Renato Augusto Vieira Nishimori

Ferramenta de Acessibilidade para Deficientes Visuais em Cores

Orientador: Prof. Dr. Roberto Hirata Jr.
Universidade de São Paulo
22 de novembro de 2013

Resumo

A deficiência visual em cores, conhecida também como daltonismo, é uma deficiência que dificulta ou impossibilita a visão ou distinção de cores pelo indivíduo, e atinge uma quantidade significativa da população.

Embora existam ferramentas que visam aumentar a qualidade visual de imagens para portadores de daltonismo, sua aplicação muitas vezes é um processo demorado ou não leva em conta que essa deficiência acontece em diferentes graus.

O trabalho consiste no desenvolvimento de um aplicativo Android com três funcionalidades voltadas a deficientes visuais em cores: a primeira é uma ferramenta que possibilita simular como uma imagem selecionada é vista por pessoas com as principais deficiências visuais em cores; as outras ferramentas são voltadas para acessibilidade do próprio usuário com deficiência em cores: um filtro parametrizável que tem como objetivo facilitar ou possibilitar que informações não visíveis (devido à dificuldade de distinguir tons de vermelho e verde) se tornem visíveis e uma ferramenta de destaque de cor onde o usuário pode escolher uma cor a ser destacada na imagem, extraindo assim informações associadas.

A metodologia de desenvolvimento empregada foi o desenvolvimento dirigido por testes (comumente chamado de TDD), visando gerar um código bem testado, mais simples e menos suscetível a erros.

Palavras-chave: Daltonismo, Android, OpenCV, Testes, TDD

Lista de Figuras

1.1 Tabela de futebol (fonte: globoesporte.globo.com). 3

2.1 Anatomia de um bulbo ocular humano 7
2.2 Espectro visível 8
2.3 Estímulos dos diferentes cones 9
2.4 Exemplo de teste de cores de Ishihara 10

3.1 Arquitetura do Sistema Android 13
3.2 Ciclo de Vida de uma Atividade Android 14

5.1 Espaço de Cores RGB 21
5.2 Cubo de Cores RGB 21
5.3 Espaço de cores LMS 22

6.1 Filtro de acessibilidade com diferentes valores de i. 25

8.1 Integração com o Android e OpenCV 32

9.1 Tela principal 33
9.2 Tela de seleção de modo do filtro de acessibilidade 34
9.3 Tela do filtro de acessibilidade, modo contínuo 34
9.4 Tela de seleção de modos da ferramenta de simulação 35
9.5 Tela de confirmação da imagem obtida da câmera 35
9.6 Tela da ferramenta de simulação 35
9.7 Tela de seleção de modos da ferramenta de destaque de cor 36
9.8 Seleção de imagem a partir do dispositivo do usuário 36
9.9 Tela da ferramenta de destaque de cor 37
9.10 Tela do seletor de cor da ferramenta de destaque de cor 37
9.11 Tela da ferramenta de destaque de cor: resultado 37
9.12 Tela de configurações do usuário 38
9.13 Tela de seleção de cor padrão 38

10.1 Relatório de cobertura de testes 40
10.2 Resultados da simulação na imagem de vegetação. 40

10.3 Resultados da simulação na imagem de um semáforo. 41
10.4 Resultados do filtro de acessibilidade aplicado sobre uma imagem com contraste verde com vermelho problemático. 41
10.5 Resultados do filtro de acessibilidade aplicado sobre um cartão de Ishihara. 42
10.6 Resultados da ferramenta de destaque de cor em um gráfico. 43

11.1 Gráfico de usuários do aplicativo 45

Lista de Algoritmos

1 Algoritmo da ferramenta de simulação de deuteranopia e protanopia. . . 24
2 Algoritmo simplicado da ferramenta de simulação de deuteranopia e protanopia. 24
3 Algoritmo do filtro de acessibilidade 26
4 Algoritmo do filtro de acessibilidade com matriz de transformação 27
5 calculaDistancia(umaComponente, outraComponente, tolerancia) 29
6 verificaDistanciaAceitavel(umPixel, outroPixel, tolerancias) 29
7 converteEmEscalaDeCinza(p) 29
8 Algoritmo para destaque de cor 30

Sumário

I Parte Objetiva viii

1 Introdução 1
1.1 Trabalhos anteriores . . . 1
1.1.1 Ferramentas de simulação . . . 1
1.1.2 Ferramentas de acessibilidade . . . 2
1.1.3 Ferramenta Visocor . . . 2
1.2 Motivação . . . 2
1.3 Objetivos . . . 4
1.4 Estrutura do trabalho . . . 5

2 Percepção de Cores e Deficiências em Cores 6
2.1 Sistema visual humano . . . 6
2.1.1 Cones e bastonetes . . . 7
2.2 Percepção de cores . . . 7
2.2.1 Espectro visível . . . 7
2.2.2 Tipos de cones humanos e percepção de cores . . . 8
2.3 Deficiência visual em cores . . . 9
2.3.1 Tipos de deficiência . . . 9

3 Android e OpenCV 11
3.1 Sobre o Android . . . 11
3.2 Versões do Android . . . 12
3.2.1 Action Bar . . . 12
3.3 Arquitetura Android . . . 12
3.3.1 Máquina virtual Dalvik . . . 13
3.4 Ciclo de vida de uma atividade . . . 14
3.5 OpenCV . . . 15

4 Testes Automatizados e TDD 16
4.1 Testes de unidade . . . 16
4.2 Cobertura de testes . . . 17
4.3 Objetos Dublês . . . 17
4.4 Refatoração . . . 17

4.5 Desenvolvimento dirigido por testes (TDD) . . . 18
4.5.1 Vantagens do uso de TDD . . . 18

5 Ferramenta de simulação de Deficiência em Cores 20
5.1 Ferramenta de simulação de deficiência em cores . . . 20
5.1.1 Espaços de cores . . . 20
5.1.2 Detalhes da ferramenta de simulação . . . 22
5.2 Parâmetros da ferramenta . . . 23
5.3 Algoritmo do filtro . . . 24

6 Filtro de Acessibilidade 25
6.1 Filtro de acessibilidade . . . 25
6.2 Parâmetro do filtro . . . 26
6.3 Algoritmo do filtro . . . 26

7 Ferramenta de Destaque de Cor 28
7.1 Ferramenta de destaque de cor . . . 28
7.2 Parâmetros da ferramenta . . . 28
7.3 Cálculo da distância entre as cores . . . 29
7.4 Conversão para escala de cinza . . . 29
7.5 Algoritmo da ferramenta de destaque de cor . . . 30

8 Sistema e sua Arquitetura 31
8.1 Descrição dos pacotes do software . . . 31
8.2 Integração com o Android e OpenCV . . . 32

9 Manual do Usuário 33
9.1 Filtro de Acessibilidade . . . 33
9.2 Ferramenta de simulação de deficiência em cores . . . 34
9.3 Ferramenta de destaque de cor . . . 36
9.4 Configurações do usuário . . . 38

10 Resultados 39
10.1 Testes e cobertura . . . 39
10.2 Ferramenta de simulação de deficiência em cores . . . 40
10.3 Filtro de acessibilidade Visocor . . . 41
10.4 Ferramenta de destaque de cor . . . 42

11 Conclusões e Trabalhos Futuros 44
11.1 Conclusões . . . 44
11.2 Informações do aplicativo na Google Play . . . 44
11.3 Trabalhos futuros . . . 45

Referências Bibliográficas 46

II Parte Subjetiva **48**

12 Decisões, Desafios e Frustações **49**

13 Disciplinas Mais Relevantes e Importância do Intercâmbio **51**

14 Agradecimentos **54**

Parte I

Parte Objetiva

Capítulo 1

Introdução

A deficiência visual em cores, também conhecida como daltonismo, é a incapacidade de enxergar uma cor ou distinguir cores. Essa deficiência se apresenta em diferentes tipos e intensidades e atinge uma parte significativa da população: 8% da população masculina e 0,5% da população feminina[1, 2].

1.1 Trabalhos anteriores

O empenho em se produzir ferramentas de acessibilidade a deficientes em cores é crescente, sendo que as ferramentas mais comuns são as de simulação dessas deficiências. Esse tipo de ferramenta se dispõe a ajudar o designer ou desenvolvedor na escolha de cores, de modo a evitar que as informações apresentadas fiquem confusas ou ilegíveis para pessoas com deficiência em cores.

1.1.1 Ferramentas de simulação

São apresentados algumas dessas ferramentas de simulação e uma pequena descrição na lista abaixo:

Color Oracle Simula dicromatismo, agindo como um filtro que é aplicado na tela inteira do computador. Disponível para Linux, Windows e OS X.

ColorDoctor Simula dicromatismo e monocromatismo e permite ao usuário salvar resultados como imagens. Disponível apenas para Windows.

Sim Daltonism Simula dicromatismo e tricomatismo anômalo, disponível apenas para OS X.

Além disso, já existem plugins para importantes ferramentas de edição gráfica (como o Photoshop e o Gimp) e simuladores na rede, onde filtros são aplicados sobre imagens ou endereços web enviados.

1.1.2 Ferramentas de acessibilidade

Embora essas ferramentas de simulação sejam importantes para evitar problemas de acessibilidade, elas tem caráter apenas preventivo. No entanto, existem também ferramentas que tentam melhorar a qualidade visual de imagens para pessoas com deficiências em cores. Algumas dessas ferramentas e uma pequena descrição são apresentadas na lista abaixo:

Chrome Daltonize Plugin para o navegador Google Chrome que, além de simular deficiência em cores, implementa filtros de acessibilidade para dicromotas.

Sim Daltonism Além de simular dicromatismo, oferece uma ferramenta para isolar uma determinada cor em uma imagem. No entanto, essa escolha de cores é bem limitada, impossibilitando a escolha de uma cor com precisão. Disponível para Android.

Embora essas ferramentas de acessibilidade sejam sem dúvida úteis, elas supõem que o usuário já conheça sua deficiência. Mesmo com a existência de aplicativos que fazem o diagnóstico de qual o tipo de deficiência, não é levado em conta que esse distúrbio acontece em diferentes graus e não é oferecido ao usuário qualquer controle da intensidade do filtro aplicado, levando à imagens demasiadamente alteradas de forma desnecessária.

1.1.3 Ferramenta Visocor

O Visocor[3] é uma ferramenta de acessibilidade desenvolvida em 2009 no TCC dos alunos André Shoji Asato e Rafael de O. Lopes Gonçalves no qual esse trabalho é baseado.

Desenvolvida como um plugin código aberto para o Compiz, a ferramenta consiste de duas funcionalidades principais: um filtro parametrizável de acessibilidade - cuja finalidade é facilitar ou permitir a percepção contrastes verde com vermelho - e uma ferramenta de destaque de cor.

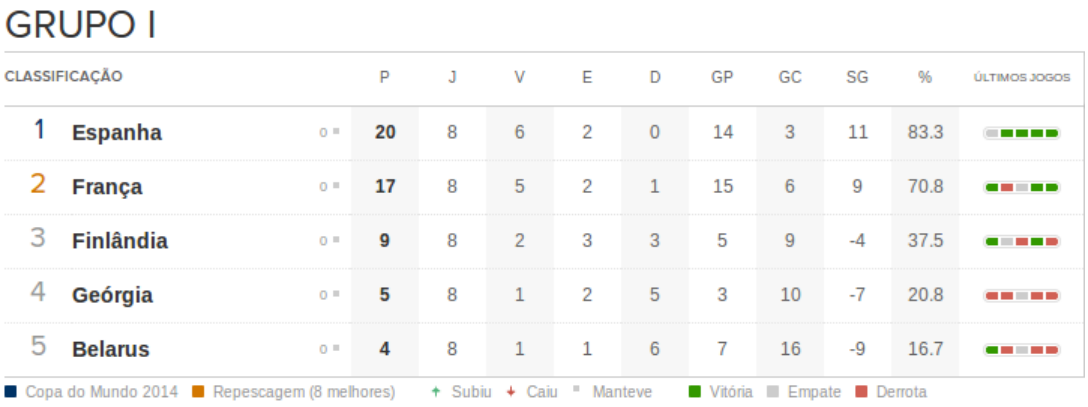
O Visocor foi escrito na linguagem de programação C e usa as interfaces de programação de aplicativos fornecidas pelo Compiz e OpenGL, permitindo seu uso em qualquer plataforma e sistema operacional que os suporte. A aplicação das ferramentas é feita em tempo real e sua configuração é feita em uma tela do Compiz, através de combinação de teclas e do uso do mouse.

1.2 Motivação

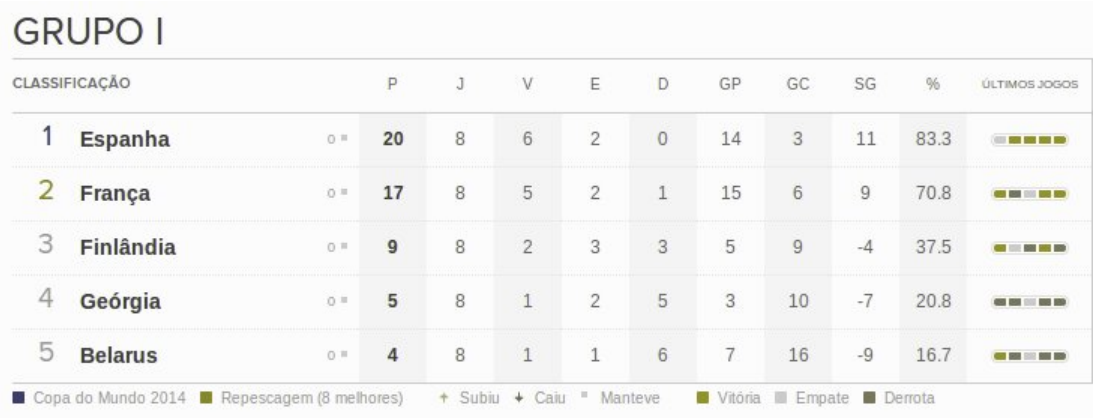
A motivação desse trabalho se dá principalmente ao interesse do responsável por esse trabalho por processamento de imagens, dispositivos móveis e ferramentas usando realidade aumentada. Embora não tenha nenhum tipo de deficiência em cores, o assunto foi escolhido após conversa com o orientador sobre quais projetos disponíveis poderiam se encaixar nas categorias citadas acima.

