

**MINISTÉRIO DA DEFESA
EXÉRCITO BRASILEIRO
DEPARTAMENTO DE CIÊNCIA E TECNOLOGIA
INSTITUTO MILITAR DE ENGENHARIA
CURSO DE GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO**

**Cap BRUNO AVELINO DE ARAUJO OLIVEIRA
JAYME BOARIN DE MAGALHÃES ALVIM**

**DESENVOLVIMENTO DE ALGORITMOS DE SEGMENTAÇÃO DE
IMAGENS EM AMBIENTE ANDROID**

**Rio de Janeiro
2017**

INSTITUTO MILITAR DE ENGENHARIA

**Cap BRUNO AVELINO DE ARAUJO OLIVEIRA
JAYME BOARIN DE MAGALHÃES ALVIM**

**DESENVOLVIMENTO DE ALGORITMOS DE
SEGMENTAÇÃO DE IMAGENS EM AMBIENTE ANDROID**

Projeto de Fim de Curso apresentado ao Curso de Graduação em Engenharia de Computação do Instituto Militar de Engenharia, como requisito parcial para a obtenção do título de Engenheiro de Computação.

Orientadora: Prof^a. Carla Liberal Pagliari - Ph.D.
Co-Orientador: Prof. Marcelo de Mello Perez - Ph.D.

Rio de Janeiro
2017

c2017

INSTITUTO MILITAR DE ENGENHARIA

Praça General Tibúrcio, 80 – Praia Vermelha

Rio de Janeiro – RJ CEP: 22290-270

Este exemplar é de propriedade do Instituto Militar de Engenharia, que poderá incluí-lo em base de dados, armazenar em computador, microfilmar ou adotar qualquer forma de arquivamento.

É permitida a menção, reprodução parcial ou integral e a transmissão entre bibliotecas deste trabalho, sem modificação de seu texto, em qualquer meio que esteja ou venha a ser fixado, para pesquisa acadêmica, comentários e citações, desde que sem finalidade comercial e que seja feita a referência bibliográfica completa.

Os conceitos expressos neste trabalho são de responsabilidade do(s) autor(es) e do(s) orientador(es).

621.3678	Oliveira, Bruno Avelino de Araujo
O48d	Desenvolvimento de algoritmos de segmentação de imagens em ambiente android / Bruno Avelino de Araújo Oliveira; Jayme Boarin de Magalhães Alvim; orientados por Carla Liberal Pagliari; Marcelo de Mello Perez – Rio de Janeiro: Instituto Militar de Engenharia, 2017.
	63p. : il.
	Projeto de Fim de Curso (PROFIC) – Instituto Militar de Engenharia, Rio de Janeiro, 2017.
1. Curso de Engenharia de Computação – Projeto de Fim de Curso. 2. Segmentação de imagens. I. Alvim, Jayme Boarin de Magalhães. II. Pagliari, Carla Liberal. III. Perez, Marcelo de Melo. IV. Título. V. Instituto Militar de Engenharia.	

INSTITUTO MILITAR DE ENGENHARIA

Cap BRUNO AVELINO DE ARAUJO OLIVEIRA
JAYME BOARIN DE MAGALHÃES ALVIM

DESENVOLVIMENTO DE ALGORITMOS DE
SEGMENTAÇÃO DE IMAGENS EM AMBIENTE ANDROID

Projeto de Fim de Curso apresentado ao Curso de Graduação em Engenharia de Computação do Instituto Militar de Engenharia, como requisito parcial para a obtenção do título de Engenheiro de Computação.

Orientadora: Profª. Carla Liberal Pagliari - Ph.D.

Co-Orientador: Prof. Marcelo de Mello Perez - Ph.D.

Aprovado em 18 de Maio de 2017 pela seguinte Banca Examinadora:

Carla L. Pagliari

Profª. Carla Liberal Pagliari - Ph.D. do IME - Presidente

Marcelo de Mello Perez

Prof. Marcelo de Mello Perez - Ph.D. do IME

Janciro

Prof. Anderson Fernandes Pereira dos Santos - D.Sc. do IME

Paulo Roberto Rosa Lopes Nunes

Prof. Paulo Roberto Rosa Lopes Nunes - Ph.D. do IME

Rio de Janeiro
2017

Dedico aos meus familiares, amigos, minha esposa Mayra e professores do Instituto Militar de Engenharia, pois me deram forças e foram fundamentais para que eu pudesse vencer os obstáculos diários. Dedico aos meus pais e familiares pelo amor incondicional e pelos valores e ensinamentos passados que me fazem crescer e evoluir constantemente.

AGRADECIMENTOS

Agradeço aos que estiveram comigo durante essa árdua jornada de estudos e me auxiliaram nessa etapa de desenvolvimento profissional. Um muito obrigado aos familiares, amigos e mestres.

Um obrigado especial aos Professores Orientadores Ph.D. Carla Liberal Pagliari e Ph.D. Marcelo de Mello Perez, por suas disponibilidades e atenções.

“Computação não se relaciona mais a computadores. Relaciona-se a viver.”

NICHOLAS NEGROPONTE

SUMÁRIO

LISTA DE ILUSTRAÇÕES	9
LISTA DE TABELAS	12
1 INTRODUÇÃO	15
1.1 Motivação	15
1.2 Objetivos	16
1.3 Justificativa	16
1.4 Estrutura da Monografia	17
2 SEGMENTAÇÃO DE IMAGENS	18
2.1 Conceito	18
2.2 Segmentação para seres humanos e para computadores	18
3 ALGORITMOS DE SEGMENTAÇÃO	21
3.1 <i>Thresholding</i>	21
3.1.1 <i>Threshold Global</i>	22
3.1.2 <i>Threshold Local</i> (ou dinâmico)	22
3.2 Baseado em Bordas	23
3.3 Baseado em Regiões	23
3.3.1 <i>Split and Merge</i>	24
3.3.1.1 Procedimento	24
3.3.1.2 EXEMPLO - <i>QUADTREE</i>	24
3.3.2 <i>Region growing</i> (abordagem <i>bottom-up</i>)	24
3.3.2.1 Procedimento	24
3.3.2.2 EXEMPLO - <i>WATERSHED</i>	25
4 ALGORITMO <i>WATERSHED</i> ADAPTADO	26
4.1 <i>Watershed</i>	26
4.2 <i>Watershed Algorithm Based On Connected Components</i>	27
4.3 Implementação do Algoritmo	28
4.3.1 Pré-Processamento	29
4.3.1.1 Conversão para Cinza	30
4.3.1.2 Filtro Bilateral	30

