro ao quinto foram utilizados valores padrões fornecidos pela biblioteca.

Os dois últimos parâmetros da função `detectMultiScale` definem o tamanho de cada região do olho que se deseja encontrar na imagem. Como tamanho mínimo definiu-se que

pelo menos 25% da imagem deve ser composta por cada olho, com isso os dois olhos juntos devem compor pelo menos metade da imagem. Como tamanho máximo deixou-se em aberto para trazer qualquer tamanho que seja acima do mínimo definido.

Nas linhas 13 a 15 é realizada uma validação para garantir que o classificador encontrou os dois olhos. Caso não tenha encontrado, é lançada uma exceção e todo o algoritmo é abortado.

Nas linhas 19 a 25 é verificada dentro dos retângulos retornados da detecção, a direção de cada um deles. Para definir qual representa o olho esquerdo e qual representa o olho direito, é verificada sua posição em relação à imagem.

Nas linhas 27 a 31 é aplicado o valor de `Y` retirado anteriormente quanto foi realizado o recorte da imagem da face. Isso se faz necessário para que os retângulos fiquem com suas posições relativas à face e não ao recorte feito. Por fim, na linha 33 é montado o resultado com os dois retângulos que representam os olhos direito e esquerdo.

3.3.2.4 Extração da região do limbo e do brilho

Nesta fase é realizada a extração das regiões fundamentais para realização do teste de Hirshberg. Esta extração, tanto do brilho quanto do limbo, é realizada através do uso da transformada de Hough aplicada à detecção de círculos. Todo o processo realizado nesta fase é aplicado em cada olho separadamente.

A primeira etapa desta fase é fazer a extração da região do limbo. A entrada do método que realiza a extração do limbo é a imagem obtida na fase de extração da região dos olhos, explicada na seção anterior. No quadro 12 é apresentado o código que realiza a extração do limbo.

Quadro 12 – Código que realiza a extração da região do limbo

```
1  public static Circle extractLimbo(Mat eyeRaw) {
2      double minimum_votes = 25;
3      int width = eyeRaw.width();
4      int min_radius = round(width * 0.14);
5      int max_radius = round(width * 0.23);
6      CutAngle[] cutAngles = new CutAngle[] { new CutAngle(40, 140),
7                                              new CutAngle(220, 320)
8                                            };
9
10     PrecisionHoughTransform hough = new PrecisionHoughTransform();
11
12     List<Circle> limboCircles = hough.houghCircle(eyeRaw,
13                                                   55,
14                                                   minimum_votes,
15                                                   min_radius,
16                                                   max_radius,
17                                                   cutAngles);
18
19     if (limboCircles.size() == 0) {
20         throw new StrabismusDetectorException("Limbo not found.");
21     }
22     Circle limboCircle = limboCircles.get(0);
23     return limboCircle;
24 }
```

Nas linhas 4 e 5 são calculados o mínimo e o máximo raio esperados. Nas linhas 6 a 8 são criados os pontos de corte. Na linha 12 é realizada a chamada da função `houghCircle` que é a responsável por realizar a extração da região do limbo.

Nas linhas 12 a 17 são passados os parâmetros para a execução da transformada. O primeiro parâmetro é a imagem que se deseja extrair os círculos. O segundo parâmetro é limiar usado para aplicar o detector de bordas.

O terceiro parâmetro é a quantidade mínima de votos que um círculo deve possuir no espaço de Hough para ser considerado um círculo. No quarto e quinto parâmetros são passados o limite mínimo e máximo de raio esperados. No último parâmetro é passado os ângulos de corte especificados anteriormente.

Nas linhas 19 a 21 é garantido que foi encontrado algum círculo, caso contrário assume-se que não foi possível encontrar a região do limbo.

Depois de realizado a extração da região do limbo a última etapa desta fase é extrair a região do brilho. Esta região se trata do brilho gerado pelo *flash* disparado no momento em que é capturada a imagem.

No quadro 13 é apresentado o código responsável por realizar a extração da região do brilho. A imagem de entrada para o método de extração do brilho é a extraída na etapa de extração do limbo.

Quadro 13 – Código que realiza a extração da região do brilho

```
1 public Circle extractBrightness(Mat limbo) {
2     brightnessCircles = hough.houghCircle(limbo,
3                                           55,
4                                           25,
5                                           4,
6                                           12);
7     if (brightnessCircles.size() == 0) {
8         throw new StrabismusDetectorException("brightness not found.");
9     }
10    return brightnessCircles.get(0);
11 }
```

Na linha 2 da função `houghCircle` que é a responsável por realizar a extração da região do brilho. Nas linhas 2 a 6 são passados os parâmetros necessários para execução da transformada. O primeiro parâmetro é a imagem de onde se deseja extrair os círculos. O segundo parâmetro é limiar usado para aplicar o detector de bordas.

O terceiro parâmetro é a quantidade mínima de votos que um círculo deve possuir no espaço de Hough para ser considerado um círculo válido. O quarto e quinto parâmetro definem o raio que deve possuir o tamanho do brilho esperado.

Diferentemente da extração do limbo, não são configurados pontos de corte. Com isso, serão analisados os 360 graus que compõem o círculo. Também é configurado o raio mínimo e máximo com valores fixos, visto que nesse caso diferentes resoluções não afetaram significativamente o tamanho do brilho.

Nas linhas 7 a 9 é garantido que foi encontrada a região do brilho. Caso não tenha sido encontrada, é lançada uma exceção e parado a execução de todo o processo.

3.3.2.5 Detecção do estrabismo

Nesta fase são realizados os cálculos em cima das regiões do limbo e do brilho extraídas pela fase anterior. No quadro 14 é apresentado o código responsável por dar início à detecção do estrabismo.

Quadro 14 – Método makesExam

```
1  public ExamResult makesExam(EyeROI leftEye, EyeROI rightEye) {
2      float leftRadius = leftEye.getLimbo().getRadius();
3      float rightRadius = rightEye.getLimbo().getRadius();
4      float proportionalityConstR = 1, proportionalityConstL = 1;
5
6      if (leftRadius > rightRadius) {
7          proportionalityConstL = rightRadius / leftRadius;
8      } else if (rightRadius > leftRadius) {
9          proportionalityConstR = leftRadius / rightRadius;
10     }
11
12     ExamEyeResult rightResult = examineEye(rightEye, proportionalityConstR);
13     ExamEyeResult leftResult = examineEye(leftEye, proportionalityConstL);
14     return new ExamResult(rightResult, leftResult);
15 }
```

O principal objetivo deste método é fazer o cálculo de proporcionalidade entre os tamanhos dos raios da região do limbo do olho esquerdo em relação ao olho direito. Existindo diferença entre o tamanho dos raios, é realizada a divisão do raio menor pelo raio maior. Este cálculo, chamado de cálculo de proporcionalidade corneana, tem como objetivo encontrar o valor proporcional a 1 *pixel* do raio menor no raio maior.

Segundo Almeida (2010, p. 73) através do cálculo da proporcionalidade é possível mensurar o posicionamento do brilho levando-se em consideração diferenças de erros refracionais. Isto porque, qualquer incremento ou diminuição no tamanho da córnea, dado artificialmente por uma lente corretora de equivalente esférico elevado, será descontado ao se calcular a proporcionalidade.

Depois de realizado o cálculo de proporcionalidade é realizado o exame. Isto é feito nas linhas 12 e 13 com a chamada do método `examineEye`. No quadro 15 é apresentado o código, de forma sintetizada, do método `examineEye`.

Quadro 15 – Método `examineEye`

```
1  private static ExamEyeResult examineEye(EyeROI roi, float proportionalityConst) {
2      Point brightnessPoint = roi.getBrightness().getCenter();
3      Point limboPoint = roi.getLimbo().getCenter();
4
5      double brightnessX = brightnessPoint.x;
6      double limboX = limboPoint.x;
7      DeflectionDirection direction = DeflectionDirection.ALRIGHT;
8      double deflection = Math.abs(brightnessX - limboX) * proportionalityConst;
9      if (deflection > 2.0) {
10         direction = brightnessX < limboX ? DeflectionDirection.LEFT :
11                                            DeflectionDirection.RIGHT;
12     } else {
13         double brightnessY = brightnessPoint.y;
14         double limboY = limboPoint.y;
15         deflection = Math.abs(brightnessY - limboY) * proportionalityConst;
16         if (deflection > 2.0) {
17             direction = brightnessY > limboY ? DeflectionDirection.TOP :
18                                                DeflectionDirection.DOWN;
19         }
20     }
21     return new ExamEyeResult(roi.getLimbo().getRadius(), deflection,
22                              direction, proportionalityConst);
23 }
```

Neste método é realizada a detecção do estrabismo de fato. Para tal, é avaliada a diferença entre o centro do limbo para o centro do brilho nas direções esquerda, direita, acima e abaixo.