O trabalho original Visocor é muito interessante e útil, mas atualmente, na era dos dispositivos móveis, não tinha a visibilidade que merece pois seu uso é preso ao Compiz. Por esse motivo foi decidido estender esse trabalho para dispositivos móveis (com sistema operacional Android), adicionando algumas funcionalidades e melhorando a interface.

Outra motivação foi presenciar as dificuldades de alguns amigos que possuem deficiência visual em cores em realizar atividades como navegar por sites (como o da figura 1.1), jogar jogos eletrônicos e jogos de tabuleiro sem acessibilidade a daltônicos, além de atividades diárias onde essa deficiência se apresenta como um problema.



(a) Imagem original



(b) Simulação de deficiência em cores

Figura 1.1: Tabela de futebol (fonte: globoesporte.globo.com).

1.3 Objetivos

O software desenvolvido nesse trabalho tem como objetivo implementar três funcionalidades de acessibilidade relacionadas à deficiência visual em cores:

Ferramenta de simulação de deficiência em cores

A ferramenta de simulação permite ao usuário simular como a imagem seria vista por pessoas portadoras dos principais tipos de deficiência visual em cores.

Embora não seja útil para os deficientes em cores, a ferramenta pode ser utilizada por desenvolvedores e designers para garantir que seu conteúdo é acessível para deficientes visuais em cores. Assim, ao invés de ter que fazer o demorado processo de aplicação de filtros em uma ferramenta de edição gráfica, o usuário pode rapidamente usar a câmera de seu dispositivo móvel para simular a visão deficiente em cores.

Filtro parametrizável de acessibilidade

O filtro possibilita que os deficientes visuais em cores consigam extrair informações de imagens onde isso não seria possível devido à dificuldade de distinguir o contraste vermelho com verde. Ao implementar esse filtro em um dispositivo móvel garante-se que ele não seja útil apenas para imagens digitais, mas também para atividades do dia à dia (como por exemplo na leitura de rotas de mapas de metrô e ônibus nas ruas).

O filtro suporta os principais tipos de deficiência visual em cores e sua grande vantagem quando comparado ao de outros trabalhos é ser parametrizável, permitindo assim que o usuário possa personalizá-lo de acordo com seu grau de deficiência e modificando a imagem o mínimo possível para que outras informações não sejam afetadas.

Ferramenta de destaque de cor

A ferramenta de destaque de cor permite que o usuário selecione uma cor que será destacada em toda a imagem. Essa ferramenta é particularmente útil nos casos em que o problema de extrair a informação na imagem não é causado pela dificuldade do deficiente em cores em distinguir contrastes, mas sim em localizar em que partes da imagem uma dada cor aparece. Isso acontece por exemplo em imagens com legendas em cores, como mapas e gráficos.

Além dessas ferramentas, foi dada uma grande ênfase à metodologia utilizada para o desenvolvimento desse software, denominada desenvolvimento dirigido por testes (ou simplesmente TDD - Test Driven Development). Essa metodologia foi escolhida para melhorar a qualidade do código escrito, garantir uma grande quantidade de testes automatizados e com isso diminuir a suscetibilidade a erros.

1.4 Estrutura do trabalho

No capítulo 2 são mostrados conceitos básicos de visão em cores e os motivos por trás dos diferentes tipos de deficiência em cores.

No capítulo 3 é feita uma revisão das tecnologias utilizadas no projeto, Android e OpenCV.

No capítulo 4 descreve-se a metodologia utilizada, o desenvolvimento dirigido por testes. Também são descritos conceitos relacionados a essa metodologia e as vantagens de seu uso.

No capítulo 5, 6 e 7, as três ferramentas são descritas em maiores detalhes, conceitos por trás de cada uma explicados e algoritmos em pseudo-código são mostrados.

No capítulo 8 informações mais detalhadas da arquitetura do sistema e sua integração com o Android e OpenCV são apresentadas.

No capítulo 9 são apresentados os estudos de caso das três ferramentas na forma de um manual do usuário.

No capítulo 10 são mostrados resultados das três ferramentas e o relatório de testes do software.

Por fim, no capítulo 11 são mostradas as conclusões assim como possibilidades de trabalhos futuros relacionadas ao projeto.

Capítulo 2

Percepção de Cores e Deficiências em Cores

2.1 Sistema visual humano

O sistema visual humano possilita o ser humano ver e é extremamente complexo e desenvolvido. Ao contrário de muitos animais, onde esse sistema somente ajuda na sobrevivência da espécie, no homem é um dos mais eficiente e complexos sistemas de percepção, pois diversos elementos no ambiente só podem ser captados, interpretados e entendidos através da visão[4].

O sistema visual humano é composto, de forma resumida, dos bulbos dos olhos (geralmente chamados simplesmente de olhos), os nervos ópticos e o cérebro. Os olhos são órgãos da visão que detectam e transformam a luz visível em um sinal neural. Após essa transformação, os nervos ópticos são responsáveis por levar o sinal neural até o cérebro, onde esse sinal será interpretado e as informações serão obtidas[5].

No cérebro, os elementos relacionados à visão são o lóbulo occipital (encarregado da interação do sinal com outras informações), parietal e temporal (encarregado da interpretação do sinal). Para esse projeto, a etapa mais importante do processo de visão é a etapa inicial que acontece nos olhos, já que é nessa fase que se manifestam as formas de deficiência em cores mais comuns[6].

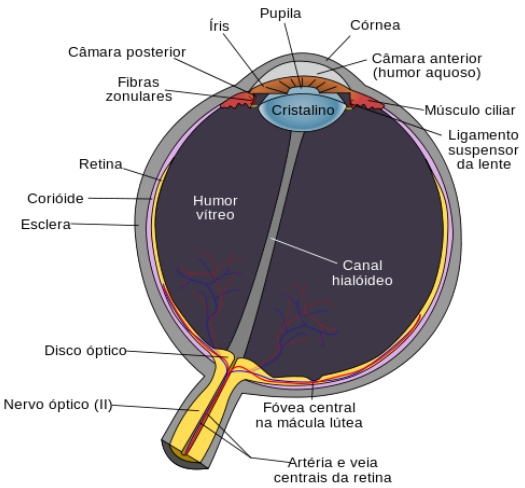


Figura 2.1: Anatomia de um bulbo ocular humano (fonte [7]).

O bulbo ocular fica localizado dentro de uma cavidade óssea no crânio e possui o formato aproximado de uma esfera. A figura 2.1 mostra os principais componentes do bulbo ocular. Entre essas componentes, há um interesse específico na retina, localizada no fundo da superfície interna do olho. É nela que estão localizadas os fotorreceptores, células responsáveis pela detecção dos estímulos luminosos, que podem ser divididos em cones e bastonetes.

2.1.1 Cones e bastonetes

Bastonetes estão mais concentrados na parte periférica da retina e possuem alta sensibilidade acromática à luz, o que significa que possuem maior sensibilidade à luz mas não conseguem distinguir cores. Bastonetes são mais importantes em ambientes com pouca luminosidade, pois devido à alta sensibilidade à luz possibilitam a visão do ambiente. Existem em torno de 120 milhões de bastonetes no olho humano.

Cones são possuem características fotópicas, o que significa que possuem menor sensibilidade à luz porém possibilitam a visão em cores, e são mais abundantes na parte central, denominada fóvea. A quantidade de cones no olho humano é bem menor que a de bastonetes, apenas 6 milhões[5].

2.2 Percepção de cores

2.2.1 Espectro visível

O espectro eletromagnético é o intervalo de todas as possíveis frequências de radiação eletromagnética. Esse intervalo é dividido em diversas bandas sendo que uma delas,

chamada de espectro visível, luz visível ou simplesmente luz pode ser detectada pelo olho humano.

Essa estreita banda visível pelo olho humano situa-se entre 390 até 700 nm, embora esse valor possa variar de pessoa para pessoa. No entanto, nem todas as cores que os olhos e cérebro podem distinguir estão contidas nesse espectro, já que algumas cores como por exemplo o rosa e roxo são compostas de misturas de diferentes comprimentos de onda.

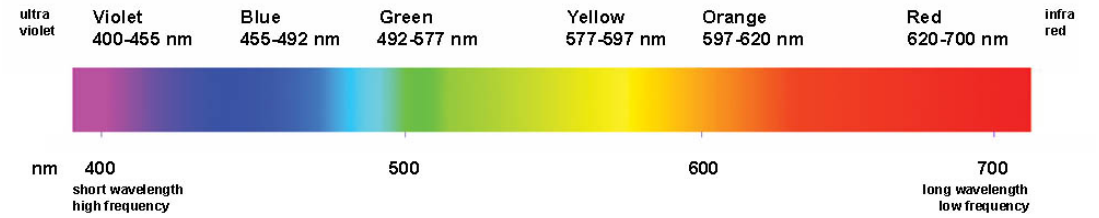


Figura 2.2: Espectro de luz visível(fonte [7]).

2.2.2 Tipos de cones humanos e percepção de cores

No olho humano, existem três diferentes tipos de cones, que podem ser classificados de acordo com o comprimento de onda pelo qual eles são estimulados. Cones estimulados por comprimentos de onda longo (por volta de 670 nm) são sensíveis à cor vermelha, enquanto os estimulados por comprimentos de onda médio (por volta de 540 nm) são sensíveis ao verde e finalmente os estimulados por comprimentos de onda curtos (por volta de 440 nm) são sensíveis ao azul (ver figura 5.3).

As diferenças nos estímulos dos diferentes cones se deve ao tipo de pigmento foto-sensível presente no mesmo, enquanto a percepção de uma cor pelo cérebro depende dos graus de excitação dos diferentes cones ao serem expostos à luz[5].

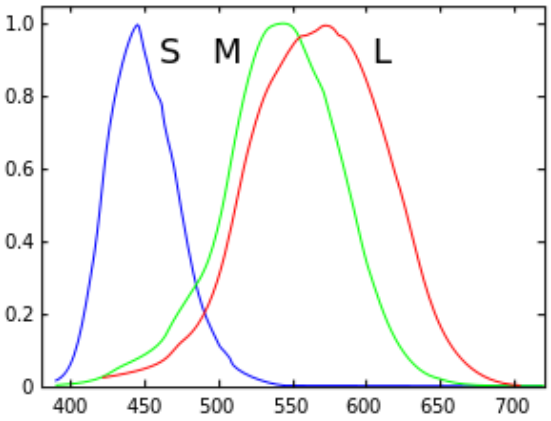


Figura 2.3: Intensidade dos estímulos dos três diferentes cones ao serem expostos a diferentes comprimentos de onda(fonte [7]).

2.3 Deficiência visual em cores

A deficiência visual em cores, também conhecida como daltonismo, se manifesta como a incapacidade ou dificuldade de enxergar uma cor ou distinguir cores que pessoas sem esse distúrbio não teriam problemas para fazê-lo. Essa deficiência atinge uma porcentagem significativa da população, sendo afetados 8% da população masculina e 0,5% da população feminina[1, 2].

Ao longo da história, essa deficiência foi estudada por muitos intelectuais e cientistas, sendo o inglês John Dalton o primeiro a publicar um artigo científico sobre o assunto, em 1798. Dalton fez uma auto-análise de sua deficiência de diferenciar cores e por isso o termo daltonismo surgiu, em sua homenagem.

Como a deficiência se apresenta em diferentes intensidades, muitas vezes o indivíduo pode levar anos até perceber que possui sua visão em cores afetadas. Porém, hoje em dia, devido principalmente à popularização da Internet, de dispositivos móveis e a interação com interfaces com diversos tipos de esquemas de cores, essas deficiências se tornaram muito mais distinguíveis e passaram a incomodar muito mais.

2.3.1 Tipos de deficiência

A deficiência visual em cores pode ser de origem congênita ou adquirida. Geralmente é hereditária, já que os genes que produzem fotopigmentos são ligados ao cromossomo X, e é uma característica recessiva. Por esse motivo, anomalias nesses genes ocasionam a deficiência visual em cores e afetam muito mais indivíduos do sexo masculino do que do sexo feminino, já que homens apresentam apenas um cromossomo X enquanto as mulheres apresentam dois. Outro motivo que ocasiona a deficiência em cores são danos físicos ou químicos aos olhos, nervo óptico ou partes do cérebro[6].

Indivíduos sem deficiências em cores são chamados tricromatas, pois possuem os três cones sem anomalia, que podem assim ser estimulados normalmente. Já indivíduos

que apresentam algum tipo de deficiência, podem ser classificados em três categorias[6]:

Monocromatismo Fenômeno muito raro, também chamado de visão acromática. O indivíduo não é capaz de enxergar nenhuma cor devido a anomalias nos três tipos de cone, vendo o mundo apenas em tons de cinza.

Dicromatismo Fenômeno mais comum entre os três citados, causado pela ausência ou diminuição de um dos três tipos de cone. O dicromatismo se divide em deuteranopia (cones sensíveis à cor verde), protanopia (cones sensíveis à cor vermelha) e tritanopia (cones sensíveis à cor azul). Entre esses três tipos, a deuteranopia e protanopia são as mais comuns.