4.3.1.3 Operador Sobel	31
4.3.1.4 Operadores Morfológicos	32
4.3.2 Segmentação por Técnica Baseada em <i>Watershed</i>	33
4.3.2.1 <i>Step 1</i>	33
4.3.2.2 <i>Step 2</i>	34
4.3.2.3 <i>Step 3</i>	34
4.3.3 Pós-Processamento	36
5 AMBIENTE DE DESENVOLVIMENTO	37
5.1 Android	37
5.2 NDK	38
5.3 JNI	38
5.4 Aplicação do JNI	38
5.5 OpenCV Java e OpenCV Nativo	39
5.6 Dispositivos de Testes	39
5.6.1 Firebase	43
5.6.1.1 Layout	43
6 APLICAÇÃO	44
6.1 Aquisição da imagem	44
6.1.1 Galeria de imagens	44
6.1.2 Câmera	47
6.2 Segmentação da imagem	49
6.2.1 <i>Watershed</i> Desenvolvido	49
7 CONCLUSÃO	56
8 CRONOGRAMA	57
8.1 Definição de Etapas	57
8.1.1 Escolha do Tema e Estudo de Viabilidade	57
8.1.2 Revisão Bibliográfica	57
8.1.3 Elaboração da Monografia	57
8.1.4 Estudo e Análise dos Algoritmos de Segmentação de Imagem	57
8.1.5 Implementação	57
8.1.6 Teste	58
8.1.7 Entrega do Relatório Final e Apresentação	58

8.2	Entregáveis	58
9	REFERÊNCIAS BIBLIOGRÁFICAS	59
10	ANEXOS	61
10.1	ANEXO 1: Ferramentas	62
10.2	Android Studio	62
10.3	OpenCV	62
10.4	Android SDK	62
10.5	Android NDK	63

LISTA DE ILUSTRAÇÕES

FIG.1.1	Tela do Aplicativo Android (imagem original e imagem segmentada).	15
FIG.1.2	Diagrama em Blocos do Projeto.	16
FIG.2.1	Imagen original. (ARBELAEZ et al., 2011)	19
FIG.2.2	Imagens segmentadas por seres humanos, gerando 8 e 16 segmentos nas duas primeiras imagens e nas duas últimas, respectivamente. (ARBELAEZ et al., 2011)	19
FIG.2.3	Imagens segmentadas por seres humanos, gerando 22 e 26 segmentos nas duas primeiras imagens e nas duas últimas, respectivamente. (ARBELAEZ et al., 2011)	19
FIG.2.4	Imagen original. (DATASET; BENCHMARK, 2017)	20
FIG.2.5	Resultados de 4 algoritmo de detecção de bordas. (DATASET; BENCHMARK, 2017)	20
FIG.3.1	Imagen original. (STANFORD, 2017)	22
FIG.3.2	Imagen segmentada pelo algoritmo <i>threshold</i> global.	22
FIG.3.3	Imagen segmentada pelo método de segmentação <i>thresholding</i> local.	23
FIG.3.4	Imagen original de um macaco à esquerda, e à direita, o resultado de sua segmentação pelo método de detecção de bordas. (STANFORD, 2017)	23
FIG.3.5	Uma representação visual de uma região à esquerda, e à direita, o resultado de sua segmentação pelo Algoritmo <i>Quadtree</i> . (STANFORD, 2017)	25
FIG.3.6	Imagen original de moedas à esquerda, e à direita, o resultado de sua segmentação pelo Algoritmo <i>Watersheed</i> . (STANFORD, 2017)	25
FIG.4.1	Abordagem " <i>flooding based watershed</i> ".Gomes (2017)	26
FIG.4.2	Abordagem " <i>rainfalling based watershed</i> ". Ruparelia (2012)	27
FIG.4.3	Ilustração do funcionamento da técnica <i>Watershed Based On Connected Components</i> . Ruparelia (2012)	28
FIG.4.4	Diagrama de blocos do algoritmo de segmentação dessa pesquisa.	28
FIG.4.5	Diagrama de blocos da etapa de pré-processamento.	29

FIG.4.6	Definição matemática do Filtro Bilateral. (BANTERLE et al., 2012)	30
FIG.4.7	Definição matemática do Operador Sobel.	31
FIG.4.8	Resultado prática da aplicação do operador Sobel.	31
FIG.4.9	Resultado prática da aplicação do operador morfológico.	33
FIG.5.1	Pilha de software Android.	37
FIG.5.2	Motorola Moto G3 - Processador Quad Core de 1.4GHz, 2 GB de RAM.	40
FIG.5.3	Samsung Galaxy S7 - Processador Octa Core de 2.3GHz, 4GB de RAM.	40
FIG.5.4	Samsung Google Nexus 10 - Processador Dual Core de 1.7GHz, 2 GB de RAM.	40
FIG.5.5	Seleção do dispositivo virtual no Android Studio.	41
FIG.5.6	Nexus 5 API 24.	42
FIG.5.7	Opções de configuração de hardware do dispositivo virtual.	42
FIG.6.1	Tela Inicial do aplicativo.	44
FIG.6.2	Sequência de atividades para aquisição da imagem por meio da galeria de imagens.	45
FIG.6.3	Sequência de atividades para aquisição da imagem por meio da galeria de imagens.	46
FIG.6.4	Sequência de atividades para aquisição e tratamento das imagens por meio da câmera do dispositivo.	47
FIG.6.5	Sequência de atividades para aquisição e tratamento das imagens por meio da câmera do dispositivo.	48
FIG.6.6	Sequência de atividades para aquisição e tratamento das imagens por meio da câmera do dispositivo.	48
FIG.6.7	Sequência de atividades para aquisição e tratamento das imagens por meio da câmera do dispositivo.	48
FIG.6.8	Resultados de segmentação utilizando o algoritmo <i>watershed</i> desenvolvido.	49
FIG.6.9	Resultados de segmentação utilizando o algoritmo <i>watershed</i> desenvolvido.	50
FIG.6.10	Resultados de segmentação utilizando o algoritmo <i>watershed</i> desenvolvido.	51

FIG.6.11 Resultados de segmentação utilizando o algoritmo <i>watershed</i> desen- volvido.	52
FIG.6.12 Resultados de segmentação utilizando o algoritmo <i>watershed</i> desen- volvido.	53
FIG.6.13 Resultados de segmentação utilizando o algoritmo <i>watershed</i> desen- volvido.	54
FIG.6.14 Resultados de segmentação utilizando o algoritmo <i>watershed</i> desen- volvido.	55

LISTA DE TABELAS

TAB.8.1 Cronograma das atividades previstas	58
---	----

RESUMO

Segmentação é uma importante etapa do processamento digital de imagens. Com um número cada vez maior de aplicações relacionadas ao domínio de visão computacional, mais presente se faz a necessidade de se aprofundar na identificação e análise de regiões relevantes de uma imagem a fim de obter melhores resultados e conclusões por meio da segmentação.