Na linha 8 é feito o cálculo da diferença na horizontal por considerando o eixo *X*. Neste cálculo é considerado a proporcionalidade corneana. Nas linhas 9 a 11 é avaliado se há desvio: caso exista é feito a verificação da direção do mesmo, que neste caso pode ser para direita ou para esquerda.

Na linha 15 é feito o cálculo da diferença na vertical por considerar o eixo *Y*. Neste cálculo também é considerada a proporcionalidade corneana. Nas linhas 16 a 18 é avaliado se há desvio: caso exista é feita a verificação da direção do mesmo, que neste caso pode ser acima ou abaixo.

Segundo Almeida (2010, p. 74), na aplicação do método de Hirschberg é apresentado uma diferença de 2 a 4 graus do eixo visual ao eixo anatômico, podendo ocasionar uma impressão de falso desvio. Com isso, como pode ser observado nas linhas 9 e 16, foi usada uma tolerância de 2 *pixels* para ser considerado um desvio.

3.3.3 Implementação da transformada de Hough

Nesta seção será apresentada a implementação que foi realizada da transformada de Hough. Esta implementação foi parte fundamental para realização do trabalho, visto que as regiões do olho mais relevantes para execução do teste de Hirshberg foram extraídas através desta técnica.

A construção do algoritmo usou como base teórica os artigos de Duda e Hart (1972), Rhody (2005) e a construção do algoritmo realizada por Pistori e Costa (2009). O algoritmo pode ser dividido em 4 etapas principais. Todo o processamento da transformada é iniciado pelo método `houghCircle` que é apresentado no quadro 16.

Quadro 16 – Método `houghCircle`

```
1  public List<Circle> houghCircle(Mat gray,
2                                  double cany_high_threshold,
3                                  double minimum_votes,
4                                  int min_radius,
5                                  int max_radius,
6                                  CutAngle[] cutAngles) {
7      Mat edges = new Mat();
8
9      Imgproc.Canny(gray,
10                   edges,
11                   cany_high_threshold * 0.5,
12                   cany_high_threshold);
13
14     return hougTransform(edges, minimum_votes, min_radius, max_radius, cutAngles);
15 }
```

A primeira etapa do algoritmo é realizada neste método que é a aplicação do detector de bordas de Canny. Para tal, é utilizado a implementação fornecida pela biblioteca OpenCV do detector de borda. Nas linhas 9 a 12 é realizado a chamada ao método da biblioteca.

O primeiro parâmetro é a imagem que será aplicada a detecção de bordas. Esta imagem deve estar em tons de cinza e ter passado por algum processo de suavização de bordas para que o algoritimo seja mais eficiente. O processo de suavização da imagem é realizado na fase de pré-processamento através do método `blur`, conforme já descrito na seção de Pré-processamento da imagem.

O segundo parâmetro é o resultado da operação do algoritmo. Se trata de uma imagem binária onde os *pixels* de borda possuem o valor 1 e os *pixels* que não representam borda com o valor 0. Em cima desta imagem gerada pelo processo do detector é realizado todo o processamento da transformada de Hough.

Os dois últimos parâmetros indicam o limiar minimo e máximo que devem ser usados pelo algoritimo de Canny. Para esta implementação foi seguido um padrão utlizado pela biblioteca OpenCV que é definir o limiar mais alto e o limiar baixo possuir 50% do valor do limiar alto.

Depois da primeira etapa que é realizar a extração das bordas através do algoritmo de Canny passa-se para a segunda etapa. Esta etapa consiste em gerar uma espécie de tabela de consulta para simplificar o processamento do algoritmo posteriormente. No quadro 17 é apresentado o código, de forma sintetizada, que constrói esta tabela.

Quadro 17 – Método `buildLookUpTable`

```
1 private void buildLookUpTable(int radiusMin, int radiusMax, CutAngle[] cutAngles) {
2     int depth = radiusMax - radiusMin;
3     lut = new int[2][360][depth];
4     int[] lutSize = new int[depth];
5     for (int radius = radiusMin; radius < radiusMax; radius++) {
6         int i = 0;
7         int indexR = radius - radiusMin;
8         for (CutAngle cutAngle : cutAngles) {
9             for (int angle = cutAngle.startAngle; angle < cutAngle.endAngle; angle++) {
10                double angleRadians = Math.toRadians(angle);
11                int rcos = round(radius * Math.cos(angleRadians));
12                int rsin = round(radius * Math.sin(angleRadians));
13                lut[0][i][indexR] = rcos;
14                lut[1][i][indexR] = rsin;
15                i++;
16            }
17        }
18    }
19}
```

Na linha 2 é feito o cálculo da profundidade da tabela. Esse valor é a diferença entre o raio mínimo e o raio máximo. Na linha 3 é definida a tabela de consulta. Esta tabela possui 3 dimensões e tem como objetivo armazenar parte da função paramétrica do círculo. Na figura 17 é apresentada função paramétrica do círculo.

Figura 17 – Equação paramétrica do círculo

$$a = x - r\cos\theta$$
$$b = y - r\ \sin\theta$$

Fonte: Rhody (2005, p. 1).

A primeira dimensão corresponde ao cálculo do seno e cosseno. A segunda dimensão serve para cobrir os 360 graus do círculo. A última dimensão cobre o raio que será analisado, que será do raio mínimo ao raio máximo.

Nas linhas 5 a 16 é realizada a iteração para a montagem da tabela. Para cada raio, do raio mínimo ao raio máximo, é avaliado todos os ângulos de cortes definidos. Para cada ângulo de corte, do ângulo inicial até o ângulo final, definido por este ângulo de corte, é feito o cálculo de parte da equação paramétrica do círculo. Este cálculo é apresentado nas linhas 11 e 12. A parte da função paramétrica que é armazenada na tabela é apresentada na figura 18.

Figura 18 – Equação paramétrica do círculo

$$rcos = r\cos\theta$$
$$rsin = r\ \sin\theta$$

A terceira etapa consiste em gerar o espaço de Hough. Para isso é utilizada a imagem binária gerada na primeira etapa do algoritmo e a tabela de consulta gerada na segunda etapa.

No quadro 18 é apresentado o código de forma simplificada da função responsável por gerar o espaço de Hough.

Quadro 18 – Método `generateHoughSpace`

```
1 private double[][][] generateHoughSpace(Mat edges, int minRadius, int maxRadius) {
2     int width =  edges.width();
3     int height = edges.height();
4     int depth = maxRadius - minRadius;
5     byte imageValues[] = new byte[(int) (edges.total() * edges.channels())];
6     edges.get(0, 0, imageValues);
7
8     double[][][] houghSpace = new double[width][height][depth];
9
10    for (int y = 0; y < height; y++) {
11        for (int x = 0; x < width; x++) {
12            if (imageValues[(y * width) + x] != 0) {
13                for (int radius = minRadius; radius < maxRadius; radius++) {
14                    int indexR = radius - minRadius;
15                    for (int i = 0; i < 360; i++) {
16                        int a = x + lut[1][i][indexR];
17                        int b = y + lut[0][i][indexR];
18                        if ((b >= 0) && (b < height) && (a >= 0) && (a < width)) {
19                            houghSpace[a][b][indexR]++;
20                        }
21                    }
22                }
23            }
24        }
25    }
26    return houghSpace;
27 }
```

Na linha 4 é feito o cálculo da quantidade de raios que será analisada. Na linha 5 é criada o vetor de *bytes*, que conterá o conteúdo da imagem. Na linha 6 é carregado o vetor de *bytes* fazendo uma chamada nativa para OpenCV. Na linha 8 é criado o espaço de Hough com suas 3 dimensões necessárias. As duas primeiras representam o *X* e *Y* do ponto que representa o centro do círculo e a última dimensão representa o intervalo de raio que será analisado.

Nas linhas 10 a 25 são percorridos todos os *pixels* da imagem. Para cada *pixel* de borda, que é todo o *pixel* que possui o valor 1, analisa todos os raios, do raio mínimo até o máximo.

Para cada raio analisado é percorrido todos os 360 graus da tabela de pesquisa e finaliza o cálculo da equação paramétrica. Com isso o valor das variáveis *a* e *b* representam o valor *X* e *Y*, coordenadas que representam o centro do provável círculo na imagem real. Para este provável centro é incrementado um voto no espaço de Hough.

A quarta e última etapa do algoritmo consiste em analisar o espaço de Hough e extrair dele todos os círculos que respeitarem a quantidade mínima de votos. No quadro 19 é apresentado o código (de forma sintetizada) responsável por fazer a extração dos círculos do espaço de Hough.