Tricromatismo anômalo Devido a uma mutação dos pigmentos nos fotoreceptores, as proporções entre as cores vermelha, verde e azul são percebidas de maneira alterada. O tricromatismo anômalo pode ser dividido em protanomalia (menor sensibilidade ao vermelho), deuteranomalia (menor sensibilidade ao verde) e tritanomalia (menor sensibilidade ao azul). Entre esses três tipos, a protanomalia e deuteranomalia são as mais comuns.

Um dos métodos mais conhecidos de diagnóstico da deficiência é através do teste de cores de Ishihara, onde são apresentados cartões contendo uma imagem com um esquema de cores tal que somente pessoas com visão normal conseguem extrair uma determinada informação daquela imagem (geralmente um número). Um exemplo de cartão de teste de cores de Ishihara é mostrado na figura 2.4.

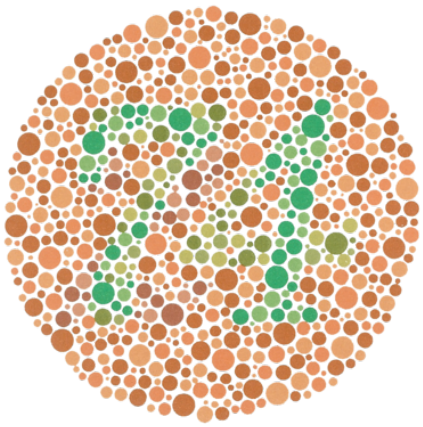


Figura 2.4: Exemplo de teste de cores de Ishirara. Pessoas com visão normal enxergam o número 74, monocromatas não enxergam nenhum número e dicromatas ou tricromatas anômalos podem enxergar o número 21.

Capítulo 3

Android e OpenCV

3.1 Sobre o Android

Android é um sistema operacional móvel baseado em uma versão modificada do Linux, desenvolvido inicialmente por uma startup com nome *Android, Inc* que foi comprada pela Google em 2005. Como a Google queria que esse sistema fosse aberto e livre, a maior parte do código está sobre licença código aberto Apache, o que possibilita que qualquer pessoa faça download do código fonte completo, modifique-o e distribua-o[8].

No fim de 2008 foi vendido o primeiro celular usando Android como sistema operacional. Em março de 2013, 64% do mercado de smartphones já eram modelos usando Android, sendo aproximadamente 12 mil o número de diferentes modelos de aparelhos (não apenas smartphones, mas também tablets). Em maio de 2013, quase 1 bilhão de aparelhos Android já tinham sido ativados e quase 50 bilhões de aplicativos tinham sido instalados da Google Play Store (a loja de aplicativos da Google)[9].

O grande sucesso do Android se deve em boa parte à decisão em relação ao código fonte, já que após o lançamento do iPhone, grandes empresas da indústria de smartphones tiveram que repensar seus produtos para se adequar a esse novo fenômeno. O Android então foi escolhido como o sistema operacional da maioria dessas empresas, já que além de gratuito possibilitava a adição de funções específicas e personalização do sistema para essas diferentes empresas. No entanto isso também traz alguns problemas já que a atualização do sistema deve também ser feita por essas empresas, algo que muitas vezes demora muito tempo para ser feito ou nem ao menos é feito.

Para o desenvolvedor de aplicativos móveis, a grande vantagem é o número de diferentes aparelhos que podem executar o aplicativo desenvolvido, sem ter que criar diversas versões para diferentes sistemas operacionais. Além disso, o fato de uma versão modificada do Java ser utilizada como linguagem de programação para Android deixa o aprendizado mais simples.

3.2 Versões do Android

O Android evoluiu muito desde seu lançamento inicial, conforme demonstrado na tabela abaixo:

Tabela 3.1: Versões do Android e datas de lançamento.

Versão	Data de Lançamento	Nome da Versão
1.1	Fevereiro de 2009	-
1.5	Abril de 2009	Cupcake
1.6	Setembro de 2009	Donut
2.0/2,1	Outubro de 2009	Eclair
2.2	Maio de 2010	Froyo
2.3	Dezembro de 2010	Gingerbread
3.0-3.2	Fevereiro de 2011	Honeycomb
4.0	Outubro de 2011	Ice Cream Sandwich
4.1-4.3	Julho de 2012	Jelly Bean

3.2.1 Action Bar

Uma das mudanças mais importantes na versão HoneyComb foi a introdução da componente chamada Action Bar - que funciona como uma barra de título, de menu e modos de navegação - e que foi adotada pela grande maioria dos aplicativos[10].

Nesse trabalho, para suportar versões antigas do Android que ainda tem uma participação significativa no mercado (Gingerbread) e ao mesmo tempo adotar essa componente amplamente utilizada por aplicativos Android, a solução encontrada foi o uso da biblioteca ActionBarSherlock[11]. Essa biblioteca implementa a Action Bar em versões de Android que não a suportam e, caso a versão utilizada tenha suporte nativo, usa a implementação nativa.

3.3 Arquitetura Android

A figura 3.1 mostra as diversas camadas que formam o sistema Android, podendo ser dividida em 5 partes principais, em 4 camadas[9]:

Aplicativos Todos os aplicativos, tanto os nativos quanto os desenvolvidos por terceiros, pertencem a essa camada e se comunicam com a camada inferior através de bibliotecas de API (Interface de Programação de Aplicativos).

Arcabouço de Aplicativos Responsável pelas classes utilizadas pelos aplicativos e também pela abstração para acesso ao hardware, além de recursos para a interface do usuário e seus aplicativos.

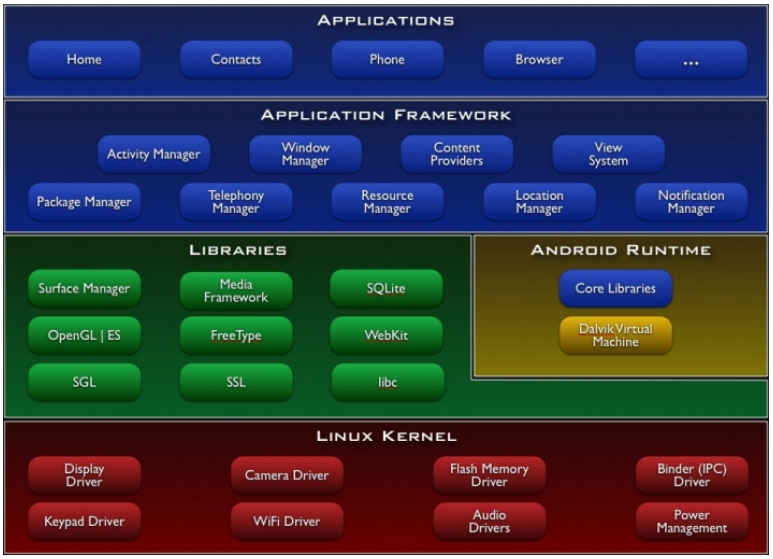


Figura 3.1: Arquitetura do sistema Android, separada em 4 camadas: Aplicativos, Arcabouço de Aplicativos, Bibliotecas, Android Runtime e Núcleo Linux (fonte: [12]).

Bibliotecas São executadas acima do núcleo Linux e incluem bibliotecas principais em linguagem C/C++ e também outras como: bibliotecas gráficas, de controle de mídia, de acesso a banco de dados, entre outras.

Android Runtime É o motor do dispositivo Android e o que o torna um dispositivo Android ao invés de uma implementação diferente do Linux. Formam a base do arcabouço de aplicativos e incluem as bibliotecas núcleo (que proporcionam a maioria das funcionalidades incluídas em bibliotecas Java, assim como algumas exclusivas do Android) e a máquina virtual Dalvik (detalhes na seção 3.3.1).

Núcleo Linux Composto por drivers de hardware, gerenciamento de processos, memória e energia, rede e segurança. Além disso, proporciona uma abstração entre o hardware e as bibliotecas.

3.3.1 Máquina virtual Dalvik

A máquina virtual Dalvik não é simplesmente uma máquina virtual Java, mas sim uma máquina virtual personalizada para o Android de modo a garantir uma boa eficiência. Os aplicativos são geralmente escritos em Java, compilados para bytecode para máquina virtual Java e então convertidos para arquivos .dex (executáveis Dalvik, que minimizam o uso de memória e processamento necessário). O Android também permite o desenvolvimento em C/C++, mas novamente o executável final gerado é um arquivo compatível com a máquina virtual Dalvik[9].

A Dalvik usa chamadas ao núcleo Linux para funcionalidades baixo nível, sendo responsável como intermediária a todo acesso ao hardware e sistema.

3.4 Ciclo de vida de uma atividade

Uma atividade de um projeto Android é uma tela de interface na aplicação. Todos os aplicativos são formados por uma ou mais atividades, sendo uma delas a atividade inicial que é mostrada ao abrir o aplicativo. Sendo assim, é extremamente necessário entender exatamente o comportamento de uma atividade do momento que ela é criada até o momento em que é destruída.

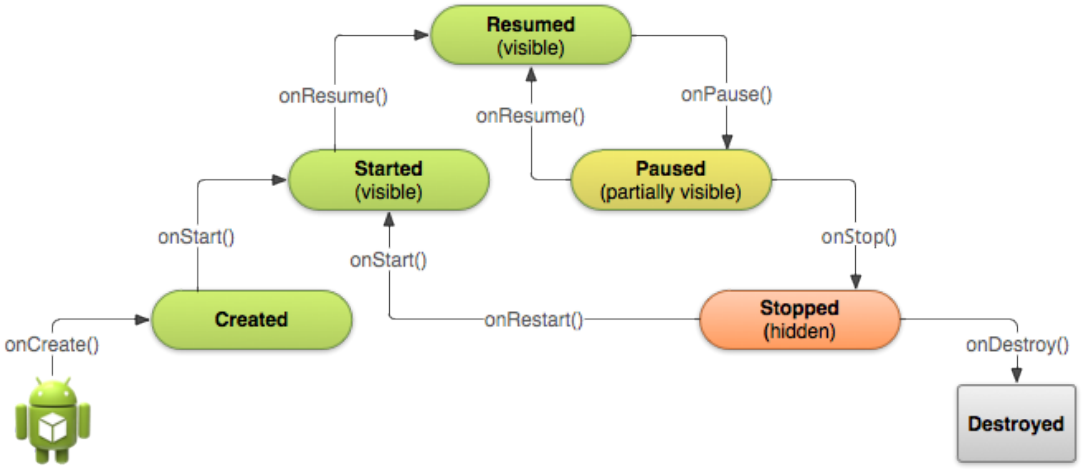


Figura 3.2: Ciclo de vida de uma atividade Android (fonte: [12]).

Os diferentes eventos que governam os estágios de uma atividade o ciclo de vida de uma atividade podem ser vistos no diagrama 3.2 e são descritos a seguir:

onCreate() Chamado quando a atividade é criada pela primeira vez.

onStart() Chamado assim que a atividade fica visível para o usuário.

onResume() Chamado assim que a atividade está pronta para interação com o usuário.

onPause() Chamado quando uma nova atividade é invocada ou uma atividade anterior é retomada.

onStop() Chamado quando a atividade não está mais visível para o usuário.

onDestroy() Chamado antes que o sistema destrua a atividade, seja para liberar recursos ou a pedido do usuário.

onRestart() Chamado quando a atividade que estava parada volta a ficar visível.

Além disso, mudanças de configuração do dispositivo, como por exemplo manusear o celular trocando do modo retrato para o modo paisagem fazem com que o evento onDestroy seja chamado, seguido do onCreate.

3.5 OpenCV

A biblioteca OpenCV[13] (Open Source Computer Vision Library) é uma biblioteca livre para uso e de código aberto para visão computacional e processamento de imagem em tempo real. Escrita em C++, mas com interfaces também em diferentes linguagens como Java e Python, a OpenCV é também multiplataforma, podendo ser executada em Windows, Linux, OS X, Android, iOS, entre outros sistemas operacionais.

Embora existam diversos algoritmos de Visão Computacional e Aprendizagem Computacional nessa biblioteca, o uso de OpenCV no projeto foi devido somente a simplicidade e eficiência para aplicação de filtros em tempo real sobre a imagem obtida da câmera.

Capítulo 4

Testes Automatizados e TDD

4.1 Testes de unidade

Um teste de unidade é uma maneira de se testar, de forma independente e automatizada, pequenos fragmentos de código (geralmente um método) de forma a prevenir e corrigir bugs antes que o código entre em produção, garantindo que o código se comporta da maneira que o desenvolvedor esperava[14]. Testes de unidade são uma parte essencial de qualquer software a ser desenvolvido, embora muitas vezes sejam feitos de forma errada ou completamente ignorados.

Os benefícios de se escrever testes de unidades são muitos: erros são encontrados mais rapidamente, a atenção ao design de classe é maior, facilitando assim mudanças posteriores e integração, e serve de documentação pois o código do teste mostra como o desenvolvedor planejou que a unidade testada fosse utilizada[14].

Um erro ainda muito comum é o desenvolvimento de testes somente ao final do projeto. Embora alguns dos benefícios citados ainda sejam válidos, o ideal é o desenvolvimento de testes juntamente com código de produção, ou até desenvolver testes antes do código (prática denominada TDD, mais detalhes na sessão 4.5). Outro problema de deixar os testes somente para o final é que os mesmos acabam muitas vezes diminuídos ou eliminados devido a prazos de entrega.

Embora escrever testes de unidade gaste um tempo que poderia estar sendo utilizado para implementar novas funcionalidades, escrever testes não deve ser visto como perda de produtividade pois, ao escrever uma bateria de testes abrangente e clara, o tempo gasto para escreve-la será recuperado rapidamente pela facilidade em achar e corrigir bugs e pela diminuição do tempo de depuração associada.