Embora a identificação de imagens seja algo natural e uma atividade consideravelmente fácil para os seres humanos, a mesma tarefa em sistemas computacionais automatizados torna-se significativamente mais complexa. Com a crescente tendência de utilização de máquinas e sistemas para a realização das atividades na sociedade, nota-se a importância de desenvolver e implementar novas ferramentas capazes de trazer uma nova abordagem ao problema. Considerado como um problema mal colocado, isto é, para o qual não há uma solução universal, o estudo das principais técnicas consolidadas e o desenvolvimento de novas é a estratégia que se procura utilizar em projetos de pesquisa nesse domínio.

Esse projeto introduz o problema da segmentação, abordando sua motivação, suas dificuldades e sua aplicação, apresenta o estudo das técnicas mais relevantes e, como objetivo principal, desenvolve um aplicativo móvel em ambiente Android para a segmentação de imagens.

ABSTRACT

Image segmentation is an important stage of the digital processing of images. With an increasing number of applications related to the field of computer vision, the more present the need to deepen the identification and analysis of relevant regions of an image in order to obtain better results and conclusions through segmentation.

Although the identification of images is something natural and a considerably easy activity for humans, the same task in automated computer systems becomes significantly more complex. With the growing tendency to use machines and systems to carry out activities in society, it is important to develop and implement new tools capable of bringing a new approach to the problem. Considered as an "ill-posed" problem, that is, for which there is no universal solution, the study of the main consolidated techniques and the development of new ones is the strategy that is sought to be used in research projects in this field.

This project introduces the problem of segmentation, approaching its motivation, difficulties and its application, presents the study of the most relevant techniques and, as main objective, develops an Android mobile application for the segmentation of images.

1 INTRODUÇÃO

1.1 MOTIVAÇÃO

Cada vez mais as máquinas estão sendo utilizadas em atividades de nossa sociedade. Atuando na substituição de profissionais ou no auxílio aos mesmos, elas estão presentes e participando do cotidiano ativamente. A implementação de sistemas de visão computacional tem se tornado, portanto, uma necessidade mais forte à medida que as aplicações que envolvem o tratamento de imagens se desenvolvem e buscam se aproximar da visão e da análise humana. A segmentação é uma técnica utilizada em diversas atividades que envolvem o processamento digital de imagens. Diversas são as aplicações que fazem uso da identificação e análise de uma imagem, necessitando do estudo de regiões específicas das mesmas a fim de alcançar resultados e conclusões de forma eficiente. Por se tratar de um problema sem solução universal e de vasta aplicação, existem inúmeras possibilidades a serem exploradas e muito se tem estudado sobre essa área, com a evolução de novas técnicas e algoritmos que trazem uma nova abordagem ao problema. Esse projeto visa estudar e implementar técnicas de segmentação de imagens no ambiente Android, possibilitando o aprendizado de diferentes métodos da área de processamento de imagens, incluindo a área de visão computacional, bem como de desenvolvimento de aplicativos no ambiente Android. A Figura 1.1 exibe o resultado do aplicativo que este projeto de final de curso está desenvolvendo.

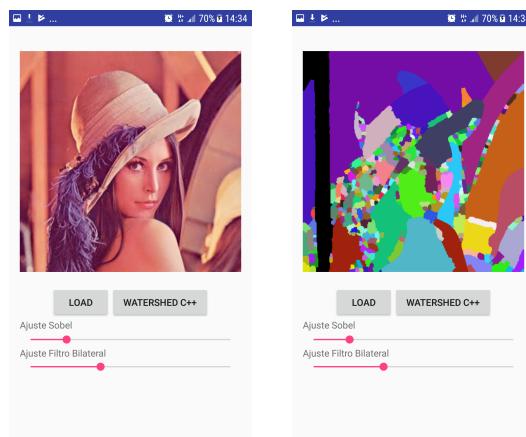


FIG. 1.1: Tela do Aplicativo Android (imagem original e imagem segmentada).

1.2 OBJETIVOS

O objetivo dessa pesquisa é o desenvolvimento de uma aplicação móvel, no ambiente Android, capaz de segmentar imagens por meio de diferentes algoritmos baseados em técnicas de segmentação e procedimentos de pré-processamento na área de Visão Computacional. Para estudar a eficiência dos resultados obtidos, será realizada uma análise comparativa com relação aos algoritmos fornecidos pela biblioteca OpenCV, ferramenta de apoio a essa pesquisa descrita na seção 10.3 do anexo 1. A realização de todas as etapas de desenvolvimento até a concepção do produto final, disponível na plataforma do GitHub, no *link* <https://github.com/brunoavelino/SEG>, para livre utilização, permitirá aos alunos maior conhecimento nesse assunto que é um dos domínios mais promissores da tecnologia. A Figura 1.2 ilustra os passos do projeto, onde o aplicativo poderá adquirir imagens diretamente da câmera do dispositivo ou da galeria de imagens do dispositivo. O bloco denominado Pré-Processamento ilustra uma possível etapa de procedimentos a serem realizados sobre as imagens oriundas da câmera ou do banco de imagens, com a finalidade de tratar as imagens para os algoritmos de segmentação. O bloco seguinte, Técnicas de Segmentação implementa um ou mais métodos de segmentação, descritos no capítulo 2. Finalmente, o bloco Exibição vai exibir a imagem segmentada na tela do dispositivo.

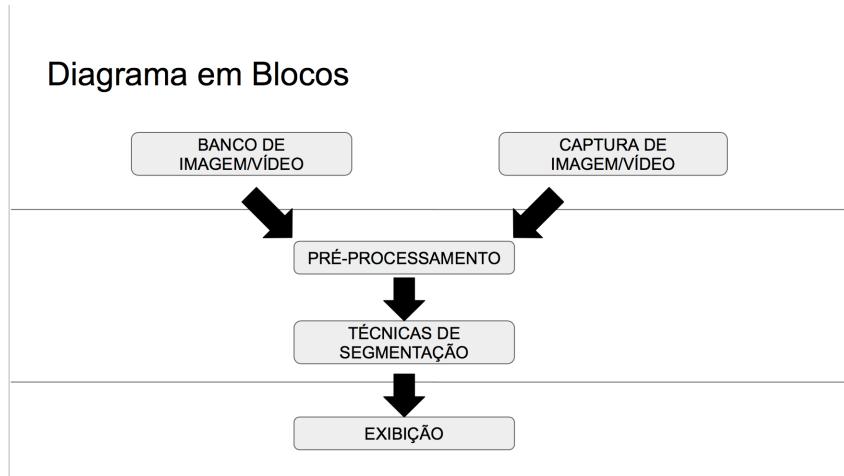


FIG. 1.2: Diagrama em Blocos do Projeto.