Quadro 19 – Método `checkValidCircles`

```
1 private List<Circle> checkValidCircles(int minRadius, int maxRadius,
2                                         double minimum_votes,
3                                         int width, int height,
4                                         double[][][] houghSpace) {
5     List<Circle> ret = new ArrayList<Circle>();
6     int radiusOfMax = minRadius;
7     for (int y = 0; y < height; y++) {
8         for (int x = 0; x < width; x++) {
9             for (int radius = minRadius; radius < maxRadius; radius++) {
10                int indexR = radius - minRadius;
11                if (houghSpace[x][y][indexR] > minimum_votes) {
12                    radiusOfMax = indexR;
13                    ret.add(new Circle(x, y, radiusOfMax + minRadius));
14                }
15            }
16        }
17    }
18    return ret;
19}
```

Nas linhas de 7 a 17 é percorrido todo o espaço de Hough. O objetivo deste método é verificar no espaço de Hough quais círculos encontrados possuem pelo menos o mínimo configurado de votos. Caso possua é criado um círculo representado pela posição *X* e *Y* correspondente ao centro do círculo e o raio do círculo.

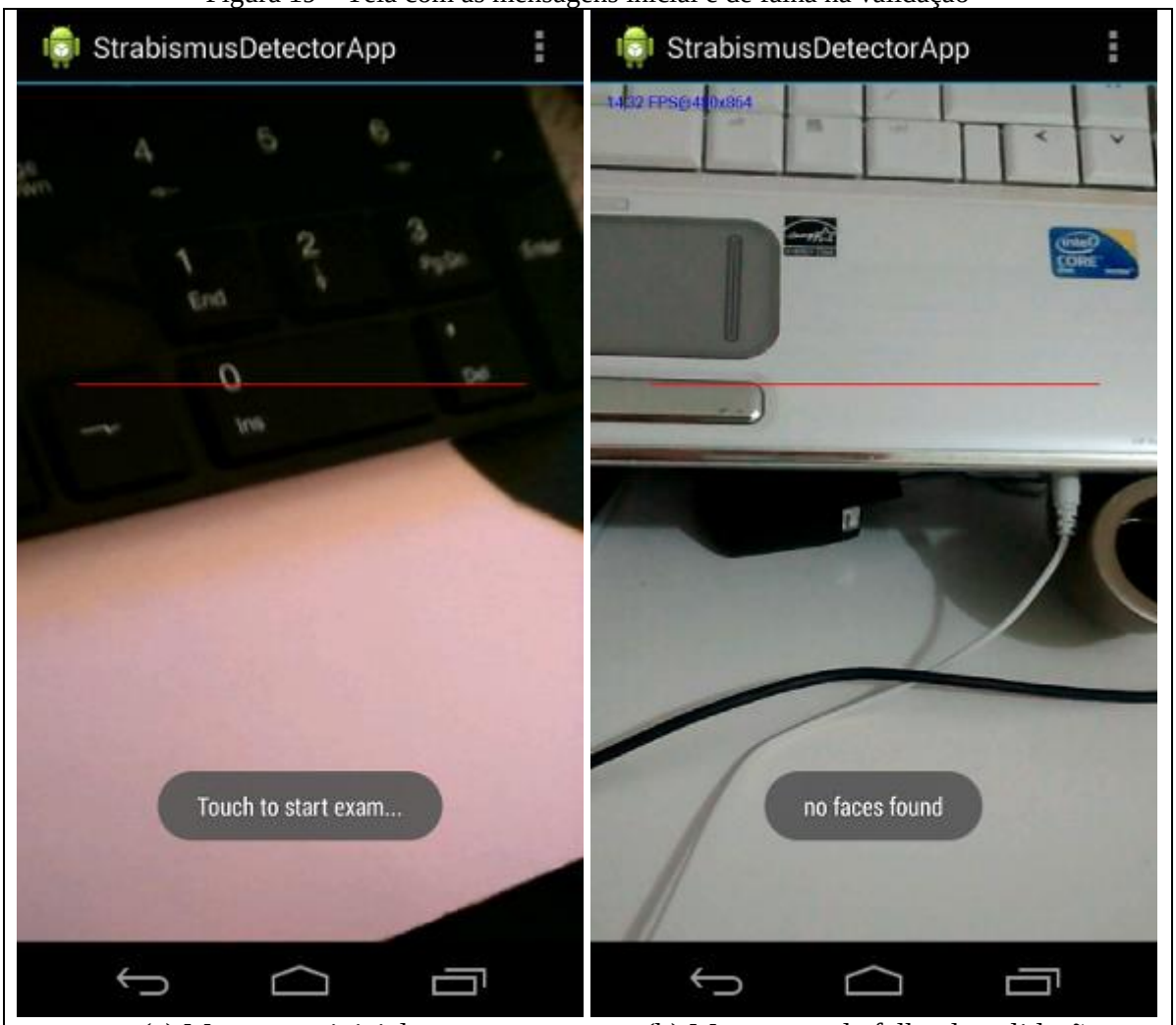
3.3.4 Operacionalidade da implementação

Nesta seção será apresentada toda a operacionalidade do protótipo. Ao iniciar a aplicação o usuário irá visualizar a tela principal do aplicativo. Nessa tela será apresentada a visualização da câmera para o usuário.

Na visualização da câmera sempre irá aparecer uma linha vermelha na tela com o objetivo de auxiliar o usuário a deixar a cabeça do paciente na posição correta. Ao entrar na tela o sistema apresenta uma mensagem avisando ao usuário para tocar na tela para iniciar o exame. Na figura 19a é apresentada a tela com a mensagem inicial.

Ao tocar na tela o sistema começa executar o pré-exame. Com isso serão apresentadas mensagens informando ao usuário os problemas encontrados nesta etapa. O usuário deve ir ajustando o posicionamento do dispositivo ou solicitar para o paciente, que está sendo examinado, ajustar sua posição para realizar o exame. Na figura 19b é apresentada a tela com a mensagem avisando o usuário o que há de errado para que ele ajuste.

Figura 19 – Tela com as mensagens inicial e de falha na validação

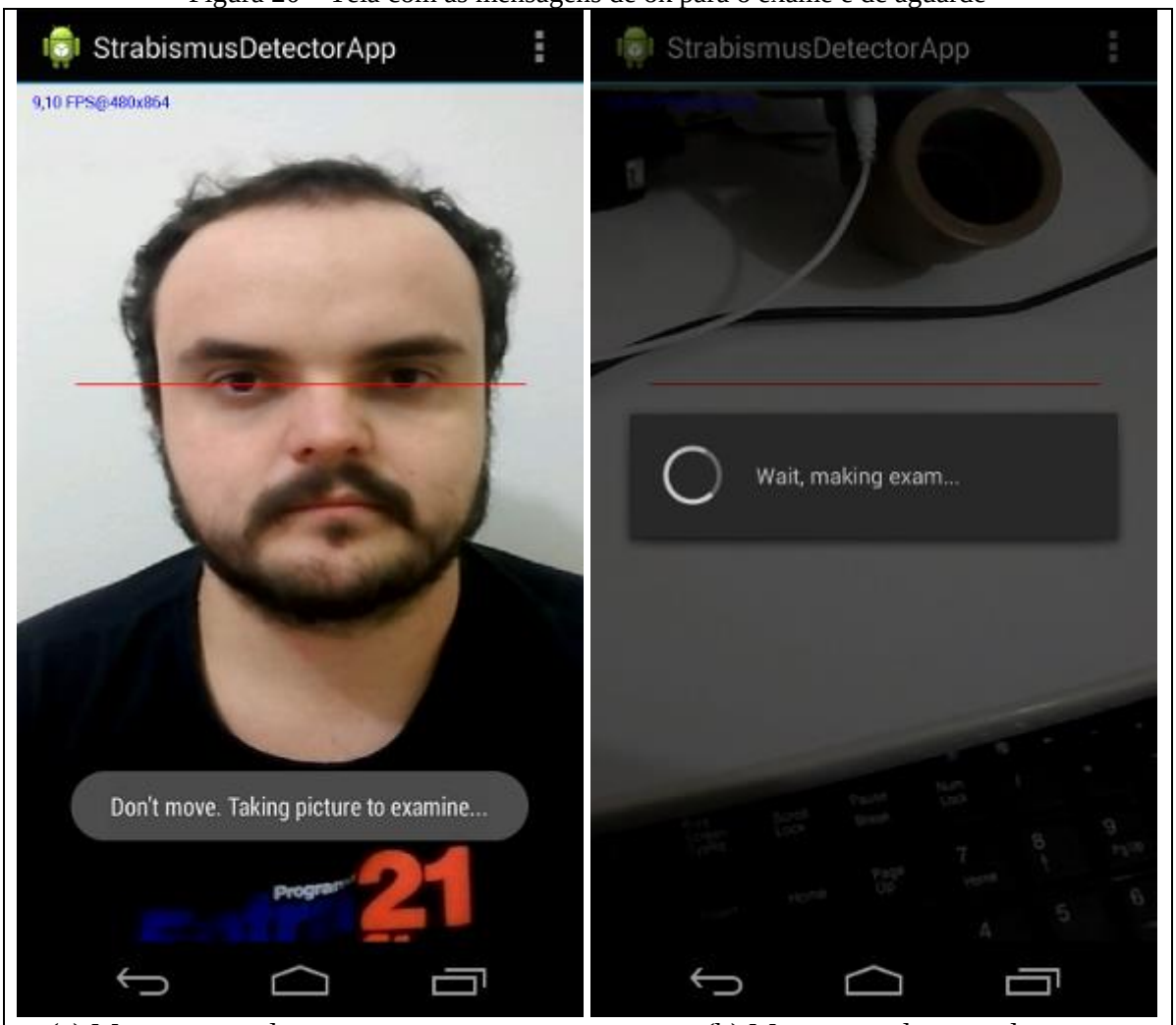


(a) Mensagem inicial (b) Mensagem de falha de validação

Quando o aplicativo detecta que a imagem está apta a ser capturada, ele apresenta a mensagem informando ao usuário que está tudo certo para o exame e que a imagem será capturada. Na figura 20a é apresentada a tela com a mensagem.