Para o desenvolvimento de testes no projeto foi utilizado o JUnit[15]. O JUnit é um arcabouço de testes de unidade para Java amplamente utilizado por desenvolvedores. Outra ferramenta utilizada foi o Robolectric[16], um arcabouço de testes de unidade especialmente desenvolvido para Android que usa o JUnit em seu auxílio. O Robolectric permite que aplicativos desenvolvidos para Android sejam rapidamente testados, ao contrário do arcabouço nativo de Android que necessita de um emulador de dispositivo móvel e por isso é mais demorado.

4.2 Cobertura de testes

Cobertura de testes é uma métrica utilizada para descrever o quanto do código fonte é testado por uma determinada bateria de testes[17]. De modo geral, uma maior cobertura de testes indica que o software desenvolvido foi mais testado do que um software com baixa cobertura. Embora seja desejável alcançar 100% de cobertura de testes, a ausência disso não deve ser algo alarmante pois pode ser muito difícil ou impossível devido a diversos fatores. Já a ausência total ou baixa porcentagem coberta é algo gravíssimo no desenvolvimento de software, como já mencionado anteriormente.

Existem diversos critérios para o cálculo de cobertura de testes. Alguns consideram o código coberto se o método é chamado ao menos uma vez, outros se todas condições internas (o verdadeiro ou falso de um if, por exemplo) foram avaliados.

Para o projeto, foi utilizada uma ferramenta para Eclipse chamada EclEmma[18], de uso livre. Essa ferramenta permite rodar os testes e gera um relatório mostrando, por classe, a porcentagem de código coberto e quais partes do código estão cobertas, parcialmente cobertas e descobertas. O critério utilizado para cálculo de cobertura nesse trabalho foi o número de instruções que são chamadas pelos testes automatizados.

4.3 Objetos Dublês

Outra idéia importante relacionada a testes de unidade é o uso de objetos dublês, mais conhecidos como objetos mock. Um objeto dublê é um objeto que simula o comportamento de objetos normais de uma classe, porém de maneira controlada[19].

Muitas vezes a criação de um objeto para testes pode ser trabalhosa devido à presença de objetos dos quais o objeto a ser testado é dependente (embora um número realmente alto desses objetos sugira um design de classes ruim). O uso de objetos dublês permite ao desenvolvedor criar esses objetos mais rapidamente e testar suas funcionalidade de forma mais independente.

Objetos dublês são muito utilizados para testes que envolvem banco de dados, comportamento não-determinístico, estados que são difíceis de serem testados (exemplo: falha em conexões de rede ou disco), entre outros.

Para o projeto, utilizou-se o amplamente reconhecido arcabouço de objetos dublês chamado Mockito[20] e também sua extensão PowerMockito[21]. Esses arcabouços possibilitam, de maneira fácil e rápida, a criação de objetos dublês e a reescrita de métodos do objeto real de forma a obter-se um comportamento controlado.

4.4 Refatoração

Outro processo extremamente importante no desenvolvimento de software é a refatoração. A refatoração consiste em modificar o código já existente de modo melhorar a estrutura interna do código, porém sem alterar seu comportamento externo (ou seja, mudar **como** o código faz, mas não **o que** o código faz)[22].

O uso dessa técnica tende a melhorar o design do código, pois a cada refatoração idealmente teremos um código mais limpo e claro. Observe que é fundamental a presença de testes de unidade que cubram o código sendo refatorado, garantindo assim que o código continua sendo capaz de realizar as funcionalidades iniciais e que nenhum novo bug foi introduzido durante a refatoração.

4.5 Desenvolvimento dirigido por testes (TDD)

Desenvolvimento dirigido por testes (Test-Driven Development, TDD) é uma prática onde primeiramente se escrevem os testes para o código para somente depois escrever o código em si. Essa prática ganhou popularidade em 2002 após a publicação do livro TDD By Example, de Kent Beck[17]. Para utilizar essa prática o desenvolvedor deve seguir uma sequência de desenvolvimento a cada funcionalidade, conforme descrito abaixo:

1. Escrever o teste de unidade para uma nova funcionalidade;
2. Executar o teste (aqui o teste falha pois ainda não há código para fazê-lo passar);
3. Implementar o código mais simples de modo a fazer o teste passar (conhecido como baby steps);
4. Executar o teste novamente, que dessa vez irá passar;
5. Refatorar o código (tanto da funcionalidade quanto dos testes), se possível.

4.5.1 Vantagens do uso de TDD

Praticantes de TDD afirmam que a prática é muito vantajosa para o desenvolvimento de software. Algumas das vantagens (sendo que algumas intensificam as vantagens citadas anteriormente sobre o uso de testes de unidade) citadas são[19]:

Simplicidade Ao escrever o código mais simples para passar o teste e somente posteriormente buscar uma refatoração, o desenvolvedor evita soluções complexas e desnecessárias.

Feedback mais rápido Ao escrever testes antes do próprio código, ao invés de somente após ter escrito o código, o desenvolvedor recebe um feedback mais rápido, podendo encontrar bugs e design mal feito enquanto ainda é cedo e simples de corrigi-lo.

Código desenvolvido já começa testado Como para toda funcionalidade a ser desenvolvida os testes devem ser escritos antes, o código para aquela funcionalidade já começa testado.

Melhoria no design Idéias importantes de programação orientada à objetos como alta coesão e baixo acoplamento tendem a ser seguidas mais à risca, pois caso contrário o teste se torna muito difícil de ser escrito e objetos das classes a serem testadas muito difíceis de serem instanciadas.

Refatoração constante Outra vantagem é a constante refatoração do código, deixando-o sempre mais claro e bem escrito. A presença de testes de unidade proporcionam também uma maior segurança para o desenvolvedor efetuar a refatoração.

Capítulo 5

Ferramenta de simulação de Deficiência em Cores

A partir desse capítulo serão explicadas mais profundamente as ferramentas e o sistema desenvolvido nesse trabalho.

5.1 Ferramenta de simulação de deficiência em cores

A ferramenta de simulação de deficiência em cores tem como objetivo possibilitar um indivíduo com visão normal a enxergar imagens como um indivíduo com deficiências em cores (com protanopia ou deuteranopia) enxergaria e é baseada no trabalho *Digital Video Colourmaps for Checking the Legibility of Displays by Dichromats* de Viénot, Brettel e Mollon.[1]

Essa funcionalidade é especialmente útil para ajudar, por exemplo, designers e desenvolvedores web a criarem conteúdo com um esquema de cores acessível para deficientes em cores.

Para uma descrição mais detalhada do algoritmo da ferramenta primeiramente é necessário explicar o conceito de espaços de cores, já que o algoritmo envolve uma conversão do tradicional espaço RGB para o espaço LMS, que simula a visão humana utilizando respostas dos cones como suas coordenadas.

5.1.1 Espaços de cores

Espaços de cores (também chamados de sistema de cores, modelos de cores ou espectro de cores) são modelos matemáticos abstratos criados que tentam formalizar a descrição de cores através de tuplas (geralmente triplas ou quadruplas) de números[23].

Espaço de Cores RGB

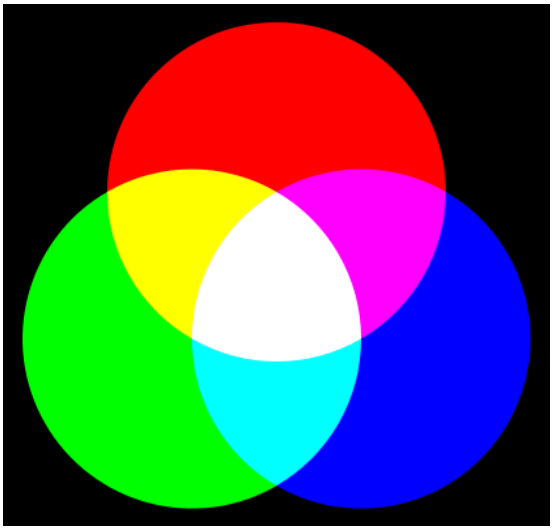


Figura 5.1: Espaço de Cores RGB (fonte [7]).

O RGB (Vermelho, Verde, Azul) é o espaço de cores mais conhecido pois a reprodução de cores em dispositivos eletrônicos é baseado nele. Esse modelo se baseia em coordenadas cartesianas e o espaço formado é um cubo, onde as cores primárias do sistema mais as cores branca, preta, ciano, magenta e amarela são os vértices (ver figura 5.2).

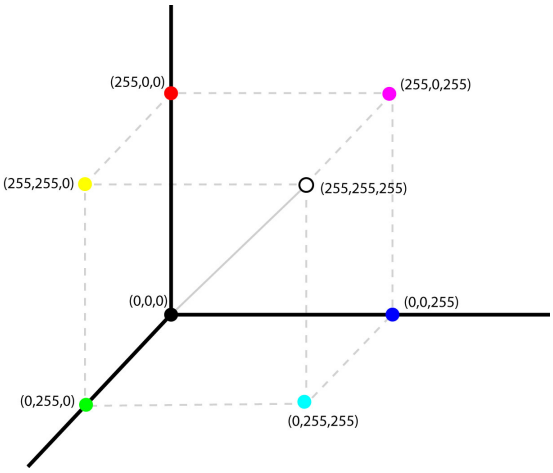


Figura 5.2: Cubo de Cores RGB (fonte [7]).

Espaço de Cores LMS

O LMS é um sistema de cores menos utilizado porém muito importante para esse trabalho no algoritmo de simulação de deficiência em cores. Nesse sistema uma cor

é representada pela resposta dos três tipos de cones no olho humano, através das coordenadas L (comprimento de onda longo, associado ao vermelho), M (comprimento de onda médio, associado ao verde) e S (comprimento de onda curto, associado ao azul).

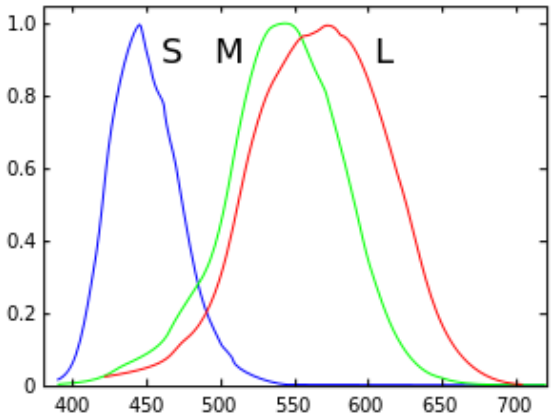


Figura 5.3: Representação do espaço de cores LMS (fonte [7]).

5.1.2 Detalhes da ferramenta de simulação

Dada uma imagem com pixels no modelo RGB com valores de cada componente $\in$ [0..255], o algoritmo consiste de 6 passos:

1. Calcular a quantidade fotométrica relativa para cada componente. Esse passo basicamente mapeia os valores $[R, G, B]$ no intervalo discreto [0..255] para valores $[R', G', B']$ em um intervalo contínuo [0..1], através da fórmula $X' \leftarrow (X/255)^{2.2}$ com X $\in$ R,G,B.

2. Normalizar as quantidades fotométricas relativas de forma a reduzir o domínio de cores, para que sejam obtidos valores válidos após o próximo passo. Essa etapa consiste de uma transformação linear $Y' \leftarrow \alpha * Y + \beta$, com Y $\in$ R', G', B' onde os valores de alfa e beta variam conforme o tipo de simulação.

3. Efetuar uma multiplicação por uma matriz C que faz a conversão do espaço de cores RGB normalizado para o espaço de cores LMS. Essa matriz é descrita em 5.1.2.

4. Efetuar uma multiplicação por uma matriz S que simula a ausência ou diminuição de um dos tipos de cone no espaço LMS. Essa matriz varia conforme o tipo de simulação.

5. Efetuar a multiplicação pela matriz C^{-1} que faz a conversão de volta do espaço LMS para o modelo RGB.

6. Mapear os valores no intervalo contínuo [0..1] de volta para o intervalo discreto [0..255] (operação inversa do passo 1).

$$C = \begin{bmatrix} 17.8824 & 3.45565 & 0.0299566 \\ 43.5161 & 27.1554 & 3.86714 \\ 4.11935 & 3.86714 & 1.46709 \end{bmatrix}$$

Note que, por causa dos passos 1 e 6, a ferramenta não pode ser representada apenas por transformações lineares e portanto não pode ser escrita como uma simples multiplicação do vetor [R G B] por uma matriz.

No entanto, é possível desenvolver um algoritmo simplificado que implementa apenas as etapas 3, 4 e 5. Esse algoritmo simplificado pode ser representado por uma multiplicação por uma matriz, mas não gera cores tão detalhadas quanto o descrito acima, devido a arredondamentos que são feitos.

No algoritmo simplificado podemos aproveitar a abordagem matricial disponibilizadas pelas classes Bitmap do Android e Mat do OpenCV, a qual é bem mais eficiente do que agir sobre componentes como é feito no algoritmo completo. Essa abordagem matricial é utilizada no caso do modo contínuo da ferramenta de simulação que usa OpenCV, já que a imagem da câmera do smartphone deve ser tratada em tempo real com o menor tempo de latência possível.

5.2 Parâmetros da ferramenta

Os parâmetros que mudam conforme o tipo de simulação (deuteranopia ou protanopia) são α, β e a matriz S:

Tabela 5.1: Parâmetros da ferramenta de simulação.

Tipo de simulação	α	β	S
Deuteranopia	0.957237	0.0213814	$\begin{bmatrix} 1 & 0.494207 & 0 \\ 0 & 0 & 0 \\ 0 & 1.24827 & 1 \end{bmatrix}$
Protanopia	0.992052	0.003974	$\begin{bmatrix} 0 & 0 & 0 \\ 2.02344 & 1 & 0 \\ -2.52581 & 0 & 1 \end{bmatrix}$

5.3 Algoritmo do filtro

No algoritmos, alguns passos foram unificados de forma a melhorar a eficiência da ferramenta.