1.3 JUSTIFICATIVA

Este trabalho pode ser justificado com base nos seguintes pontos:

- Inexistência de uma solução única para o problema em questão. Trata-se de um problema ainda em aberto com muitas oportunidades a serem exploradas;
- Diversas aplicações necessitam da segmentação de imagens como parte importante das suas atividades, além de outras que podem ser aprimoradas e desenvolvidas por meio de sua utilização; e
- Visão computacional como domínio promissor da tecnologia, com uma atuação cada vez maior no mercado e interação com outros domínios como robótica e inteligência artificial.

1.4 ESTRUTURA DA MONOGRAFIA

A seguir é apresentado o capítulo 2, onde são introduzidos os problemas inerentes à segmentação. Na sequência, o capítulo 3 descreve três principais técnicas de segmentação. No capítulo 4 é apresentado o algoritmo desenvolvido para essa aplicação, adaptado do algoritmo original *watershed*, o qual é introduzido neste mesmo capítulo, além de sua implementação com auxílio da biblioteca OpenCV. O capítulo 5 descreve o ambiente de desenvolvimento utilizado para a construção dessa aplicação, entrando em detalhes sobre o processo de portabilidade do algoritmo implementado em C++ para o ambiente Android, com auxílio da ferramenta JNI. O capítulo 6 é responsável por ilustrar o funcionamento do aplicativo desenvolvido e, finalmente, o capítulo 7 sintetiza a pesquisa, abordando os impactos desse projeto.

2 SEGMENTAÇÃO DE IMAGENS

A segmentação é um problema do tipo "*ill-posed*" ("mal definido"), uma vez que não existe uma solução única, universal. Por isso, a segmentação é considerada o problema mais difícil de ser resolvido em análise de imagens (STRAND, 2017).

2.1 CONCEITO

A segmentação de imagens tem como uma de suas interpretações como a divisão em regiões ou categorias consideradas "relevantes", que correspondem a objetos ou partes de objetos. Decidir o que é relevante em uma imagem depende do problema a ser resolvido, em que os objetos segmentados devem corresponder às áreas de interesse da aplicação. Dentre essas aplicações, podemos citar os seguintes exemplos:

- Aplicações militares: Reconhecimento de alvos terrestres, aéreos e navais;
- Análise de imagens médicas: Identificação de doenças como tumores;
- Veículos autônomos; e
- Robótica.

2.2 SEGMENTAÇÃO PARA SERES HUMANOS E PARA COMPUTADORES

Para seres humanos, a identificação de regiões similares ou objetos diferentes presentes em uma imagem é um processo fácil. Seu sistema cognitivo auxiliado por seu sistema visual permite reconhecer e segmentar os objetos de forma instantânea sem a percepção de todo esse processo. Além disso, o uso da segmentação por distância como técnica auxiliar é outro fator que contribui para esse processo, uma vez que sua visão estereoscópica lhes fornece informação de profundidade. No caso de computadores, essa tarefa se torna mais complexa, pois envolve a análise de características de cada pixel ou da distribuição da população de pixels. Para isso, deve-se implementar algoritmos de segmentação, os quais serão explorados com maior profundidade nas seções subsequentes dessa pesquisa. A figura 2.1 exibe uma imagem que foi manualmente segmentada por 4 seres humanos. É possível notar que cada pessoa não identificou como "relevantes" as mesmas áreas, conforme ilustrado pela figura 2.2.



FIG. 2.1: Imagem original. (ARBELAEZ et al., 2011)

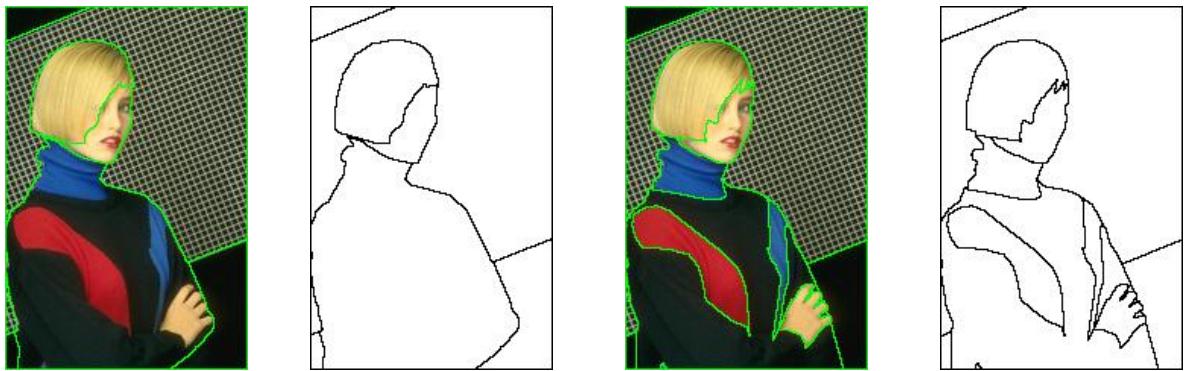


FIG. 2.2: Imagens segmentadas por seres humanos, gerando 8 e 16 segmentos nas duas primeiras imagens e nas duas últimas, respectivamente. (ARBELAEZ et al., 2011)

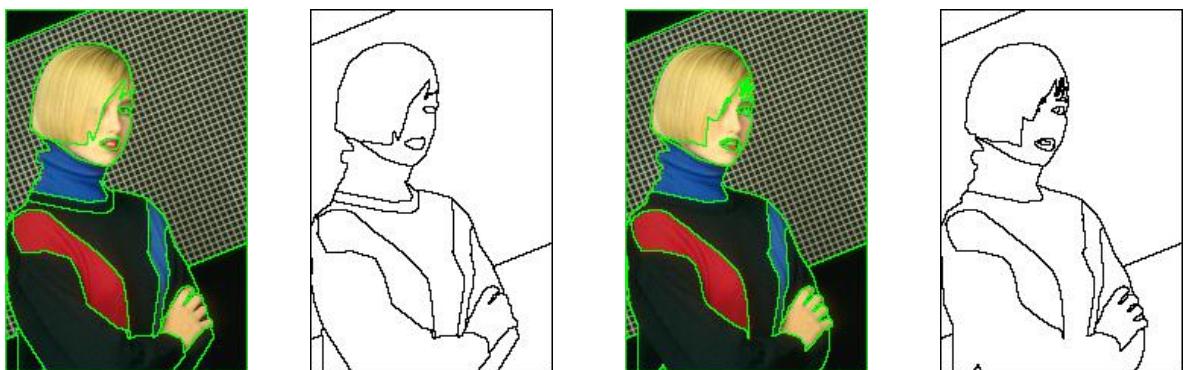


FIG. 2.3: Imagens segmentadas por seres humanos, gerando 22 e 26 segmentos nas duas primeiras imagens e nas duas últimas, respectivamente. (ARBELAEZ et al., 2011)