Após este processo é apresentada a tela, representada na figura 20b, informando ao usuário para que ele aguarde, pois o exame está sendo executado. Após o exame executado, é apresentada a tela com o resultado do exame.

Figura 20 – Tela com as mensagens de ok para o exame e de aguarde



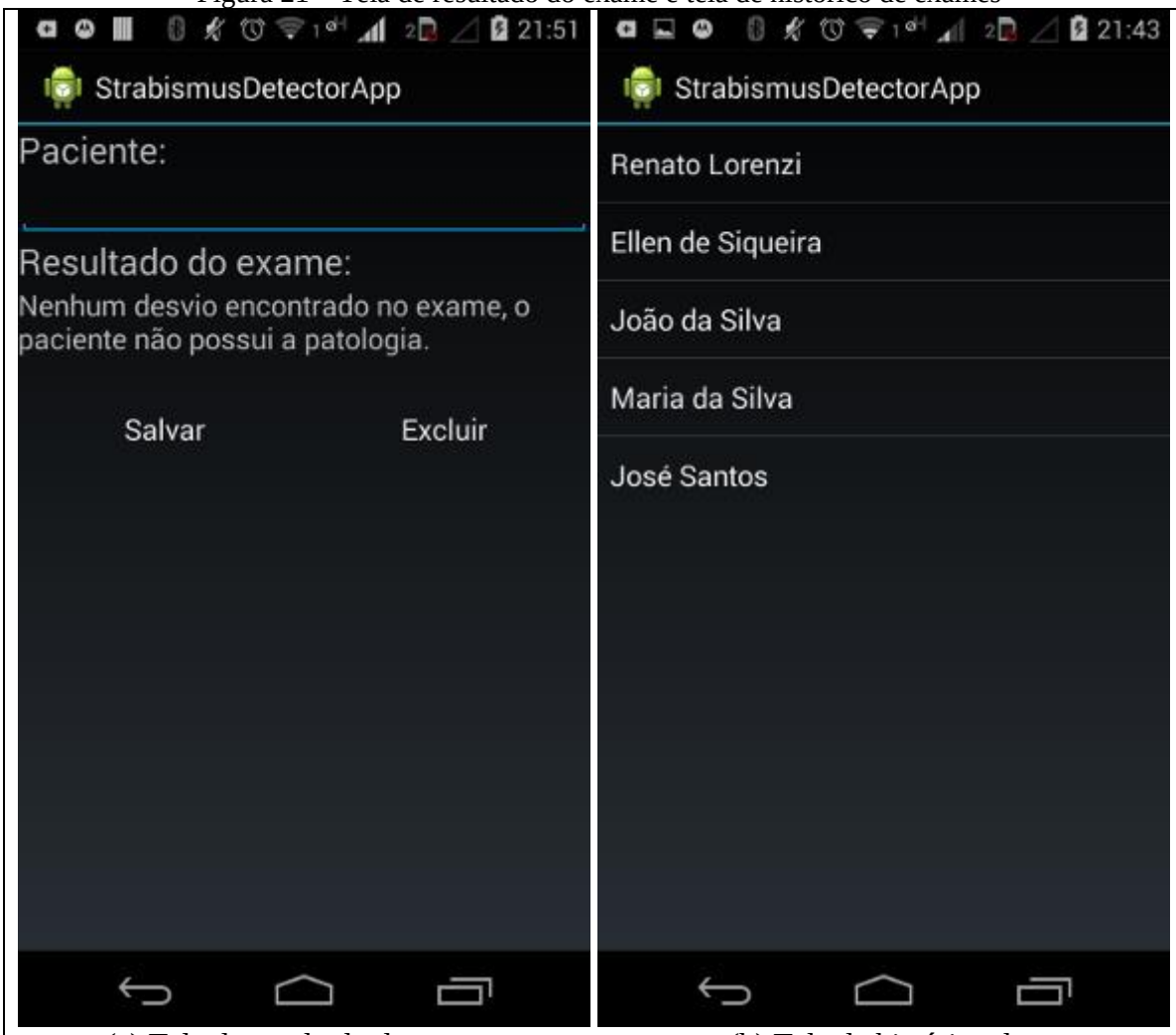
(a) Mensagem tudo certo para o exame (b) Mensagem de aguarde

Na tela que apresenta o resultado, apresentada na figura 21a, o usuário pode informar um nome para o paciente que realizou o exame, depois disso salvar ou descartar o resultado do exame realizado. Caso usuário salve, ele poderá consultar este resultado posteriormente na tela de histórico de exame.

Depois de o usuário salvar ou descartar o exame realizado, ele será redirecionado novamente para tela principal. Na tela principal será apresentada a mensagem para ele iniciar o exame novamente e todo processo se inicia novamente.

Na tela inicial do aplicativo a qualquer momento o usuário poderá selecionar o menu, que fica no topo da tela, para acessar a tela de histórico de execução. Nessa tela, representada pela figura 21b, o usuário poderá selecionar um exame existente para consulta ou também, alterar o nome do paciente.

Figura 21 – Tela de resultado do exame e tela de histórico de exames



(a) Tela de resultado do exame (b) Tela de histórico de exames

Ao selecionar um exame na lista será apresentada a tela que apresenta o resultado do exame. Nessa tela ele poderá alterar o nome do paciente, salvar ou excluir o exame em questão. Após finalizar a ação na tela de exibição do resultado do exame o usuário será redirecionado novamente para tela de histórico.

3.4 RESULTADOS E DISCUSSÃO

Nesta seção são apresentados os resultados dos experimentos realizados. Na subseção 3.4.1 serão apresentados os resultados levando em consideração o quesito precisão. Na subseção 3.4.2 são abordados os resultados obtidos nos testes realizados de consumo de memória da aplicação. Na subseção 3.4.3 serão apresentados os resultados obtidos nos testes de desempenho de cada fase do processo de detecção do estrabismo. Para finalizar, na subseção 3.4.4 são discutidas as principais diferenças entre o presente trabalho e os trabalhos correlatos.

3.4.1 Precisão

Nesta seção serão apresentados os resultados no quesito de precisão obtidos pelo aplicativo. Serão abordados os resultados das fases de extração da face e da região dos olhos, da fase de extração do limbo e do brilho e por fim da detecção do estrabismo.

Para realizar os testes de precisão foram coletadas amostras em campo com autorização dos participantes. Foram conseguidas 54 amostras de 23 pessoas. Procurou-se obter o maior número de pacientes que possuíssem estrabismo, porém foram conseguidas 3 pessoas.

Além das 54 amostras extraídas das 23 pessoas diferentes, também foram utilizadas mais 56 amostras de 4 pessoas em condições de luminosidade diferente para validar as fases de extração do algoritmo. Estas amostras das mesmas pessoas foram cruciais para a calibração das fases de extração de região de interesse (mais detalhes na seção 3.4.1.2).

Para realizar a captura das amostras foi utilizado o aplicativo desenvolvido com o intuito de testá-lo e até validar a etapa de pré-exame desenvolvida. O uso do aplicativo foi fundamental para a correta captura das amostras.

Como dispositivo móvel para realizar a captura das imagens foi utilizado o *smartphone* Motorola Moto G que possui uma câmera de 5 megapixels e Android versão 4.4.2. As amostras capturadas possuem uma resolução de 1944 x 2592 *pixels*.