Entrada: Image img, α, β, Matriz C, Matriz S
1 **para** $l \leftarrow 0$ **até** *img.largura()* **faça**
2 **para** $a \leftarrow 0$ **até** *img.altura()* **faça**
3 [R G B] $\leftarrow$ img.getPixel(l, a);
4 `// Passos 1 e 2`
5 $R^* \leftarrow \alpha * (R/255.0)^{2.2} + \beta$;
6 $G^* \leftarrow \alpha * (G/255.0)^{2.2} + \beta$;
7 $B^* \leftarrow \alpha * (B/255.0)^{2.2} + \beta$;
8 `// Passos 3 a 5. A matriz` A `é pré-calculada e` $A = C \times S \times C^{-1}$.
9 $[R^*G^*B^*] \leftarrow [R^*G^*B^*] \times A$;
10 `// Passo 6`
11 $R^* \leftarrow (R^*)^{1/2.2} * 255$;
12 $G^* \leftarrow (G^*)^{1/2.2} * 255$;
13 $B^* \leftarrow (B^*)^{1/2.2} * 255$;
14 [R G B] $\leftarrow [R^*G^*B^*]$;
15 **fim**
16 **fim**

Algoritmo 1: Algoritmo da ferramenta de simulação de deuteranopia e protanopia.

Entrada: Image img, Matriz C, Matriz S
1 **para** $l \leftarrow 0$ **até** *img.largura()* **faça**
2 **para** $a \leftarrow 0$ **até** *img.altura()* **faça**
3 [R G B] $\leftarrow$ img.getPixel(l, a);
4 `// A matriz` A `é pré-calculada e` $A = C \times S \times C^{-1}$.
5 [R G B] $\leftarrow$ [R G B] $\times A$;
6 **fim**
7 **fim**

Algoritmo 2: Algoritmo simplicado da ferramenta de simulação de deuteranopia e protanopia.

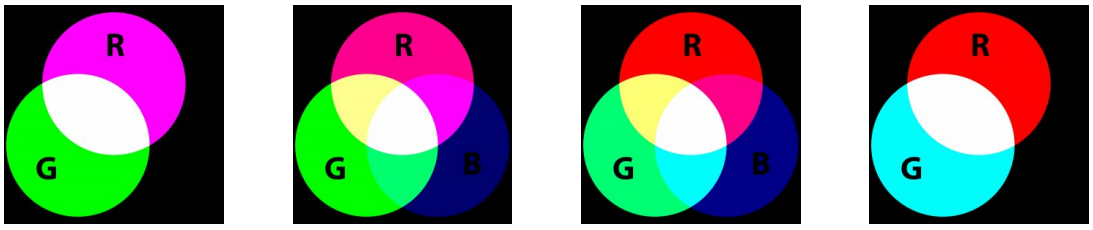
Capítulo 6

Filtro de Acessibilidade

6.1 Filtro de acessibilidade

O filtro de acessibilidade tem como objetivo melhorar a percepção de cores para tricromatas anômalos com deuteranomalia ou protanomalia e também para dicromatas com deuteranopia ou protanopia (ou seja, a maior parcela de deficientes em cores). O filtro se baseia na idéia de que embora o contraste entre tons vermelhos e verdes seja difícil de ser visto, os contrastes entre tons vermelhos e azuis e entre tons verdes e azuis são percebidos normalmente.

Considerando o espaço de cores RGB, o filtro de acessibilidade age apenas a sobre a componente azul de cada pixel da imagem. Conforme o valor do parâmetro i escolhido no filtro, ele transforma o contraste entre tons vermelhos e verdes em contraste entre tons azuis e verdes (alterando a intensidade da componente azul em pixels com alta intensidade da componente vermelha) ou transforma em contraste entre tons azuis e vermelhos (alterando a intensidade da componente azul em pixels com alta intensidade da componente verde).



(a) i = -1 (b) i = -0.5 (c) i = 0.5 (d) i = 1

Figura 6.1: Filtro de acessibilidade com diferentes valores de i.

6.2 Parâmetro do filtro

O filtro possui como parâmetro apenas um valor de intensidade contínuo $i \in [-1..1]$, que é controlado pelo usuário no smartphone em tempo real através de um slider. Esse valor i determina os valores dos três parâmetros do filtro de acessibilidade, α, β e γ:

$$\alpha = \begin{cases} 0, & \text{se} \quad i \geq 0, \\ -i, & \text{c.c.} \end{cases}$$

$$\beta = \begin{cases} 0, & \text{se} \quad i \leq 0, \\ i, & \text{c.c.} \end{cases}$$

$$\gamma = 1 - (\alpha + \beta)$$

6.3 Algoritmo do filtro

Entrada: Image img, α, β, γ
1 **para** $l \leftarrow 0$ **até** *img.largura()* **faça**
2 **para** $a \leftarrow 0$ **até** *img.altura()* **faça**
3 pixel $p \leftarrow$ img.getPixel(l, a);
4 $p_{azul} \leftarrow (p_{vermelho} * \alpha) + (p_{verde} * \beta) + (p_{azul} * \gamma)$;
5 **fim**
6 **fim**

Algoritmo 3: Algoritmo do filtro de acessibilidade

Como o filtro é matematicamente simples, efetuando apenas transformações lineares sobre a tripla [R G B] de um pixel, ele pode ser representado pela matriz de transformação M. Assim como na ferramenta de simulação, o uso de uma matriz de transformação é mais eficiente pois permite aproveitar a abordagem matricial disponibilizadas pelas classes Bitmap do Android e Mat do OpenCV.

$$M = \begin{bmatrix} 1 & 0 & \alpha \\ 0 & 1 & \beta \\ 0 & 0 & \gamma \end{bmatrix}$$

```
   Entrada: Image img, Matriz M
1  para l ← 0 até img.largura() faça
2  |   para a ← 0 até img.altura() faça
3  |   |   [R G B] ← img.getPixel(l, a);
4  |   |   [R G B] ← [R G B] × M;
5  |   fim
6  fim
```

Algoritmo 4: Algoritmo do filtro de acessibilidade com matriz de transformação

Capítulo 7

Ferramenta de Destaque de Cor

7.1 Ferramenta de destaque de cor

O filtro de acessibilidade, embora ajude o usuário a perceber contrastes entre tons das cores vermelha e verde, não possibilita o usuário a identificar, dada uma cor, em que regiões da imagem aquela cor ocorre. Essa necessidade ocorre por exemplo em imagens com legendas em cores em que é difícil ou impossível para o indivíduo com deficiência em cores identificar onde a cor de um item da legenda aparece.

A ferramenta de destaque de cor tem então como objetivo possibilitar o usuário a destacar uma determinada cor em uma imagem, possibilitando assim extrair informações relacionadas a essa cor mais facilmente. Ela consiste em, para cada pixel da imagem, calcular a distância entre a cor desse pixel e a cor escolhida para ser destacada pelo usuário. Caso essa distância seja suficientemente pequena, a ferramenta não altera a cor do pixel. Caso contrário, o pixel é convertido para a escala de cinza.

7.2 Parâmetros da ferramenta

A ferramenta possui como parâmetros a cor selecionada pelo usuário e valores de tolerância para as componentes vermelha, verde e azul, selecionáveis pelo usuário. Ao utilizar valores de tolerância individuais para cada componente, a ferramenta permite uma maior precisão no cálculo da distância entre as cores. Isso possibilita o usuário a, por exemplo, dar uma maior peso às diferenças na componente vermelha do que às diferenças na componente azul na hora do cálculo da distância entre as cores.

7.3 Cálculo da distância entre as cores

Entrada: umaComponente, outraComponente, tolerancia
1 $distanciaAoQuadrado \leftarrow (umaComponente - outraComponente)^2$;
2 **se** $tolerancia == 0$ **então**
3 | **retorna** $distanciaAoQuadrado == 0.0$;
4 **fim**
5 **retorna** $distanciaAoQuadrado/tolerancia^2$;

Função `calculaDistancia`(*umaComponente, outraComponente, tolerancia*)

Entrada: Pixel umPixel, Pixel outroPixel, tolerancias
1 $soma \leftarrow 0$;
2 **para** $i \in$ *vermelho, verde, azul* **faça**
3 | $soma \leftarrow soma + calculaDistancia(umPixel_i, outroPixel_i, tolerancias_i)$;
4 **fim**
5 **retorna** $soma < 1.0$;

Função `verificaDistanciaAceitavel`(*umPixel, outroPixel, tolerancias*)

Observe que o cálculo de distância aceitável não verifica se a distância entre cada uma das componentes está na faixa de tolerância estabelecido pelo usuário independentemente, mas sim se a soma acumulada está dentro de um limite aceitável. Dessa forma, caso o usuário escolha valores pequenos de tolerância, somente cores realmente próximas à cor escolhida pelo usuário são destacadas. Caso o usuário queira ampliar a gama de cores selecionadas, basta aumentar o valor de tolerância das componentes.

7.4 Conversão para escala de cinza

Na conversão de uma cor para escala de cinza, as três componentes do modelo RGB não têm pesos iguais na conversão já que a visão em um humano tricromata é mais sensível ao verde (que tem assim um maior peso na conversão) e menos sensível ao azul (que tem portanto um menor peso na conversão)[1].

Entrada: Pixel p
1 $p_{vermelho} \leftarrow p_{vermelho} * 0.2126$;
2 $p_{verde} \leftarrow p_{verde} * 0.7152$;
3 $p_{azul} \leftarrow p_{azul} * 0.0722$;

Função `converteEmEscalaDeCinza`(p)

7.5 Algoritmo da ferramenta de destaque de cor

Utilizando as funções verificaDistanciaAceitavel e converteEmEscalaDeCinza, descritas em 6 e 7 respectivamente, podemos escrever o algoritmo 8 para destaque de cor. Note que como o algoritmo efetua diversos cálculos que não são transformação lineares, especialmente na parte do cálculo de distância, ele não é eficiente o suficiente para ser aplicado em uma imagem em tempo real (continuamente).

```
   Entrada: Image img, Pixel corEscolhida, tolerancias
1  para l ← 0 até img.largura() faça
2  |  para a ← 0 até img.altura() faça
3  |  |  pixel p ← img.getPixel(l, a);
4  |  |  se não verificaDistanciaAceitavel(p, corEscolhida, tolerancias) então
5  |  |  |  converteEmEscalaDeCinza(p);
6  |  |  fim
7  |  fim
8  fim
```

Algoritmo 8: Algoritmo para destaque de cor

Capítulo 8

Sistema e sua Arquitetura

Nesse capítulo é feita uma breve explicação sobre a arquitetura do sistema e como as classes de implementação das ferramentas interagem com as bibliotecas do Android e OpenCV.

8.1 Descrição dos pacotes do software

A descrição dos pacotes Java que compõe o sistema é feita a seguir:

- **br.usp.ime.tcc.activities.colorpicker**

  Responsável pelas atividades (telas do aplicativo) relacionadas à seleção de cor em uma foto.

- **br.usp.ime.tcc.activities.components**

  Responsável pelo carregamento de componentes que são utilizadas em todas as atividades do sistema.

- **br.usp.ime.tcc.activities.filter**

  Responsável pelas telas de seleção do modo (galeria, câmera ou contínuo) de cada ferramenta.

- **br.usp.ime.tcc.activities.filter.bitmap**

  Responsável pelas telas onde as ferramentas são aplicados em imagens carregadas da câmera ou da galeria, utilizando a classe bitmap do Android.

- **br.usp.ime.tcc.activities.filter.continuous**

  Responsável pelas telas onde as ferramentas são aplicadas de modo contínuo sobre a imagem da câmera, utilizando a biblioteca OpenCV.

- **br.usp.ime.tcc.activities.settings**

  Responsável pela atividade de configuração do usuário e também por salvar e carregar essas configurações.

- **br.usp.ime.tcc.activities**

  Responsável por atividades que não se enquadram em uma das categorias acima.

- **br.usp.ime.tcc.filter.colorhighlight**

  Responsável pelo algoritmo da ferramenta de destaque cor.

- **br.usp.ime.tcc.filter.simulation**

  Responsável pelo algoritmo da ferramenta de simulação de deficiência em cores.

- **br.usp.ime.tcc.filter.visocor**

  Responsável pelo algoritmo do filtro de acessibilidade Visocor.

- **br.usp.ime.tcc.utils**

  Responsável por carregar imagens no aplicativo e salvar imagens alteradas no dispositivo, entre outras.

8.2 Integração com o Android e OpenCV

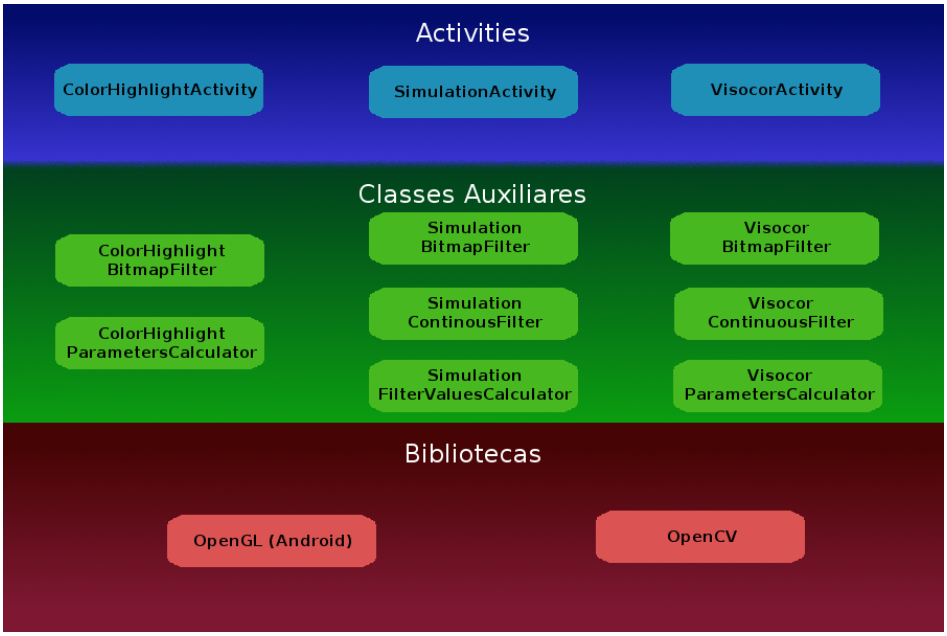


Figura 8.1: Integração das ferramentas com o Android e OpenCV.