Ainda que o processo de segmentação para humanos seja fácil e automático, é comum e natural que diferentes pessoas identifiquem objetos ou partes de objetos distintos em uma

dada imagem. Isso se deve à percepção de relevância atribuída a cada região variar com a interpretação pessoal de cada um. O mesmo problema ocorre de forma mais acentuada com relação aos diferentes algoritmos. Cada algoritmo tem a sua própria abordagem para tratar do mesmo problema e, de acordo com sua implementação, leva a diferentes resultados, que podem ser analisados comparativamente. É importante dizer que o mesmo algoritmo pode ser mais ou menos eficiente de acordo com a imagem utilizada como dado de entrada, possibilitando diversos estudos na área como o tema desta pesquisa. A figura 2.4 ilustra este problema(DATASET; BENCHMARK, 2017) .



FIG. 2.4: Imagem original. (DATASET; BENCHMARK, 2017)

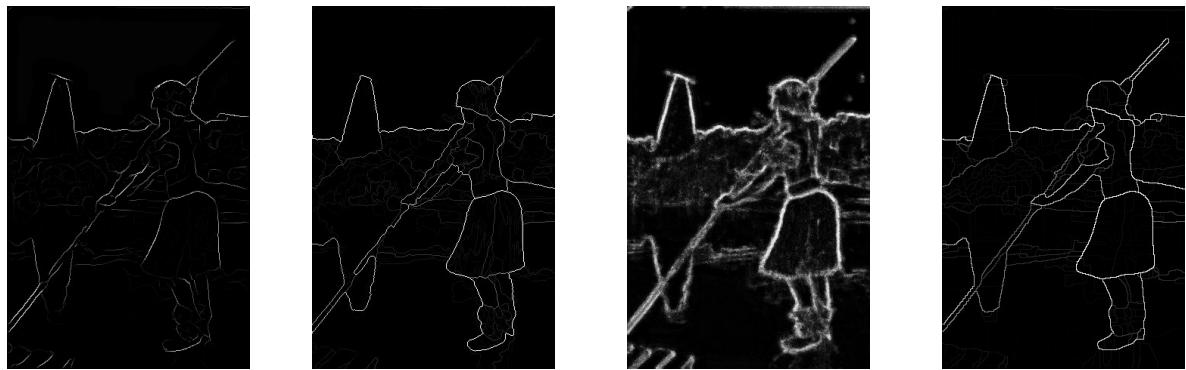


FIG. 2.5: Resultados de 4 algoritmo de detecção de bordas. (DATASET; BENCHMARK, 2017)

Devido a grande variedade de algoritmos existentes, essa pesquisa se limitará à explicação dos principais e mais utilizados, os quais fornecem um bom entendimento das técnicas utilizadas e servem como base para o desenvolvimento de novas técnicas.

3 ALGORITMOS DE SEGMENTAÇÃO

Este capítulo descreve quatro técnicas de segmentação de imagens, sendo a por limiar (*thresholding*) a mais simples. As técnicas baseadas em bordas e regiões são mais complexas e demandam um ônus computacional mais elevado.

Na aplicação desenvolvida nesse projeto, prioriza-se a técnica de segmentação por *watershed*, a qual será mais aprofundada no capítulo 4. Porém, o estudo das principais técnicas, que serão apresentadas a seguir, se torna relevante para a construção da base de conhecimento que fundamenta essa pesquisa.

- *Thresholding*;
- Método Baseado em Bordas; e
- Método Baseado em Regiões
 - *Split and Merge*; e
 - *Watershed*.

3.1 THRESHOLDING

É um método simples de segmentação de imagens. Busca dividir a imagem em duas categorias: objetos (*foreground*) e plano de fundo (*background*). Cada *pixel* é alocado a uma categoria de acordo com seu valor em níveis de cinza.

Dado um limiar T (*threshold*), o *pixel* com valor f_{ij} e localizado na posição (i,j) é alocado à:

$$f(i,j) = \begin{cases} categoria1, & \text{se } f_{ij} \leq T; \\ categoria2, & \text{caso contrário.} \end{cases}$$

O limiar T pode ser escolhido manualmente, testando-se diferentes valores de T e analisando qual deles é mais eficiente na identificação dos objetos de interesse. O *threshold* T também pode ser escolhido a partir do histograma da imagem, e escolhe-se T como o valor que maximiza a separação entre duas distribuições de níveis de cinza.

Na figura 3.1, há uma imagem original do Einstein que será segmentada de duas formas distintas pela técnica de segmentação de imagens por limiar (*thresholding*). Enquanto na figura 3.2 há um limiar T fixo , na figura 3.3 ele é variável.

3.1.1 *THRESHOLD GLOBAL*

O mesmo valor de T é usado para a imagem inteira.

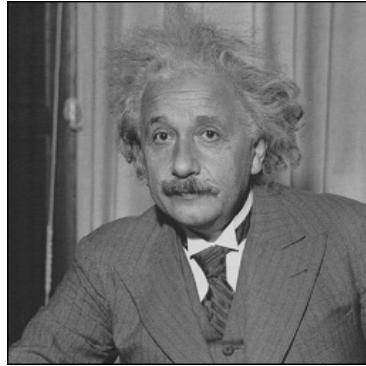


FIG. 3.1: Imagem original. (STANFORD, 2017)



FIG. 3.2: Imagem segmentada pelo algoritmo *threshold* global.

3.1.2 *THRESHOLD LOCAL (OU DINÂMICO)*

Divide-se a imagem em regiões distintas e adota-se um valor T para cada uma delas, onde esse valor funcionará como *threshold* local.

Pode-se observar pela figura 3.3 que houve a separação em mais regiões nesse método do que na figura 3.2, já que foi usado um limiar mais apropriado a cada região.



FIG. 3.3: Imagem segmentada pelo método de segmentação *thresholding* local.

3.2 BASEADO EM BORDAS

Primeiramente, classificam-se os *pixels* como “borda” ou “não-borda”. Depois, divide-se a imagem em regiões, baseado nas bordas detectadas.

As bordas são identificadas por meio das descontinuidades, isto é, variações abruptas nos valores dos *pixels*.

Como pode-se perceber na figura 3.4, por esse método houve a segmentação da imagem de um macaco em regiões de olhos, nariz e outras partes.