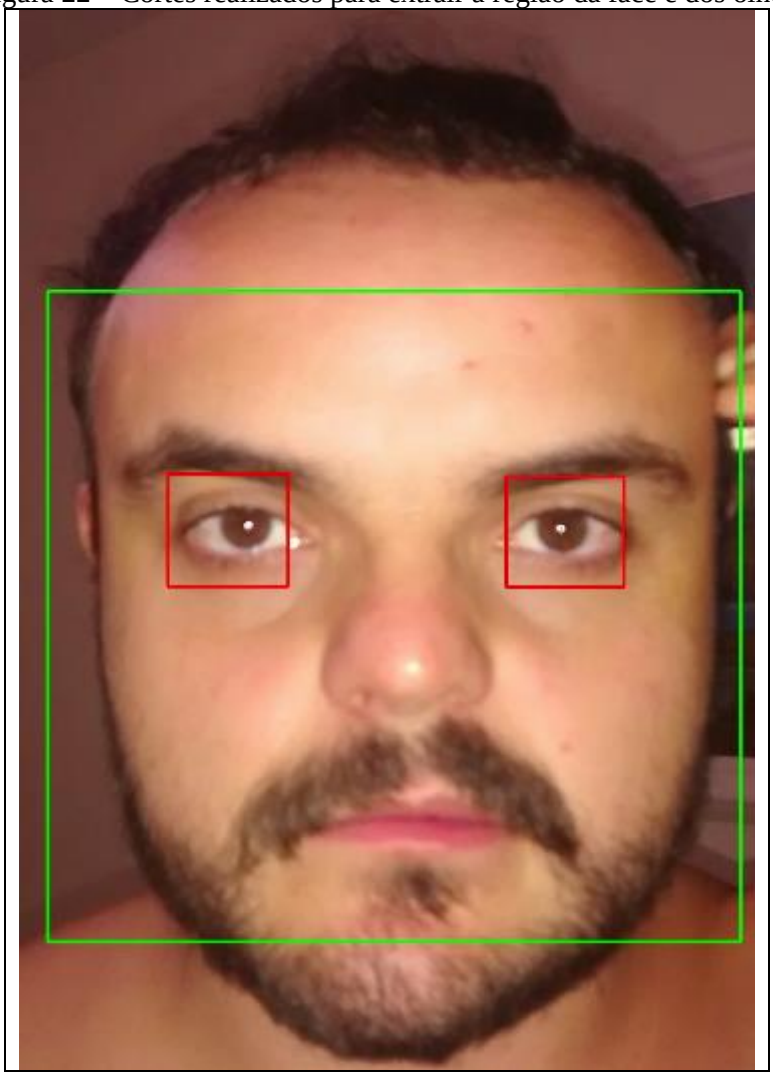
3.4.1.1 Precisão na extração da região da face e dos olhos

Na fase de extração da região da face e dos olhos foram utilizadas as 110 amostras disponíveis para efetuar os testes. O resultado das duas extrações atingiu 100% do objetivo no quesito precisão.

Este resultado foi obtido principalmente pelo exame proporcionar um ambiente controlado. Outro ponto que contribui com o sucesso desta fase foi o pré-exame realizado pelo aplicativo na fase de aquisição da imagem.

No pré-exame a foto só é tirada caso a região da face e dos olhos tenha sido encontrada. Na figura 22 é apresentado um exemplo da região que será extraída. Na cor verde é apresentada a região da face e na cor vermelha a região dos olhos, ambas detectadas pelo algoritmo.

Figura 22 – Cortes realizados para extrair a região da face e dos olhos



3.4.1.2 Precisão na extração da região do limbo e do brilho

Nesta seção são apresentados os resultados da extração da região do limbo e do brilho. Na etapa de extração da região do limbo foi alcançada uma precisão de 95.45%, sendo que das 220 amostras (110 faces x 2 olhos), apenas 10 não foram extraídas corretamente.

Nesta etapa houve alguns empecilhos. O primeiro deles foi em relação ao uso da transformada de Hough. Inicialmente foi realizado testes utilizando a implementação padrão disponibilizada pelo OpenCV. Esta implementação possui um ótimo desempenho sendo até duas vezes mais rápida do que a implementação realizada no presente trabalho (dados comprovados nos testes realizados).

O problema encontrado na implementação padrão do OpenCV é que a mesma não permite que sejam definidos ângulos de corte para análise. Isto foi fundamental para o resultado obtido na extração da região do limbo, visto que são analisados apenas os ângulos de interesse.

Outro empecilho foi definir o limiar passado para o operador de Canny. O operador de Canny consiste em parte essencial para o sucesso da execução da transformada de Hough, pois todo o processamento realizado pela transformada é realizado baseado nas bordas encontradas pelo operador de Canny.

Para solucionar este problema foram efetuados testes utilizando o método de Otsu (1979). Este método tem como objetivo, através do histograma da imagem, encontrar o melhor valor para o limiar, do ponto de vista estatístico, e também é o método mais estável para tal realização atualmente (FANG;YUE;YU, 2009, p. 109).

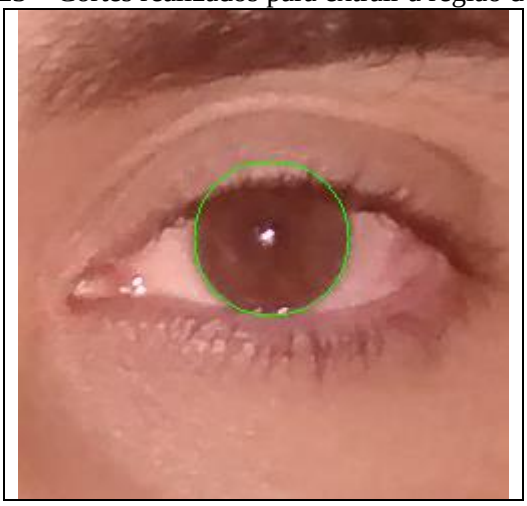
Porém utilizando este método o algoritmo atingiu apenas 72.72% de precisão. Com este resultado decidiu-se por usar o limiar configurado manualmente e fixo para todas as imagens, que possui o valor de 55.

Os testes realizados para se chegar ao limiar foram efetuados visualmente. Para cada uma das 220 imagens foi testado o limiar em uma faixa de 40 a 100 e para cada valor testado foi analisado todas as imagens contando as imagens onde a região de interesse era retornada com sucesso. Com isso o limiar de valor 55 foi o que retornou a maior quantidade de acertos.

Foram realizados testes com um limiar maior do que 100 e menor do que 40. O resultado foi uma perda na definição da região das bordas de interesse, para o limiar maior e uma quantidade alta de ruídos na imagem para o limiar menor.

Na figura 23 é apresentado um exemplo da região que será extraída. Na cor verde é apresentada a região do limbo detectada pelo algoritmo, dentro da região do olho extraída na etapa descrita anteriormente.

Figura 23 – Cortes realizados para extrair a região do limbo



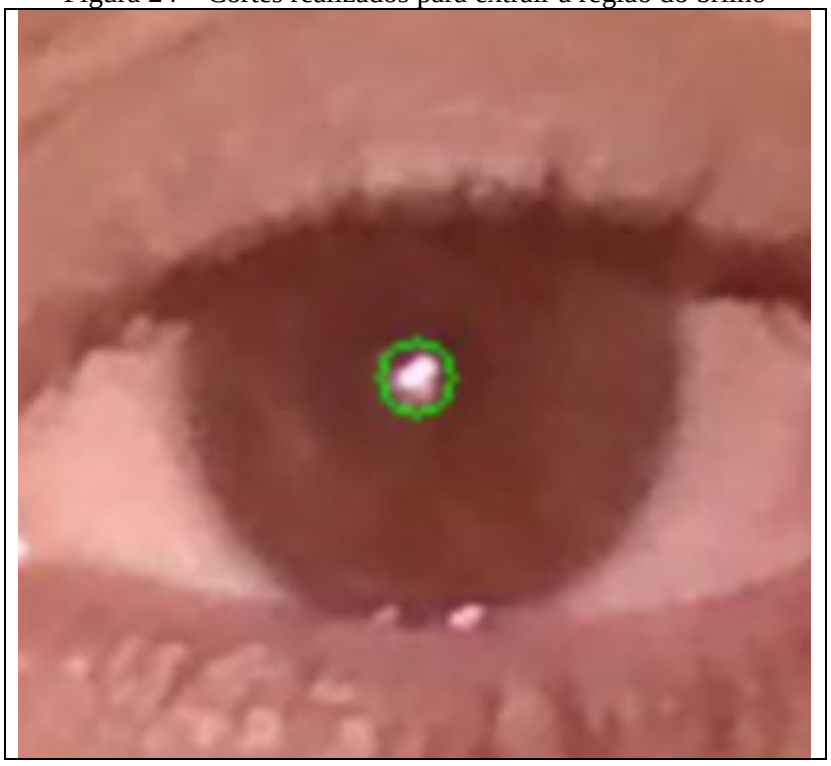
Depois de apresentado o resultado da extração da região do limbo será apresentado os resultados obtidos na extração do brilho gerado pelo *flash* da câmera. Nesta etapa foi

alcançada uma precisão de 96.36%, sendo que das 220 amostras, apenas 8 não foram extraídas corretamente.

Nesta etapa o principal desafio foi em relação a conseguir eliminar o ruído deixado pelo operador de Canny sem perder a região de interesse. Para solucionar o problema foram realizados os mesmo testes realizados para definição do limiar da extração da região do brilho citada acima. O valor do limiar que obteve o melhor resultado foi 55. Com esse valor foi possível reduzir o ruído da imagem sem perder a precisão da borda da região de interesse.