Na figura 8.1 temos a arquitetura do sistema conforme o ponto de vista de integração com as bibliotecas do Android e OpenCV. As atividades responsáveis por cada uma das ferramentas faz uso das classes responsáveis pela implementação dos algoritmos, que por sua vez fazem uso de chamadas da biblioteca OpenCV e de classes do Android que manipulam Bitmaps.

Capítulo 9

Manual do Usuário

Nesse capítulo é apresentado um manual do usuário, embora o aplicativo tente ser o mais intuitivo possível e tenha botões de ajuda ao usuário.

Para cada ferramenta é demonstrado um modo de entrada de imagem (da câmera, da galeria ou contínuo) e o que o usuário precisa fazer para aplicar a ferramenta. É mostrada também a tela de configurações, onde o usuário pode escolher valores padrões para os parâmetros das ferramentas e também alterar opções mais avançadas como a tolerância para cada componente RGB na ferramenta de destaque de cor.

9.1 Filtro de Acessibilidade

O modo de entrada apresentado para o filtro de acessibilidade será o modo contínuo. Ao abrir o aplicativo, a tela principal (figura 9.1) é mostrada. Nessa tela são mostradas as opções para as três ferramentas do trabalho na parte principal, assim como o menu de configurações que é mostrado na ActionBar.

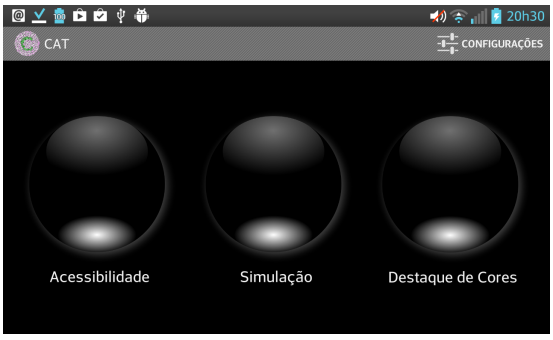


Figura 9.1: Tela principal.

Selecionando a ferramenta de acessibilidade, são mostradas as três opções de entrada conforme pode ser visto em 9.2.



Figura 9.2: Tela de seleção de modo do filtro de acessibilidade.

Após selecionar o modo contínuo, o aplicativo começa a processar a imagem da camêra traseira continuamente e o slider do parâmetro do filtro é mostrado na parte inferior da tela, podendo facilmente ser alterado pelo usuário (figura 9.3).

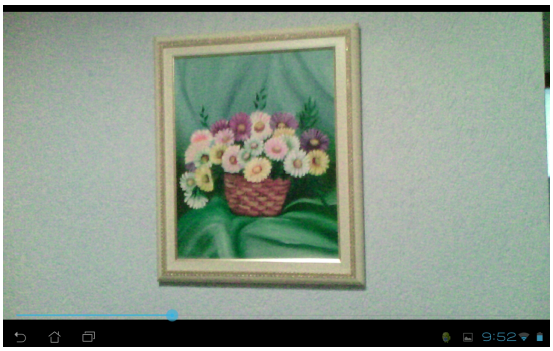


Figura 9.3: Tela do filtro de acessibilidade, modo contínuo.

O modo contínuo depende do OpenCV para ser executado e por isso o usuário precisa ter instalado um aplicativo adicional do OpenCV em seu dispositivo móvel. Caso o usuário não tenha esse pacote instalado, uma mensagem que o encaminha à página de download do aplicativo adicional é mostrada ao selecionar esse modo.

Além disso, por depender do OpenCV, essa opção de entrada é a única que não possibilita ao usuário salvar a imagem processada em seu dispositivo. No entanto isso pode ser feito em alguns dispositivos que possuem atalhos para captura de tela.

9.2 Ferramenta de simulação de deficiência em cores

O modo de entrada apresentando para a ferramenta de simulação será a partir da câmera. Na tela principal, após a escolha a ferramenta de simulação são mostradas as três opções de entrada assim como um seletor de que tipo de simulação deverá ser feita (figura 9.4).



Figura 9.4: Tela de seleção de modos da ferramenta de simulação.

Selecionando a opção "Câmera", o aplicativo de câmera é chamado para que o usuário possa tirar uma foto e, após confirmar a foto (figura 9.5), o aplicativo desenvolvido aplica a simulação que havia sido escolhida na imagem.

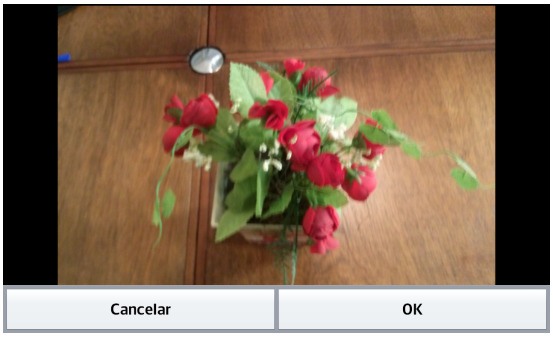


Figura 9.5: Tela de confirmação da imagem obtida da câmera.

A tela de resultado pode ser vista na figura 9.6. Foi implementada uma funcionalidade de zoom na foto processada, para que o usuário possa ver os detalhes mais claramente caso necessário. Além disso, na ActionBar são mostradas as opções de salvar a foto processada no dispositivo móvel ou descartar a mesma.

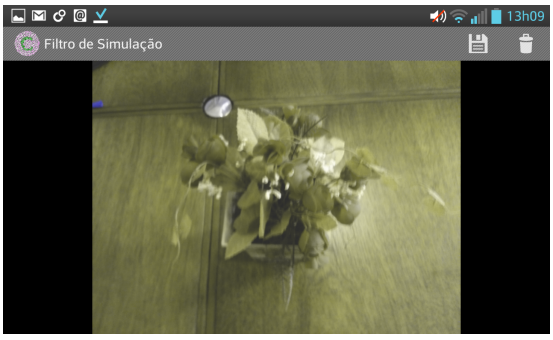


Figura 9.6: Tela da ferramenta de simulação.

9.3 Ferramenta de destaque de cor

O modo de entrada apresentado para a ferramenta de destaque de cor será a partir da galeria. Partindo da tela principal, após a escolha da ferramenta de destaque de cor são mostradas as opções de entrada (figura 9.7).

Note que a opção modo contínuo não está entre as opções, pois a ferramenta de destaque de cor é matematicamente mais complexa e por isso não é rápida o bastante para ser aplicada a cada frame em tempo real.

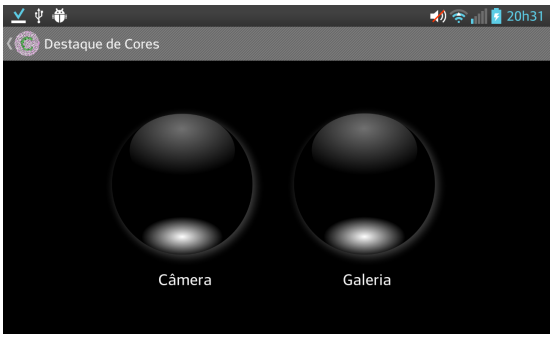


Figura 9.7: Tela de seleção de modos da ferramenta de destaque de cor.

Selecionando o modo "Galeria", as fotos disponíveis no dispositivo móvel do usuário são mostradas para que ele escolha a imagem a qual será aplicada a ferramenta (figura 9.8).

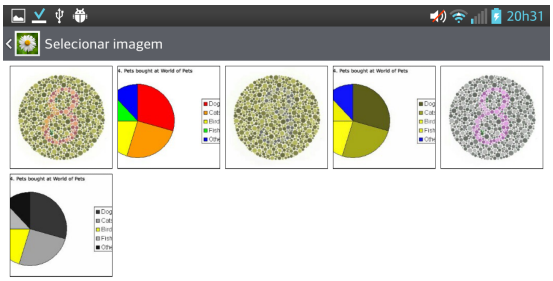


Figura 9.8: Seleção de imagem a partir do dispositivo do usuário.

Após a escolha, a ferramenta é aplicada destacando a cor padrão das configurações do usuário (que pode ser alterada conforme vai ser explicado na sessão de configurações do usuário 9.4). Nessa tela também é mostrado o botão para seleção de cor a ser destacada (figura 9.9).

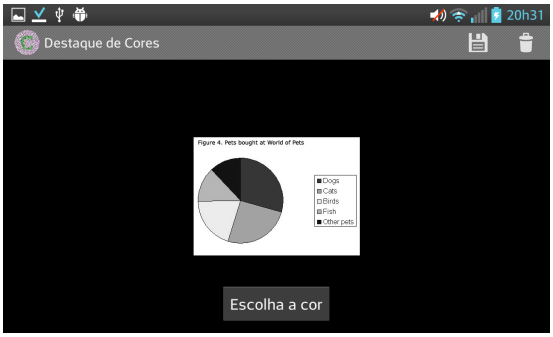


Figura 9.9: Tela da ferramenta de destaque de cor.

Ao clicar no botão "Escolha a cor", a tela do seletor de cor é mostrada. Nessa tela (figura 9.10), o usuário pode dar zoom na imagem para escolher a cor desejada mais facilmente, e para selecionar uma cor basta dar um clique longo no ponto da tela com a desejada (que é então mostrada na pequena caixa de exemplo acima da imagem). A ActionBar dessa tela possui, além do botão de confirmar ou descartar a cor escolhida, um botão de ajuda caso o usuário tenha alguma dificuldade com esse processo.

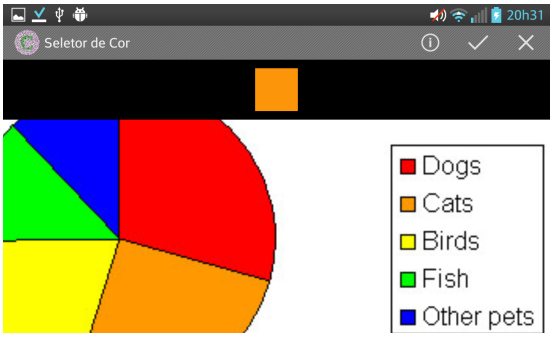


Figura 9.10: Tela do seletor de cor da ferramenta de destaque de cor.



Figura 9.11: Tela da ferramenta de destaque de cor: resultado.

Após a escolha de cor a ser destacada, o resultado pode ser visto na figura 9.11.

Novamente, são apresentadas as opções de salvar ou descartar a imagem processada.

9.4 Configurações do usuário

A tela de configurações do usuário, acessível a partir da tela principal, pode ser vista na figura 9.13. Nessa tela o usuário tem acesso aos valores padrões usados pelo aplicativo (como por exemplo o valor inicial do parâmetro do filtro e cor de destaque padrão), além de opções mais avançadas como os valores de tolerância de cada componente RGB para a ferramenta de destaque de cor.

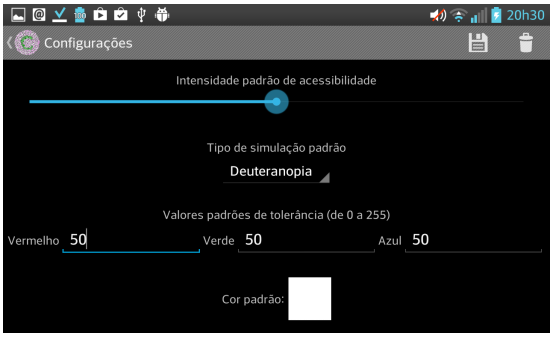


Figura 9.12: Tela de configurações do usuário.

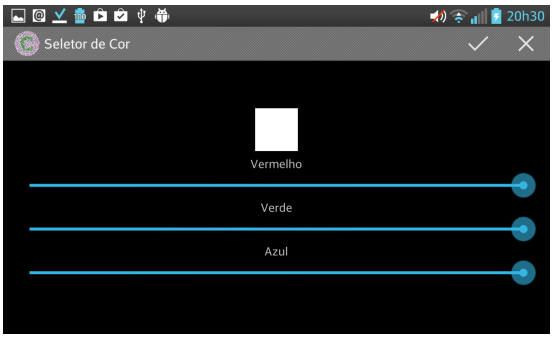


Figura 9.13: Tela de seleção de cor padrão a ser destacada.

Capítulo 10

Resultados

10.1 Testes e cobertura

A técnica de desenvolvimento dirigido por testes se mostrou muito eficaz: foram escritos aproximadamente 170 testes e esses testes cobriram quase 92% do projeto. Mais importante que isso, durante o desenvolvimento não houveram grandes problemas com bugs difíceis de se achar ou de se resolver: eles eram sempre achados e resolvidos rapidamente devido à constante cobertura elevada durante todo o projeto, que rapidamente indicavam em que método e linha estava o bug.

Portanto, embora a produtividade inicial tenha sido reduzida pelo tempo gasto pesquisando e pensando não apenas como implementar determinada funcionalidade mas também seus testes, o tempo perdido definitivamente foi menor que o tempo que teria sido gasto procurando bugs, fazendo depuração de código e resolvendo-o.