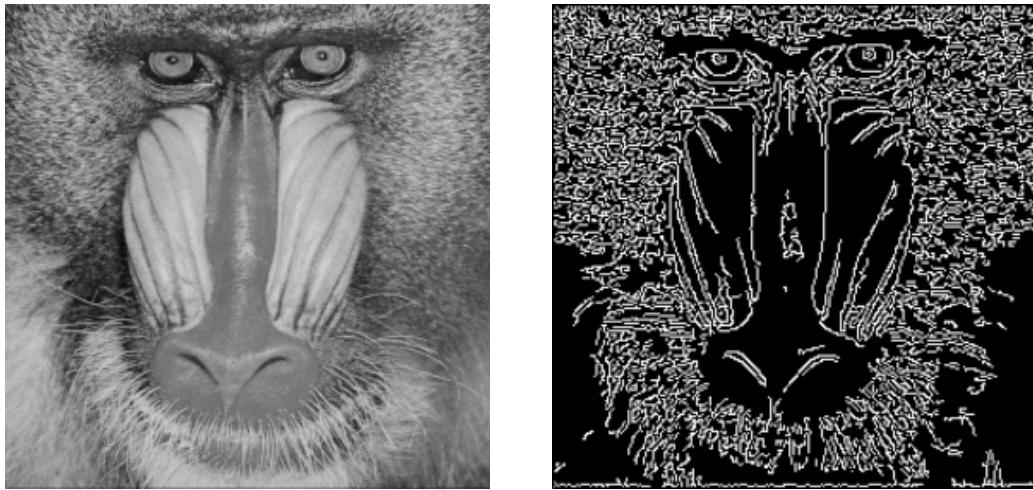


FIG. 3.4: Imagem original de um macaco à esquerda, e à direita, o resultado de sua segmentação pelo método de detecção de bordas. (STANFORD, 2017)

3.3 BASEADO EM REGIÕES

Uma região pode ser “definida” como um grupo de *pixels* conectados com propriedades similares.

Porém, é um conceito importante e difícil de definir, já que depende da interpretação do que seria uma região em determinado caso.

As figuras 2.1, 2.2, 2.3, 2.4 e 2.5 da seção 2.2 do capítulo 2, demonstram as diferentes interpretações do conceito de região pelas diferentes quantidades de regiões notadas por humanos nas figuras 2.2 e 2.3. Tal fato também acontece com os computadores, como nota-se pelas figuras 2.5.

3.3.1 *SPLIT AND MERGE*

3.3.1.1 PROCEDIMENTO

As etapas fundamentais deste algoritmo são:

- a) Criar critério para definir uma área homogênea.
- b) Começar com a imagem completa e divide em 4 sub-imagens.
- c) Checar cada sub-imagem e dividi-la novamente em 4 novas sub-imagens caso ela não seja homogênea.
- d) Repetir item c) até que não se consiga mais subdividir.
- e) Comparar sub-imagens com suas regiões vizinhas e agrupá-las se forem homogêneas.
- f) Repetir item e) até que não se consiga mais agrupar.

3.3.1.2 EXEMPLO - *QUADTREE*

Um exemplo de segmentação baseada em região *Split and Merge* é o Algoritmo *Quadtree*. A figura 3.5 abaixo ilustra a segmentação de imagem por este algoritmo, e observa-se a segmentação da região que contém água na figura.

3.3.2 *REGION GROWING* (ABORDAGEM BOTTOM-UP)

3.3.2.1 PROCEDIMENTO

As etapas fundamentais deste algoritmo são:

- a) Identificar o ponto de partida.
- b) Incluir *pixels* vizinhos com características similares (nível de cinza, textura, cor, etc).
- c) Continuar até que todos os *pixels* estejam associados com um dos pontos de partida.

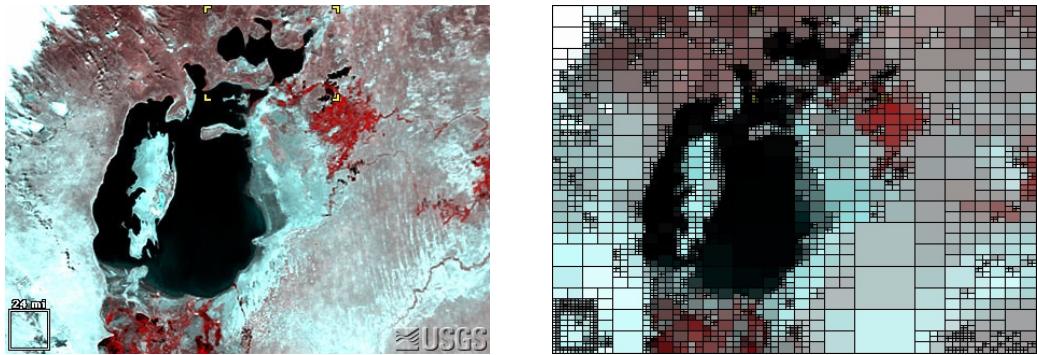


FIG. 3.5: Uma representação visual de uma região à esquerda, e à direita, o resultado de sua segmentação pelo Algoritmo *Quadtree*. (STANFORD, 2017)

3.3.2.2 EXEMPLO - WATERSHED

Um exemplo de segmentação baseada no método *Region growing* é o Algoritmo *Watershed*. A figura 3.6 abaixo ilustra a segmentação de imagem por este algoritmo, e observa-se regiões distintas correspondentes a cada moeda da figura.



FIG. 3.6: Imagem original de moedas à esquerda, e à direita, o resultado de sua segmentação pelo Algoritmo *Watershed*. (STANFORD, 2017)

4 ALGORITMO WATERSHED ADAPTADO

O algoritmo proposto nessa pesquisa é baseado no algoritmo "*Watershed Algorithm Based On Connected Components*", apresentado em Ruparelia (2012), o qual trata-se de uma variação da técnica de *watershed* com resultados de segmentação considerados satisfatórios além da menor complexidade computacional com relação à abordagem tradicional *watershed*.

4.1 WATERSHED

Watershed é uma técnica de segmentação eficiente e poderosa. Tal técnica tem a tendência de gerar sempre resultados com contornos fechados e bem definidos, o que é de grande importância para o processo de segmentação de imagens. Além disso, comparada a outras técnicas de segmentação, apresenta menor complexidade computacional.