Na figura 24 é apresentado um exemplo da região que será extraída. Na cor verde é apresentada a região do brilho detectado pelo algoritmo. A imagem foi ampliada em 2 vezes para facilitar a visualização.

Figura 24 – Cortes realizados para extrair a região do brilho



3.4.1.3 Precisão na detecção do estrabismo

Nesta seção são apresentados os resultados obtidos na detecção do estrabismo pela técnica proposta. Para tal, foram utilizadas as 52 amostras de 21 pessoas diferentes, sendo que destas, 3 possuem um laudo oftalmológico que comprova possuir a patologia.

Como o objetivo é validar a eficiência da detecção do estrabismo, foram excluídas duas pessoas das 23, que não possuem estrabismo, do grupo de amostras analisadas. Isto se fez necessário tendo em vista que destas duas pessoas foi tirado apenas uma amostra cada uma e esta amostra não teve sucesso na fase de extração do limbo.

Como o objetivo da ferramenta é ser algo de fácil uso sem a necessidade de um especialista e um ambiente estritamente controlado, foi capturada uma média de 2,47 amostras por pessoa. Foi capturado mais de uma amostra por pessoa com o objetivo aumentar o banco de imagens a se analisar.

O resultado gerado nesta etapa foi que das 52 amostras analisadas 40.38% delas obtiveram sucesso na realização do exame. Como foram capturadas mais de uma amostra por pessoa, esse percentual de 40.38% das que obtiveram sucesso cobriram 85.71% ou 18 dos pacientes.

Na tabela 1 são apresentados os resultados obtidos pela ferramenta em relação ao resultado esperado, onde o paciente possui o laudo oftalmológico que comprova possuir a patologia. A tabela é constituída por 3 colunas. A primeira coluna é dado um identificador ao paciente analisado. A segunda é dividida em duas subcolunas apresentando o desvio em *pixels* do olho esquerdo e do olho direito. A última coluna, também dividida em duas subcolunas, apresenta o resultado obtido pela ferramenta em relação ao resultado esperado.

Tabela 1 – Resultado obtido na detecção do estrabismo

	Desvio no olho		Estrabismo	
Paciente	Esquerdo	Direito	Técnica	Esperado
1	1.0	1.0	Não	Não
2	0.97	2.0	Não	Não
3	0.0	0.96	Não	Não
4	2.0	0.0	Não	Não
5	1.0	1.92	Não	Não
6	1.0	1.0	Não	Não
7	0.0	0.0	Não	Não
8	1.0	1.89	Não	Não
9	1.93	1.0	Não	Não
10	0.97	0.0	Não	Não
11	1.0	1.0	Não	Não
12	2.0	1.0	Não	Não
13	0.0	2.0	Não	Não
14	1.73	0.0	Não	Não
15	0.0	0.0	Não	Não
16	1.0	3.87	Sim	Sim
17	1.0	3.0	Sim	Sim
18	4.0	2.90	Sim	Sim
19	1.0	2.93	Sim	Não
20	3.0	2.99	Sim	Não
21	2.9	3.0	Sim	Não

Os pacientes 1 a 15 são os pacientes que não possuem a patologia e que a técnica teve o resultado esperado. Os pacientes 16 a 18 são os pacientes que possuem a patologia e a

técnica obteve o resultado esperado. Os 3 últimos pacientes foi onde a técnica não atendeu ao resultado esperado.

Nos 3 pacientes onde não foi obtido o diagnóstico correto o problema foi a forma que foram extraídas as amostras, onde as mesmas passaram com sucesso pelas etapas de extração do limbo e do brilho, porém o alinhamento da cabeça não estava correto. O desalinhamento da cabeça faz com que o brilho gerado pelo *flash* da câmera não fique posicionado na localização correta da região dos olhos, ocasionando em um falso positivo.

Para amenizar este efeito foi criada a etapa de pré-exame. Nela é verificado o alinhamento da cabeça antes de capturar a imagem. Na figura 25 é apresentado um exemplo de desalinhamento.

Figura 25 – Desalinhamento da cabeça

Conforme colocado a etapa do pré-exame ameniza o problema, porém não consegue resolvê-lo. Isso acontece, pois o desalinhamento da cabeça é verificado através da analise do alinhamento dos retângulos gerados pela fase de extração dos olhos (conforme mencionado na seção 3.3.2.1). Para ter um resultado mais eficiente, poderia se analisar o centro da região do brilho, porém isso não é possível, pois em pacientes estrábicos o centro do limbo pode não estar alinhado.

Para os 3 pacientes estrábicos foi obtido o resultado de 100% de precisão. Para considerar o resultado válido foi realizada a análise da imagem (manualmente) e verificado a correta localização das regiões do limbo e do brilho pela aplicação. Por fim, foi confrontado o resultado obtido com o diagnóstico médico da pessoa.

3.4.2 Consumo de memória

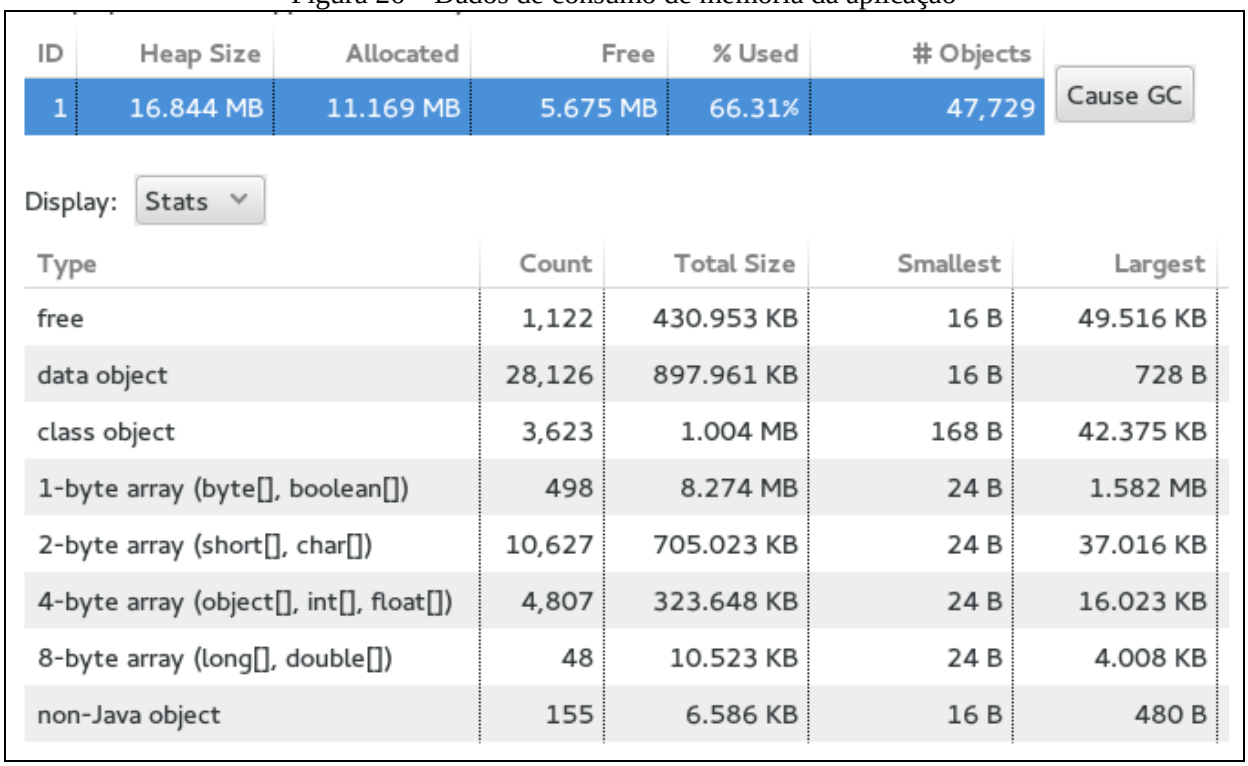
Nesta seção são apresentadas as métricas extraídas da aplicação de consumo de memória. Em relação ao consumo de memória da aplicação todas as métricas foram extraídas fazendo o uso da ferramenta Dalvik *Debug Monitor Server* (DDMS) que vem embarcada com a IDE Eclipse ADT.

Na plataforma Android cada aplicação executa em seu próprio processo e possui sua própria VM (DDMS, 2014). O DDMS permite que seja efetuado o monitoramento de todos os recursos utilizados pela aplicação.

No quesito consumo de memória foram efetuados testes em todas as funcionalidades do aplicativo: ao iniciar o aplicativo, na fase de pré-exame até a execução do exame.

O consumo de memória se manteve entre 11 e 14MB de memória alocada. A memória total alocada pela aplicação se manteve em torno de 16 a 18MB visto que a memória *heap* sempre mantinha 5 a 6MB de memória livre para aplicação. Na figura 26 é apresentado os dados de consumo de memória da aplicação.