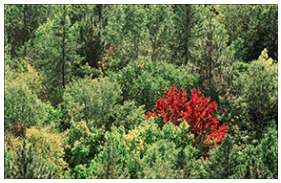
A cobertura do projeto detalhada por pacotes e classes pode ser vista no relatório da ferramenta EclEmma em 10.1. Percebe-se que a maior parte dos pacotes e classes possui uma cobertura de pelo menos 80% das instruções.

No entanto, algumas classes que fazem uso do OpenCV possuem uma cobertura muito menor já que não foi possível testar diversos métodos devido à dificuldade do arcabouço de testes Robolectric em lidar com dependências externas. Como no geral as classes que fazem uso do OpenCV são responsáveis apenas por multiplicações de matrizes, a falta de testes não é um problema grave.

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
src - TCC	91.9 %	3,083	270	3,353
br.usp.ime.tcc.filter.visocor	74.9 %	254	85	339
VisocorContinousFilter.java	0.0 %	0	85	85
VisocorBitmapFilter.java	100.0 %	183	0	183
VisocorParametersCalculator.java	100.0 %	71	0	71
br.usp.ime.tcc.utils	79.9 %	226	57	283
Constants.java	0.0 %	0	3	3
BitmapLoader.java	71.9 %	97	38	135
Utils.java	79.2 %	61	16	77
FileSaver.java	100.0 %	68	0	68
br.usp.ime.tcc.activities	82.5 %	226	48	274
ContinousFilterActivity.java	68.6 %	96	44	140
MainActivity.java	97.0 %	130	4	134
br.usp.ime.tcc.activities.filter	90.9 %	261	26	287
ColorHighlightFilterActivity.java	77.8 %	28	8	36
SimulationFilterActivity.java	88.4 %	61	8	69
FilterActivity.java	92.1 %	116	10	126
VisocorFilterActivity.java	100.0 %	56	0	56
br.usp.ime.tcc.activities.colorpicker	91.6 %	395	36	431
ImageColorPickerActivity.java	79.4 %	131	34	165
SettingsColorPickerActivity.java	98.3 %	113	2	115
ColorPickerActivity.java	100.0 %	151	0	151
br.usp.ime.tcc.filter	92.9 %	13	1	14
BitmapFilter.java	92.9 %	13	1	14
br.usp.ime.tcc.activities.filter.bitmap	97.5 %	549	14	563
BitmapFilterActivity.java	95.0 %	153	8	161
VisocorBitmapFilterActivity.java	98.0 %	144	3	147
ColorHighlightBitmapFilterActivity.java	98.6 %	219	3	222
SimulationBitmapFilterActivity.java	100.0 %	33	0	33
br.usp.ime.tcc.activities.settings	99.5 %	611	3	614
SettingsActivity.java	99.4 %	500	3	503
SettingsManager.java	100.0 %	111	0	111
br.usp.ime.tcc.activities.components	100.0 %	188	0	188
ComponentUtils.java	100.0 %	188	0	188
br.usp.ime.tcc.filter.colorhighlight	100.0 %	201	0	201
ColorHighlightBitmapFilter.java	100.0 %	107	0	107
ColorHighlightParametersCalculator.java	100.0 %	94	0	94
br.usp.ime.tcc.filter.simulation	100.0 %	159	0	159
SimulationBitmapFilter.java	100.0 %	47	0	47
SimulationFilterValuesCalculator.java	100.0 %	112	0	112

Figura 10.1: Relatório detalhado por pacotes e classes da cobertura de testes.

10.2 Ferramenta de simulação de deficiência em cores



(a) Imagem original



(b) Simulação de deuteranopia



(c) Simulação de protanopia

Figura 10.2: Resultados da simulação na imagem de vegetação.

Resultados da simulação de deuteranopia e protanopia sobre duas imagens podem ser vistos nas figuras 10.2 e 10.3. Nessas imagens é possível perceber como pode ser difícil para um deficiente visual em cores notar os constrastes entre vermelho e verde, principalmente no caso da imagem da vegetação. Nota-se também que os dois tipos de simulação podem gerar imagens bem semelhantes, como no caso da imagem do

semáforo.



(a) Imagem original



(b) Simulação de deuteranopia



(c) Simulação de protanopia

Figura 10.3: Resultados da simulação na imagem de um semáforo.

10.3 Filtro de acessibilidade Visocor



(a) Imagem original



(b) Simulação de deuteranopia



(c) Simulação de protanopia



(d) Imagem com filtro de acessibilidade aplicado



(e) Simulação de deuteranopia da imagem com filtro aplicado

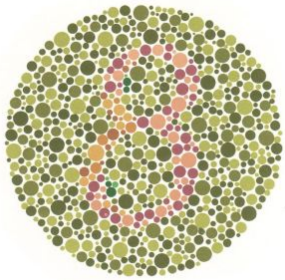


(f) Simulação de protanopia da imagem com filtro aplicado

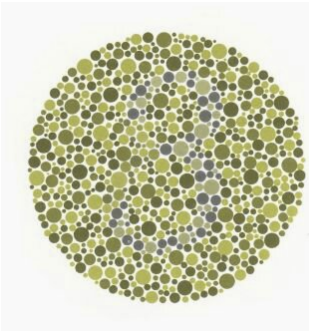
Figura 10.4: Resultados do filtro de acessibilidade aplicado sobre uma imagem com contraste verde com vermelho problemático.

Resultados do filtro de acessibilidade Visocor podem ser vistos nas figuras 10.4 e 10.5, onde o parâmetro de intensidade utilizado para o filtro foi i = -1. Após a aplicação do filtro, o contraste problemático verde com vermelho foi transformado em contrastes que

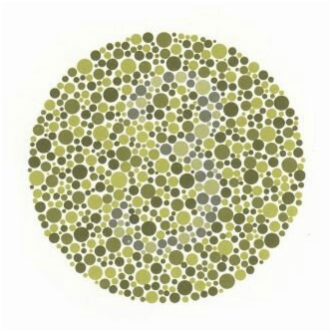
o usuário portador de deuteranopia ou protanopia pode identificar facilmente, inclusive em imagens criadas com contrastes extremamente problemáticos como é o caso do cartão de Ishihara.



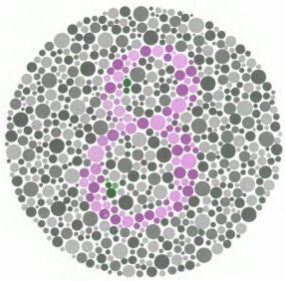
(a) Imagem original



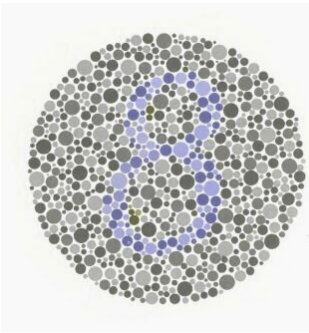
(b) Simulação de deuteranopia



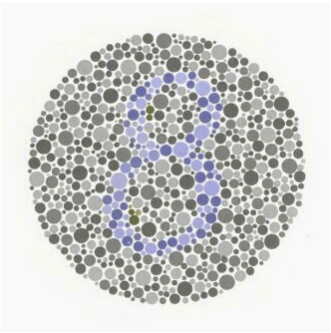
(c) Simulação de protanopia



(d) Imagem com filtro de acessibilidade aplicado



(e) Simulação de deuteranopia da imagem com filtro aplicado



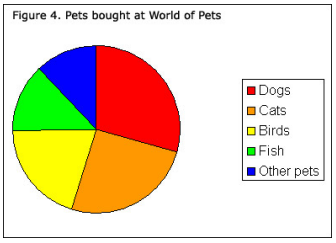
(f) Simulação de protanopia da imagem com filtro aplicado

Figura 10.5: Resultados do filtro de acessibilidade aplicado sobre um cartão de Ishihara.

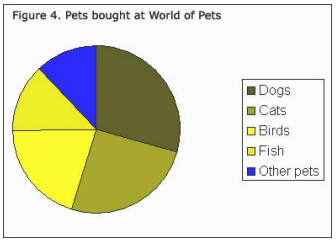
10.4 Ferramenta de destaque de cor

O resultado da ferramenta de destaque de cor pode ser visto na figura 10.6, onde a mesma é aplicada sobre um gráfico circular. Esse é o típico caso de uso para qual essa ferramenta foi projetada: uma figura em que o problema não é o contraste entre as cores (como no caso do cartão de Ishihara), mas sim como identificar onde uma determinada cor aparece na imagem.

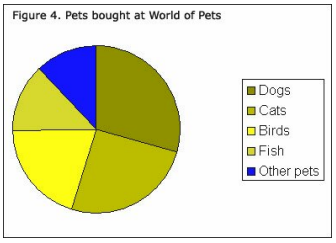
Após a seleção de cor pelo usuário e a aplicação da ferramenta, todas as cores que não são próximas o suficiente da cor selecionada foram covertidas para escala de cinza. Como somente a cor selecionada mantém sua cor original, fica extremamente fácil indentificar onde essa cor aparece.



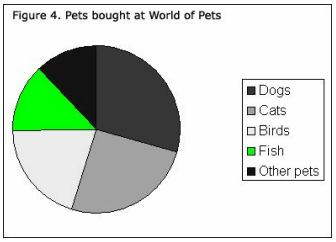
(a) Imagem original



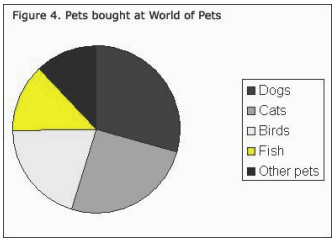
(b) Simulação de deuteranopia



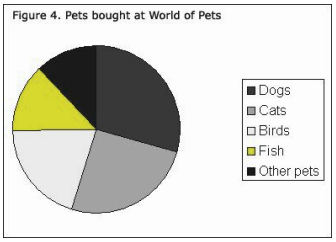
(c) Simulação de protanopia



(d) Imagem após destaque de cor



(e) Simulação de deuteranopia com destaque de cor



(f) Simulação de protanopia com destaque de cor

Figura 10.6: Resultados da ferramenta de destaque de cor em um gráfico.

Capítulo 11

Conclusões e Trabalhos Futuros

11.1 Conclusões

Apesar do grande número de ferramentas gráficas que buscam melhorar a qualidade da imagem para interpretação humana e seus constantes avanços, o número de ferramentas que tenta melhorar a qualidade de imagens para deficientes visuais em cores de um modo fácil e rápido ainda é bastante limitado. Além disso, essas ferramentas muitas vezes não levam em consideração que uma determinada deficiência possui diferentes graus e por isso acabam alterando a imagem demasiadamente de maneira desnecessária.

Levando isso em consideração esse trabalho foi desenvolvido, consistindo de três funcionalidades: uma ferramenta de simulação de deficiência em cores, um filtro de acessibilidade parametrizável para deficientes em cores e uma ferramenta para destaque de cor em imagens.

A ferramenta de simulação é de uso preventivo, para que o conteúdo gerado seja acessível. O filtro parametrizável foi desenvolvido para melhora da qualidade visual para portadores de deficiência visual em cores com diferentes graus. Por fim, a ferramenta de destaque de cor em imagens foi desenvolvida para extração de informações associadas a uma dada cor, como é o caso de imagens com legendas coloridas.

Todas as ferramentas se mostraram de uso fácil e intuitivo, como é esperado de ferramentas desenvolvidas para dispositivos móveis, gerando uma experiência melhor ao usuário. O uso por deficientes visuais em cores e resultados apresentados usando ferramentas de simulação de deficiência em cores mostram que as ferramentas de fato conseguem possibilitar e facilitar a extração de informações em imagens onde isso não era possível.

11.2 Informações do aplicativo na Google Play

A primeira versão do aplicativo foi disponibilizada na loja de aplicativos Google Play no dia 16 de outubro de 2013. Após o feedback e sugestão de alguns usuários, assim como algumas correções, foram lançadas as versões 2.0 e 3.0, sendo essa última lançada no dia 26 de outubro de 2013.

A página do aplicativo pode ser encontrada em https://play.google.com/store/apps/details?id=br.usp.ime.tcc.activities e até o momento possui cerca de 50 usuários. O aplicativo foi desenvolvido seguindo as práticas de internacionalização do Android desde o início e atualmente possui versões em português e inglês. Por esse motivo, embora a maioria dos usuários sejam brasileiros, existe também uma grande parcela de usuários de outros países (conforme pode ser visto na figura 11.1).

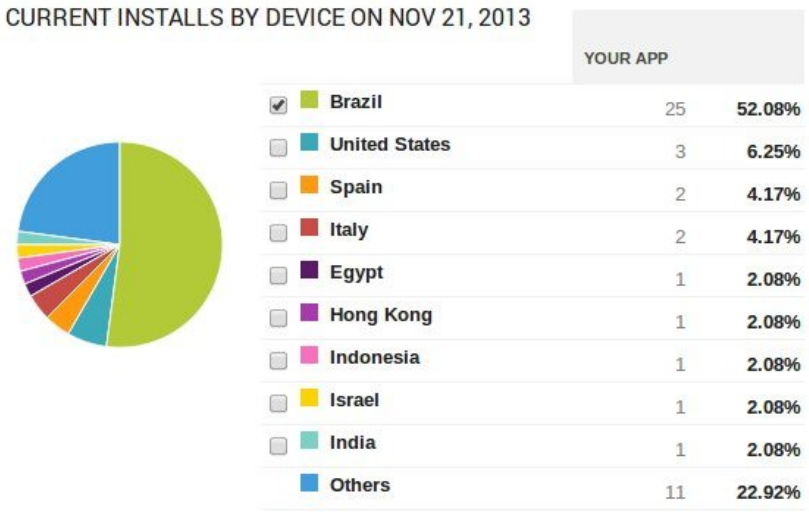


Figura 11.1: Gráfico de usuários do aplicativo.