Duas abordagens são utilizadas para explicar a ideia básica do *watershed* na segmentação de imagens. A primeira, denominada "*flooding based watershed*", trata a imagem em níveis de cinza com uma paisagem formada por vales, onde encontram-se os mínimos locais. Considerando um processo de inundação com a água subindo a partir de cada um dos vales, serão construídas barragens nos pontos de encontro da água oriunda de dois vales distintos, chamadas de *watersheds*. Essas barragens, portanto, são interpretadas como bordas entre diferentes regiões da imagem.(STRAND, 2017)

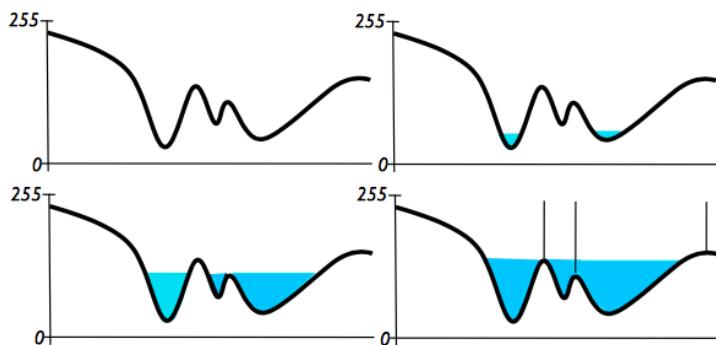


FIG. 4.1: Abordagem "*flooding based watershed*".Gomes (2017)

A outra abordagem, denominada "*rainfalling based watershed*" trata a imagem em níveis de cinza da mesma forma que a primeira, porém o fluxo de água ocorre a partir

de gotas de água que ao incidirem em qualquer ponto da superfície escorrerão para um determinado vale, onde encontra-se um mínimo local. O conjunto de pontos para os quais a gota de água escorre para o mesmo local é interpretada como uma região e os limites entre duas regiões adjacentes, interpretados como bordas, são as *watersheds*. (STRAND, 2017)

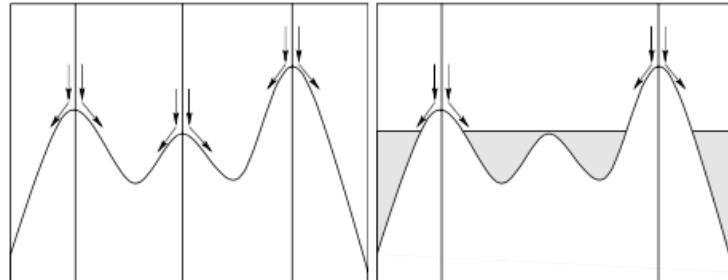


FIG. 4.2: Abordagem "rainfalling based watershed". Ruparelia (2012)

Ambas abordagens tratam da mesma ideia básica da técnica, sendo duas formas diferentes de ilustrar seus funcionamento sobre os quais diferentes algoritmos são propostos.

Um dos principais problemas relacionados à abordagem tradicional do *watershed* é o problema de *over-segmentation*. Para reduzir esse problema, diversas variações de algoritmos baseados na técnica de *watershed* foram implementados, tal qual o algoritmo utilizado nessa pesquisa, explicado detalhadamente na seção 4.3 do capítulo 4.

Assim as diferenças entre o algoritmo proposto e o tradicional, como o implementado na biblioteca OpenCV, apresentada na seção 10.3 do anexo 1, conduzirão a uma análise comparativa entre os resultados obtidos por meio de cada um deles.

4.2 WATERSHED ALGORITHM BASED ON CONNECTED COMPONENTS

A análise comparativa dos algoritmos *watershed* baseado em componentes conectados e *watershed* tradicional permite verificar a maior eficiência do primeiro devido ‘as seguintes características:

- Fila FIFO ao invés de fila hierárquica - algoritmo tradicional - que requer sequências de acesso à memória não uniformes;
- Estrutura de dados mais simples; e
- Menor tempo de execução.

Este algoritmo tem como princípio conectar cada *pixel* (componente), caso este não seja o mínimo local, ao menor *pixel* vizinho. Todos os *pixels* direcionados para o mesmo mínimo local formam um segmento e são, portanto, rotulados da mesma forma.

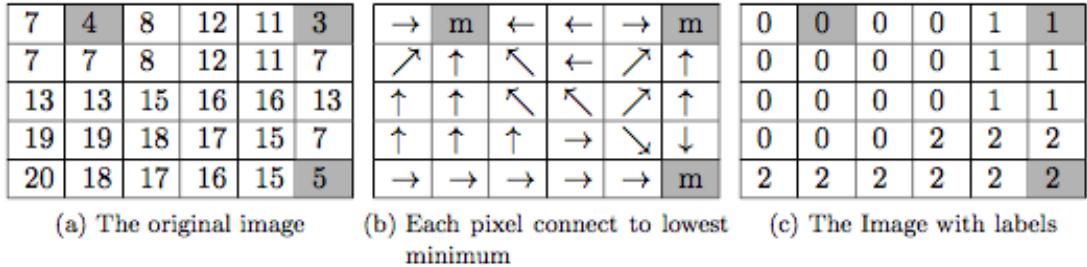


FIG. 4.3: Ilustração do funcionamento da técnica *Watershed Based On Connected Components*. Ruparelia (2012)

4.3 IMPLEMENTAÇÃO DO ALGORITMO

O algoritmo de segmentação proposto nessa pesquisa, assim como os demais algoritmos baseados na técnica de segmentação por *watershed*, segue a estrutura apresentada na figura 4.4. A imagem original passa por uma etapa de pré-processamento, em que diferentes procedimentos são realizados. Após a etapa de pré-processamento, a imagem é segmentada pela técnica *watershed* e seu resultado pode ser processado, na etapa de pós-processamento, a fim de corrigir algumas falhas geradas pelo processo de segmentação para, finalmente, obter a segmentação como resultado final. Todas as etapas serão explicadas nas próximas seções de acordo como foram implementadas no algoritmo proposto por essa pesquisa.

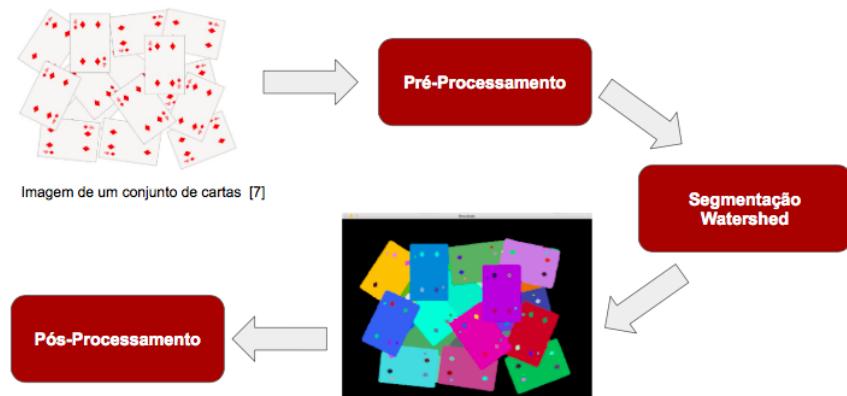


FIG. 4.4: Diagrama de blocos do algoritmo de segmentação dessa pesquisa.