Figura 26 – Dados de consumo de memória da aplicação



ID	Heap Size	Allocated	Free	% Used	# Objects
1	16.844 MB	11.169 MB	5.675 MB	66.31%	47,729

Cause GC

Display: Stats

Type	Count	Total Size	Smallest	Largest
free	1,122	430.953 KB	16 B	49.516 KB
data object	28,126	897.961 KB	16 B	728 B
class object	3,623	1.004 MB	168 B	42.375 KB
1-byte array (byte[], boolean[])	498	8.274 MB	24 B	1.582 MB
2-byte array (short[], char[])	10,627	705.023 KB	24 B	37.016 KB
4-byte array (object[], int[], float[])	4,807	323.648 KB	24 B	16.023 KB
8-byte array (long[], double[])	48	10.523 KB	24 B	4.008 KB
non-Java object	155	6.586 KB	16 B	480 B

Como pode ser visto na figura 26 o maior consumidor de memória são os *byte array* ocupando de 8 a 9MB. Estes objetos são objetos básicos como coleções e todos os objetos que não se encaixam nas outras categorias. Vale destacar os objetos *byte array*, pois estes compõem as imagens que permanecem carregadas durante a execução.

3.4.3 Desempenho

O primeiro teste realizado para medir o desempenho teve como objetivo validar a aplicação Android em seu uso. Neste teste foi avaliado o pré-exame onde a aplicação tem como objetivo auxiliar o usuário a realizar o exame.

Como neste processo a aplicação fica avaliando imagem por imagem capturada pela câmera e dando um *feedback* em cima de cada imagem recebida, este processo foi separado da *thread* principal, com isso obteve-se uma boa taxa de FPS. Nas medições realizadas a aplicação durante o pré-exame manteve uma taxa de 13 a 15 FPS. Com isso tendo uma fluência sem travamentos na visualização.

Outro teste realizado foi em relação à API de detecção de estrabismo em toda sua execução. Estes testes foram realizados no dispositivo Android Motorola Moto G (Processador 1.2 GHz Quad Core, 1GB RAM) e um notebook HP (Processador Intel Core 2 duo de 2.10GHz, 4GB RAM) com sistema operacional Arch Linux. Na tabela 2 são apresentados os resultados obtidos em cada hardware para cada etapa do processamento.

Tabela 2 – Comparação de desempenho entre hardwares

	Tempo (milissegundos)	
Fases	Smartphone	Notebook
Extração da região da face	25,6	15,2
Extração da região dos olhos	154	35,8
Extração da região do limbo	1516,2	159,8
Extração da região do brilho	143,6	16,2

Os resultados obtidos levaram em consideração testes específicos, nos quais o mesmo código e as mesmas amostras executaram sobre os dois hardwares. Cada fase foi executada 10 vezes com amostras distintas, com isso foi realizada a média das execuções.

Como pode ser observado na tabela 2 a execução teve tempos menores executando no *notebook*, conforme esperado em razão do maior poder de processamento. Porém pode-se observar que o ponto mais custoso para o algoritmo é a fase de extração do limbo.

Como no pré-exame, onde o desempenho é mais exigido tendo em vista o *feedback* contínuo ao usuário, a fase de extração de limbo e brilho não são utilizadas, o desempenho inferior nessas fases não chegam a afetar o uso da aplicação.

3.4.4 Comparativo entre o trabalho desenvolvido e os trabalhos correlatos

Nesta seção são apresentadas e discutidas as principais características da comparação entre os trabalhos correlatos, as quais estão ilustradas no quadro 20.

Quadro 20 – Quadro comparativo com os trabalhos correlatos

Características	Medeiros (2008)	Almeida (2010)	Krueger (2012)	Lorenzi (2014)
dispositivos móveis			X	X
computadores convencionais	X	X		X
utilização do OpenCV	X		X	X
implementação em	Java	C++	Objective-C	Java
reconhecimento de olhos escuros	X	X		X
reconhecimento de olhos claros	X	X	X	X
Auxílio na captura das imagens				X
Quantidade de amostras mais relevantes		X		

O quadro 20 ilustra as principais características do trabalho desenvolvido em relação aos trabalhos correlatos. Como diferencial possui uma fase de auxílio na captura da imagem, chamada no presente trabalho de pré-exame. Com isso é possível reduzir os erros na captura das imagens e também possibilitar que pessoas comuns consigam utilizar o aplicativo.

Outras características que se pode citar é o suporte a dispositivos móveis (com objetivo de efetuar a detecção do estrabismo) onde o poder de processamento e a capacidade de memória são limitados em relação aos computadores convencionais. Em paralelo ao suporte a dispositivos móveis também o suporte a execução em computadores tradicionais através da API de detecção do estrabismo.

Um ponto importante a ser citado é em relação às quantidades de amostras mais relevantes. O trabalho de Almeida (2008) possuiu uma quantidade maior de amostras de pessoas que possuíam o estrabismo. Com isso o analise dos resultados pode ser mais preciso em relação ao funcionamento da técnica.

Outro ponto em relação do trabalho de Almeida (2008) é que as imagens foram capturadas em um consultório (ambiente controlado) com o auxílio de um especialista, com isso tendo um melhor aproveitamento das amostras. No presente trabalho não foi possível encontrar auxílio de um especialista em estrabismo, com isso os resultados foram baseados nos diagnósticos que os pacientes possuíam do especialista. A técnica empregada neste trabalho usou como base o trabalho de Almeida (2008).

4 CONCLUSÕES

O presente trabalho apresentou a viabilidade do desenvolvimento de um software para detecção de estrabismo para a plataforma Android. Para tal, utilizou-se de técnicas de processamento de imagens e visão computacional, alcançando os objetivos propostos.

A técnica utilizada para a detecção do estrabismo foi o método de Hirshberg. Este método consiste em analisar o reflexo gerado por um ponto de luz em ambos os olhos e verificar o posicionamento deste foco em relação à região do limbo.

As principais técnicas de processamento de imagem utilizadas foram o classificador em cascasta Haar e a transformada de Hough. O classificador em cascata Haar foi utilizado com objetivo extrair a região da face e dos olhos. A transformada de Hough foi aplicada a círculos, com o objetivo de extrair a região do brilho e do limbo dos olhos.

As técnicas de processamento de imagem utilizadas se mostraram eficiente, visto que nas etapas de extração das regiões da face e da região dos olhos tiverem o resultado de 100% de acerto, sobre as amostras analisadas. Na etapa de extração das regiões do limbo e do brilho também houve um bom resultado, sendo que para extração da região do limbo obteve 95.45% e a extração da região do brilho obteve 96.36% de acerto em cima das amostras analisadas.

Em relação à detecção de estrabismo, os resultados apresentaram que a técnica proposta se mostrou eficiente, sendo que 85.71% deram o resultado conforme o esperado e 14.29% deram um resultado falso positivo. Isso mostra que o uso da técnica é viável visto que taxa de erro foi relativamente pequena, porém ainda existe a necessidade de possuir uma base de testes maior para que se tenha uma melhor avaliação do aplicativo.

Na fase de detecção do estrabismo o resultado obtido só não foi maior pela qualidade das amostras obtidas. Como o objetivo da aplicação é ser de fácil uso, sem a necessidade de um ambiente controlado e sem a necessidade de um especialista, as amostras obtidas levaram isso em consideração.

Para amenizar essa situação foi criada a etapa de pré-exame. Esta etapa tem como objetivo auxiliar o usuário na realização do exame, garantindo que a amostra capturada atenda algumas premissas básicas como, por exemplo, estar com os olhos alinhados.

A principal limitação da ferramenta é que a fase de pré-exame não consegue garantir 100% que a amostra capturada está correta para realização do exame. Com isso, são necessárias mais de uma amostra do mesmo usuário para conseguir mensurar a execução correta.

As ferramentas utilizadas mostraram-se eficientes, tendo destaque a biblioteca OpenCV. O uso da mesma simplificou a manipulação das imagens. Também destaca-se a IDE de desenvolvimento Eclipse ADT que possibilita a depuração e execução de aplicativos Android de forma simples e eficiente.

Por fim, pode-se destacar que o presente trabalho implementou técnicas para extrair regiões de interesse, como a região da face, dos olhos do limbo e do brilho. Todas estas extrações foram realizadas de forma independente, com isso cada etapa pode ser reaproveitada de forma independente.