11.3 Trabalhos futuros

O desenvolvimento para outras plataformas móveis importantes, como o iOS da Apple e o Windows Mobile da Microsoft, é uma das possibilidades de continuação do trabalho, pois levariam essa ferramenta de acessibilidade à grande maioria do mercado de dispositivos móveis.

Outra possibilidade, que inclusive já havia sido mencionada no trabalho anterior em que esse trabalho se baseou, seria a adição de um filtro de acessibilidade para tritanomalia e tritanopia. No entanto, isso não foi feito pois esses tipos de deficiência em cores é muito mais raro que os distúrbios que dificultam a distinção de contrastes verde com vermelho.

Uma outra função que poderia ter sido adicionada nesse aplicativo é uma ferramenta de testes que diagnostica se o usuário possui ou não algum tipo de deficiência em cores através do uso de cartões de Ishihara. No entanto, uma rápida pesquisa na loja de aplicativos Android já mostra que existem diversos aplicativos especializados em fazer somente isso e por isso também foi descartada.

Referências Bibliográficas

[1] F. Viénot, H. Brettel, and J. Mollon. Digital video colourmaps for checking the legibility of displays by dichromats. *Colour Research and Application*, 24(4):243–252, August 1999.

[2] B. Jenny and N. Kelso. Color design for the color vision impaired. *Cartographic Perspectives*, 58:61–67, 2007.

[3] A. Asato and R. Gonçalves. Visocor - sistema de acessibilidade visual. Trabalho de Formatura Supervisionado, 2009. Instituto de Matemática e Estatística - Universidade de São Paulo.

[4] S. Palmer. *Vision Science: Photons to Phenomenology*. Bradford Books, 1999.

[5] J. Amabis and G. Martho. *Biologia dos Organismos: A diversidade dos seres vivos*. São Paulo: Moderna, 2004.

[6] S. Merin. *Inherited Eye Diseases: Diagnosis and Clinical Management*. CRC Press, 2013.

[7] The free enciclopedia wikipedia. http://en.wikipedia.org/. Acessada em 22 de novembro de 2013.

[8] W. Lee. *Beginning Android 4 Application Development*. Wrox Press Ltd., 2012.

[9] R. Meier. *Professional Android 4 Application Development*. Wrox Press Ltd., 2012.

[10] G. Nudelman. *Android Design Patterns: Interaction Design Solutions for Developers*. John Wiley & Sons, 2013.

[11] Actionbarsherlock website. http://actionbarsherlock.com/. Acessada em 22 de novembro de 2013.

[12] Android developer website. http://developer.android.com/. Acessada em 22 de novembro de 2013.

[13] Opencv website. http://opencv.org/. Acessada em 22 de novembro de 2013.

[14] A. Hunt and D. Thomas. *Pragmatic Unit Testing: In Java with Junit*. Pragmatic Bookshelf, 2003.

[15] Junit website. http://junit.org/. Acessada em 22 de novembro de 2013.

[16] Robolectric website. http://robolectric.org/. Acessada em 22 de novembro de 2013.

[17] K. Beck. *Test Driven Development: By Example*. Addison-Wesley Longman Publishing Co., Inc., 2002.

[18] Eclemma website. http://www.eclemma.org/. Acessada em 22 de novembro de 2013.

[19] M Aniche. *Test-Driven Development: Teste e Design no Mundo Real*. Casa do Código, 2013.

[20] Mockito website. https://code.google.com/p/mockito/. Acessada em 22 de novembro de 2013.

[21] Powermock website. https://code.google.com/p/powermock/. Acessada em 22 de novembro de 2013.

[22] S. Freeman and N. Pryce. *Growing Object-Oriented Software, Guided by Tests*. Addison-Wesley Professional, 2009.

[23] R. Gonzalez and R. Woods. *Digital Image Processing*. Addison-Wesley Longman Publishing Co., Inc., 2001.

Parte II

Parte Subjetiva

Capítulo 12

Decisões, Desafios e Frustações

No começo de 2012 procurei o professor Hirata para ser meu orientador e ele me apresentou diversas idéias de projetos relacionados à área de Visão Computacional e Processamento de Imagens. Como minha idéia era trabalhar com desenvolvimento para dispositivos móveis e a continuação do trabalho Visocor me dava essa possibilidade, além de parecer um trabalho muito interessante e útil, esse foi o trabalho escolhido.

No entanto, no meio do ano de 2012 surgiu a possibilidade de fazer um semestre de intercâmbio no Reino Unido. Como ainda estava no começo do trabalho e não havia possibilidade de continuar o trabalho estando no exterior, a disciplina foi trancada e combinei com meu orientador de continuar no ano seguinte.

No ano de 2013, como restava apenas uma disciplina eletiva a ser feita, não tive tantos problemas com tempo apesar do estar fazendo também um estágio. No primeiro semestre talvez não tenha conseguido adiantar tanto o trabalho quanto gostaria por estar cursando também a disciplina de Laboratório de Programação Extrema, mas sempre tentei manter o ritmo trabalhando tanto no software como nos estudos e monografia.

No começo do desenvolvimento do software, a dificuldade estava sendo bem grande pois, apesar de ter feito um EP em Android na disciplina de Computação Móvel, minha experiência com Android não era suficiente para me sentir confortável um projeto bem maior. Com isso, muitas vezes surgiam (e ressurgiam) bugs que tomavam um bom tempo até serem resolvidos.

Em junho, após ter aprendido sobre desenvolvimento dirigido por testes na disciplina de Laboratório de Programação Extrema, decidi então recomeçar o software aplicando essa metodologia. Achei que era boa idéia, já que além de ser uma nova experiência desenvolvendo software de um jeito que não tinha feito no curso, a grande quantidade de testes escritos me ajudaria com o problema com bugs que tinha até o momento.

Essa decisão foi, sem dúvidas, a melhor e mais importante para o trabalho. No começo os desafios aumentaram, já que além de aprender Android estava tendo que aprender simultaneamente como fazer testes para Android e como aplicar a metodologia de desenvolvimento de forma correta. No entanto, com o tempo e experiência, as vantagens se mostraram muito grandes e o desenvolvimento do aplicativo passou a se dar muito mais rapidamente e facilmente, e os problemas com bugs eram resolvidos

de maneira bem mais rápida.

Outro desafio encontrado no desenvolvimento do software foi como lidar com a mudança de orientação do dispositivo (modo retrato e modo paisagem), já que não queria forçar o usuário a usar o aplicativo em uma orientação específica. O problema se dá ao fato que o Android recria a tela quando ocorre essa mudança e nem todas as variáveis que estavam sendo usadas são mantidas, o que gerava perda de informações em algumas das telas.

Além disso tive alguns problemas com a leitura e gravação de arquivos de imagens em diferentes dispositivos, que só foram descobertos após o aplicativo ter sido publicado na Google Play Store. Isso ocorria pois a maneira que essas operações eram realizadas em alguns dispositivos não suportavam a leitura ou gravação de metadados no arquivo de imagem que eram essenciais para carregar a foto no aplicativo.

Escrever a monografia foi outro desafio, já que a escrita não é uma atividade realizada muito freqüentemente no curso e é bem cansativa. No entanto, a estratégia de sempre ir evoluindo as duas partes do trabalho (software e monografia) simultaneamente, evitando deixar a monografia toda para o final, me ajudou muito com isso. Além disso, minha experiência em escrever relatórios de laboratório enquanto ainda estava cursando o curso de Química Ambiental me ajudou bastante.

Por fim resta citar em manter o blog atualizado, já que isso se provou um tanto difícil para quem não tem costume de fazer algo parecido e muitas vezes acabou esquecido. Também muitas vezes, no começo e meio do trabalho, sentia que não tinha conteúdo o suficiente para escrever sobre, mesmo após uma reunião com o orientador, e por isso não escrevia nada. Somente mais para o fim do trabalho, quando começaram a surgir resultados e os próximos passos a cada semana começaram a ficar bem definidos, ficou mais fácil de fazer postagens realmente informativas e úteis sobre o andamento do trabalho.

Capítulo 13

Disciplinas Mais Relevantes e Importância do Intercâmbio

Disciplinas mais relevantes

As disciplinas mais importantes, não apenas para o trabalho apresentado mas também para minha formação durante o curso, foram:

- **MAC0110** - Introdução à Computação

  MAC0122 - Princípios de Desenvolvimento de Algoritmos

  Disciplinas fundamentais para começar a entender o desenvolvimento de algoritmos e aprender a programar, pois eu não tinha nenhuma experiência prévia com isso.

- **MAC0323** - Estruturas de Dados

  MAC0328 - Algoritmos em Grafos

  MAC0338 - Análise de Algoritmos

  Algumas das minhas disciplinas favoritas do curso, por ensinar conteúdos mais avançados e muito importantes que com certeza serão muito utilizados na minha carreira. Em particular, com a disciplina de análise de algoritmos aprendi diversas técnicas de desenvolvimento de algoritmos e passei a dar muito mais atenção à eficiência desses.

- **MAC0211** - Laboratório de Programação I

  MAC0242 - Laboratório de Programação II

  Disciplinas importantes por serem os primeiros projetos grandes desenvolvidos e também os primeiros projetos em equipe.

- **MAC0417** - Visão e Processamento de Imagens

  MAT0139 - Álgebra Linear Para Computação

Disciplinas importantes para o trabalho, principalmente pela parte de como fazer processamento de imagens de forma eficiente. A disciplina de álgebra linear também foi útil pois trabalhei muito com vetores e matrizes para garantir a eficiência das ferramentas descritas no trabalho.

- **MAC0463** - Computação Móvel

  Foi a disciplina onde desenvolvi meu primeiro projeto em Android, embora em uma escala bem menor que o trabalho apresentado aqui.

- **MAC0327** - Desafios de Programação

  MAC0342 - Laboratório de Programação Extrema (XP)

  Disciplinas relevantes para melhorar minhas técnicas de programação por serem ambas extremamente práticas, embora sejam bem diferentes.

  Desafios foca em diversos algoritmos para resolver diversos problemas pequenos e é ótima para relembrar e treinar técnicas e algoritmos que aprendi durante todo o curso.

  Já em XP temos um grande projeto em que trabalhamos em todo o semestre em equipe e aprendemos diversos conceitos e técnicas de métodos ágeis, e foi sem dúvida a matéria que mais me preparou profissionalmente. Além disso, teve uma grande importância para o trabalho realizado pela atenção dada a testes automatizados e pela introdução ao desenvolvimento dirigido por testes (TDD) que foi utlizado nesse trabalhado.

- **MAC0420** - Introdução à Computação Gráfica

  MAC0331 - Geometria Computacional

  MAC0315 - Programação Linear

  MAC0320 - Introdução à Teoria dos Grafos

  MAC0438 - Programação Concorrente

  MAC0325 - Otimização Combinatória

  Outras disciplinas que me estimularam bastante durante o curso, seja por minha afinidade com elas ou pelas dificuldades que tive que superar durante o curso.

Importância do intercâmbio

No segundo semestre de 2012 tive a oportunidade de estudar um semestre na University of Leeds, no Reino Unido, com bolsa de mérito acadêmico da USP. Durante esse semestre tive a oportunidade de estudar em uma das melhores universidades do Reino Unido (com uma enorme quantidade de alunos internacionais), conhecer as diferenças do ensino em outro país, entrar em contato com diferentes culturas, conhecer outros países e fazer amigos de todos os cantos do mundo, fazendo dessa uma experiência inesquecível.

Como restavam (além do TCC) apenas duas matérias optativas eletivas para finalizar o currículo do BCC, eu pude escolher as disciplinas que mais me interessavam

na University of Leeds. Lá cursei a disciplina de Otimização Combinatória, Desenvolvimento para Web, Prática de Resolução de Problemas (um pouco semelhante com nossas disciplinas de Laboratório de Programação) e Francês Básico.

Especialmente nos cursos de Otimização Combinatória pude notar o quanto a formação dada no BCC é forte, pois não tive muitos problemas com assuntos complexos com os quais meus colegas de classe claramente tinham. O sistema de ensino de lá também me ajudou a desenvolver a prática de estudar assuntos mais profundamente por mim mesmo, já que em Leeds as horas em aula em cada disciplina são bem menores que aqui, enquanto o tempo de estudo fora de aula é bem maior.

Capítulo 14

Agradecimentos

Gostaria de agradecer primeiramente a minha mãe por sempre me apoiar em minhas decisões esses anos todos, inclusive quando decidi abandonar o mestrado e carreira em Química para me dedicar somente ao BCC. Esse incentivo e ajuda sempre me fizeram dar o melhor de mim mesmo na minha formação pessoal e acadêmica.

Agradeço ao professor Roberto Hirata Jr. por ter dado a sugestão do trabalho e por toda a ajuda e paciência durante esse ano como meu orientador. Agradeço também aos ex-alunos André Shoji Asato e Rafael de Oliveira Lopes Gonçalves pelo trabalho de conclusão de curso em 2009 no qual esse trabalho foi baseado.

Agradeço também aos outros professores do IME e do DCC pela formação que me foi dada dentro da sala de aula e pelas conversas e ajudas fora. Em especial gostaria de agradecer os professores Alfredo, Carlinhos, Coelho, Daniel Batista, Fabio Kon, Hitoshi e Paulo Feofiloff pelas excelentes disciplinas ministradas que me deram a absoluta certeza que o BCC era o lugar certo para mim.

Outro agradecimento especial é para a professora Cris, com quem fiz o maior número de disciplinas durante todo o BCC e é, sem dúvida alguma, a melhor professora que tive em todos meus anos de USP. Agradeço por todo o conhecimento que me foi passado, por fazer eu me esforçar tanto e por todos os conselhos dados.

Por fim agradeço aos meus amigos dentro e fora da USP por me ajudar a manter a sanidade durante esses anos todos.