4.1 EXTENSÕES

Sugerem-se as seguintes extensões para trabalhos futuros:

a) melhorar a etapa de pré-exame para que a mesma garanta que a amostra capturada seja válida. Com isso aumentando a viabilidade do uso da ferramenta por pessoas comuns;
b) estudar técnicas para definição de limiares de forma automática ao utilizar operador de Canny. Como se trata de uma etapa essencial para encontrar as regiões de interesse é importante que as bordas estejam bem definidas;
c) efetuar um trabalho de melhoria de desempenho em cima da transformada de Hough implementada neste trabalho. Para isso pode-se reimplementá-la em código nativo ou encontrar outra abordagem de implementação que atenda aos requisitos do problema;
d) aprimorar a quantidade e qualidade das amostras de teste. Principalmente aumentar a quantidade de pacientes estrábicos para ter uma análise dos resultados mais exata;
e) realizar a implementação de uma aplicação para dispositivos que utilizam a plataforma IOS;
f) realizar a implementação de outras técnicas com o objetivo de detectar o estrabismo. Uma técnica que poderia ser avaliada é o teste de oclusão (*cover test*);
g) efetuar testes em mais dispositivos da plataforma Android. Também com versões de plataforma diferentes;
h) implementar na aplicação uma forma para que o usuário do aplicativo possa contribuir com o banco de amostra. A idéia é o usuário baixar o aplicativo, realizar os exames e submeter as amostras para um servidor. Com isso, essas amostras poderiam ser utilizadas para aprimorar o aplicativo.

REFERÊNCIAS

ACTIVITY. **Activity**. [S.1.], 2014. Disponível em: <http://developer.android.com/reference/android/app/Activity.html>. Acesso em: 20 abr. 2014.

ALMEIDA, João D. S de. **Metodologia computacional para detecção automática de estrabismo em imagens digitais através do teste de Hirschberg**. 2010. 109 f. Dissertação (Mestrado em Engenharia de Eletricidade) - Curso de Pós-Graduação em Engenharia de Eletricidade, Universidade Federal do Maranhão, São Luís.

ANDROID SDK. **Get the Android SDK.** [S.l.], 2013. Disponível em: <http://developer.android.com/sdk/index.html>. Acesso em: 15 abr. 2013.

ANDROID DEVELOPERS. **Android, the world's most popular mobile platform**. [S.l.], 2013. Disponível em: <http://developer.android.com/about/index.html>. Acesso em: 14 abr. 2013.

ANDROID TOOLS. **Developer tools**. [S.l.], 2013. Disponível em: <http://developer.android.com/tools/index.html>. Acesso em: 14 abr. 2013.

ANDROID THREADS. **Processes and threads**. [S.1.], 2014. Disponível em: <http://developer.android.com/guide/components/processes-and-threads.html#Threads>. Acesso em: 15 abr 2014.

BICAS, Harley E. A. et al. **Estrabismo**. 2. ed. Rio de Janeiro: Cultura Médica, 2011.

BRADSKI, Gary; KAEHLER, Adrian. **Learning OpenCV**. Sebastopol: O' reilly Media, 2008.

BRUM, Graciela S. **Estrabismo**. [S.1.], 2014. Disponível em: <http://www.ceoportoalegre.com.br/estrabismo>. Acesso em: 15 abr. 2014.

CANNY, John. A computational approach to edge detection. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, Cambridge, v. 8, n. 6, p. 679-698, nov. 1986.

CASCADE CLASSIFICATION. **Haar feature-based cascade classifier for object detection**. [S.1.], 2014. Disponível em: <http://docs.opencv.org/modules/objdetect/doc/cascade_classification.html#cascade-classification>. Acesso em: 25 maio 2014.

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

DAUGMAN, John. High confidence visual recognition of persons by a test of statistical independence. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, [S.l.], v. 15, n. 11, p. 1148-1161, nov. 1993.

DDMS. **Using DDMS**. [S.1.], 2014. Disponível em: <http://developer.android.com/tools/debugging/ddms.html>. Acesso em: 20 abr. 2014.

DIAS, Carlos S. **O estrabismo**. [S.l.], 2013. Disponível em: <http://www.estrabismo.med.br/info.html#oque>. Acesso em: 16 abr. 2013.

DÓRIA, José L. **Detecção precoce das alterações da visão na criança**. [S.1.], 2006. Disponível em: <http://www.dgsaude.min-saude.pt/visao/OTec-Crianca2meses-10anos.pdf>. Acesso em 30 maio 2013.

DUDA, Richard O.; HART, Peter E.. Use of the hough transformation to detect lines and curves in pictures. In: COMMUNICATIONS OF THE ACM, 1., 1972, New York. **Anais...** New York: Rice University, 1972. v. 15, p. 11 - 15.

DUNCAN, James S.; AYACHE, Nicholas. Medical image analysis: progress over two decades and the challenges ahead. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 22, n. 1, p. 85-106, Jan. 2000.

FANG, Mei; YUE, Guangxue; YU, Qingcang. The study on an application of Otsu method in Canny operator. In: INTERNATIONAL SYMPOSIUM ON INFORMATION PROCESSING, 9., 2009, Huangshan. **Anais...** Huangshan: Jiaxing Universit, 2009. p. 109 - 112.

GONZALEZ, Rafael C.; WOODS, Richard E. **Digital image processing**. New Jersey: Pearson Prentice Hall, 2008.

IBGE. **Censo demográfico**. [S.l.], 2000. Disponível em: <http://www.ibge.gov.br/home/presidencia/noticias/08052002tabulacao.shtm>. Acesso em: 15 abr. 2013.

IMAGEJ. [S.l.], 2013. Disponível em: <http://rsb.info.nih.gov/ij>. Acesso em: 16 abr. 2013.

INTENT. **Intents and intent filters**. [S.1.], 2014. Disponível em: <http://developer.android.com/guide/components/intents-filters.html>. Acesso em: 18 abr. 2014.

JAMUNDÁ, Teobaldo. **Reconhecimento de formas**: a transformada de hough. [S.1], 2000. Disponível em: <http://www.inf.ufsc.br/~visao/2000/Hough>. Acesso em: 2 jun. 2013.

LAGANIÈRE, Robert. **OpenCV 2 computer vision application programming cookbook**. Birmingham: Packt Publishing Ltd, 2011.

KRUEGER, Matheus L. **Protótipo de software para autenticação biométrica baseada na estrutura da íris em dispositivos móveis**. 2012. 65 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

LIENHART, Rainer; KURANOV, Alexander; PISAREVSKY, Vadim. **Empirical analysis of detection cascades of boosted classifiers for rapid object detection**. Santa Clara: Intel Corporation, 2002. 7 p.

MARQUES FILHO, Ogê; VIEIRA NETO, Hugo. **Processamento digital de imagens**, Rio de Janeiro: Brasport, 1999.

MEDEIROS, Israel, D. **Ferramenta voltada à medicina preventiva para diagnosticar casos de estrabismo**. 2008. 67 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

MVC. **Model-View-Controller (MVC)**. [S.1.], 2014. Disponível em: <http://www.dsc.ufcg.edu.br/~jacques/cursos/map/html/arqu/mvc/mvc.htm>. Acesso em: 14 maio 2014.

OPENCV. [S.1], 2014. Disponível em: <http://opencv.org/about.html>. Acesso em: 10 mar. 2014.

OTSU, Nobuyuki. A threshold selection method from gray-Level histograms. **IEEE Transactions on Systems, Man and Cybernetics**, v. 9, n. 1, p. 62-66, jan. 1979.

PISTORI, Hemerson; COSTA, Eduardo Rocha. **Hough Circles**. [S.1.], 2014. Disponível em: <http://rsbweb.nih.gov/ij/plugins/hough-circles.html>. Acesso em: 22 abr. 2014.

RHODY, Harvey. **Hough circle transform**. [S.1.], 2005. Disponível em: <http://www.cis.rit.edu/class/simg782/lectures/lecture_10/lec782_05_10.pdf>. Acesso em: 23 maio 2014.

SIVA, Jailton; FERREIRA, Bruno F. de A; PINTO, Hugo S. R. **Princípios da oftalmologia**: avaliação oftalmológica. [S.l.]. 2013. Disponível em: <http://www.ligadeoftalmo.ufc.br/arquivos/ed_-_principios_-_avaliacao_oftalmologica.pdf>. Acesso em: 4 jun. 2013.

VALE, Giovane M.; POZ, Aluir P. D. O processo de detecção de bordas de canny: fundamentos, algoritmos e avaliação experimental. In: SIMPÓSIO BRASILEIRO DE GEOMÁTICA, 48., 2002, Presidente Prudente. **Anais...** Presidente Prudente: Universidade Estadual Paulista, 2002. p. 292-303.

VARELLA, Drauzio. **Estrabismo**. [S.1.], 2014. Disponível em: <http://drauziovarella.com.br/letras/e/estrabismo>. Acesso em: 25 abr. 2014.

VAUGHAN, Daniel; ASBURY, Taylor. **Oftalmologia geral**. 3. ed. São Paulo: Atheneu, 1990.

VIOLA, Paul; JONES, Michael. Rapid object detection using a boosted cascade of simple features. In: CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, 1., 2001, Cambridge. **Anais...** Cambridge: [s.1.], 2001. v. 1, p. 511 - 518.

WILDES, Richard P. Iris recognition: an emerging biometric technology. **Proceedings of the IEEE**, v. 85, n. 9, p. 1348-1363, Sept. 1997.