4.3.1 PRÉ-PROCESSAMENTO

A etapa de pré-processamento tem como objetivo preparar a imagem que será utilizada na segmentação efetivamente. Para isso alguns procedimentos são realizados com o objetivo de retirar ruídos e realçar bordas para facilitar o processo de segmentação a fim de obter um resultado mais eficiente. Além disso, assim como a etapa de pós-processamento, esta etapa visa diminuir o problema de *over-segmentation*, o qual tende a ocorrer quando aplica-se a técnica de *watershed*. A ordem em que esses procedimentos ocorrem nesta etapa consta na estrutura apresentada pela figura 4.5.

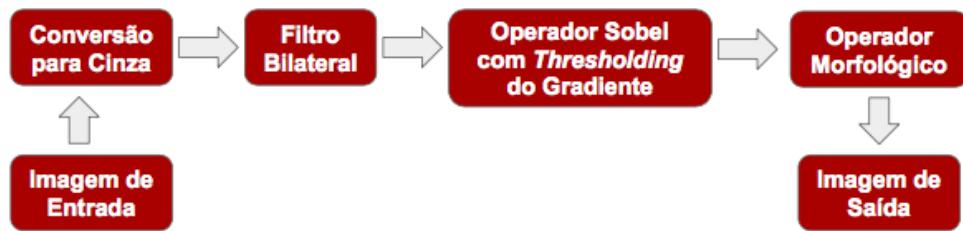


FIG. 4.5: Diagrama de blocos da etapa de pré-processamento.

4.3.1.1 CONVERSÃO PARA CINZA

Um dos procedimentos realizados na etapa de pré-processamento é a conversão da imagem original colorida para a imagem em níveis de cinza. Utiliza-se a quantização em 256 níveis de cinza (8 bits), onde o nível zero representa o preto e o nível 255, o branco. O objetivo dessa conversão é tratar a imagem em um canal já que o escopo da presente pesquisa é tratar imagens em escala de cinza.

4.3.1.2 FILTRO BILATERAL

O filtro biltateral é utilizado como um dos primeiros procedimentos da etapa de pré-processamento a fim de reduzir os ruídos presentes na imagem a ser segmentada, preservando suas bordas.

Abaixo a definição matemática do Filtro Bilateral:

$$I^{\text{filtered}}(x) = \frac{1}{W_p} \sum_{x_i \in \Omega} I(x_i) f_r(\|I(x_i) - I(x)\|) g_s(\|x_i - x\|)$$

$$W_p = \sum_{x_i \in \Omega} f_r(\|I(x_i) - I(x)\|) g_s(\|x_i - x\|)$$

I^{filtered} is the filtered image;

I is the original input image to be filtered;

x are the coordinates of the current pixel to be filtered;

Ω is the window centered in x ;

f_r is the range kernel for smoothing differences in intensities (this function can be a [Gaussian function](#));

g_s is the spatial kernel for smoothing differences in coordinates (this function can be a Gaussian function).

FIG. 4.6: Definição matemática do Filtro Bilateral. (BANTERLE et al., 2012)

4.3.1.3 OPERADOR SOBEL

O operador Sobel é um operador de detecção de bordas que se baseia no operador gradiente. Os gradientes horizontal, G_x , e vertical, G_y , são calculados considerando uma janela de tamanho 3, isto é, $3 \times 3 \text{ pixels}$, e calcula-se uma aproximação do gradiente em cada ponto da seguinte forma: (OPENCV, 2017c)

Abaixo a definição matemática do operador Sobel:

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * I \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * I$$

$$G = \sqrt{G_x^2 + G_y^2} \quad G = |G_x| + |G_y|$$

FIG. 4.7: Definição matemática do Operador Sobel.

Abaixo um exemplo prático da aplicação do operador Sobel:

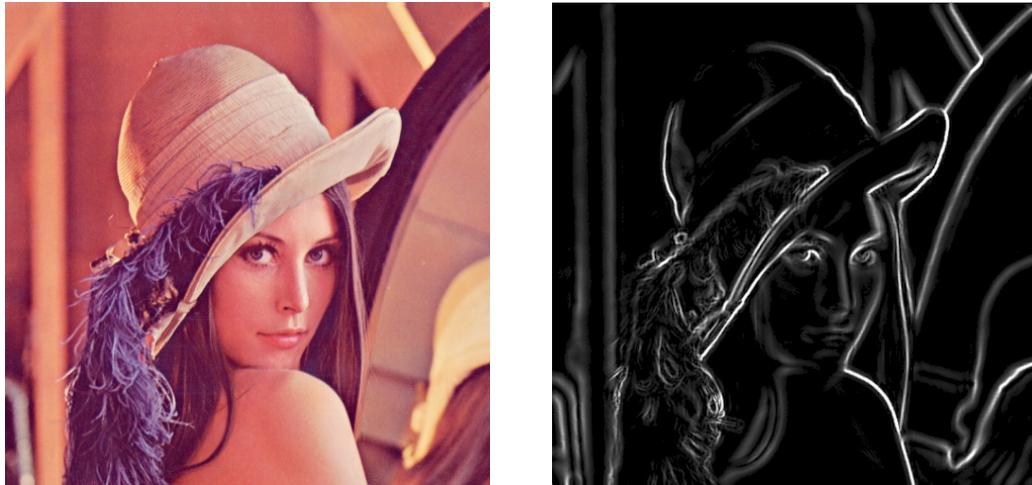


FIG. 4.8: Resultado prática da aplicação do operador Sobel.

4.3.1.4 OPERADORES MORFOLÓGICOS

Após a aplicação de um detector de bordas, como o operador Sobel, a imagem pode ficar com bordas "falhadas". Como a técnica de *watershed* requer que as regiões estejam "fechadas", ou seja, sem falhas nas bordas, é aplicado um conjunto de operações morfológicas para conectar estas regiões fragmentadas. Existem duas operações morfológicas básicas:

- Erosão; e
- Dilatação.

A biblioteca OpenCV fornece 5 transformações morfológicas baseadas nessas duas operações básicas:

- *Opening*;
- *Closing*;
- *Morphological Gradient*;
- *Top Hat*; e
- *Black Hat*.

No algoritmo utiliza-se apenas a transformação do gradiente morfológico - *Morphological Gradient* -, a qual é a diferença entre a dilatação e a erosão de uma imagem. Essa transformação é útil para encontrar o contorno de um objeto em uma dada imagem. (OPENCV, 2017a)

$$dst = morph_{grad}(src, element) = dilate(src, element) - erode(src, element)$$

Abaixo um exemplo prático da aplicação do operador morfológico:

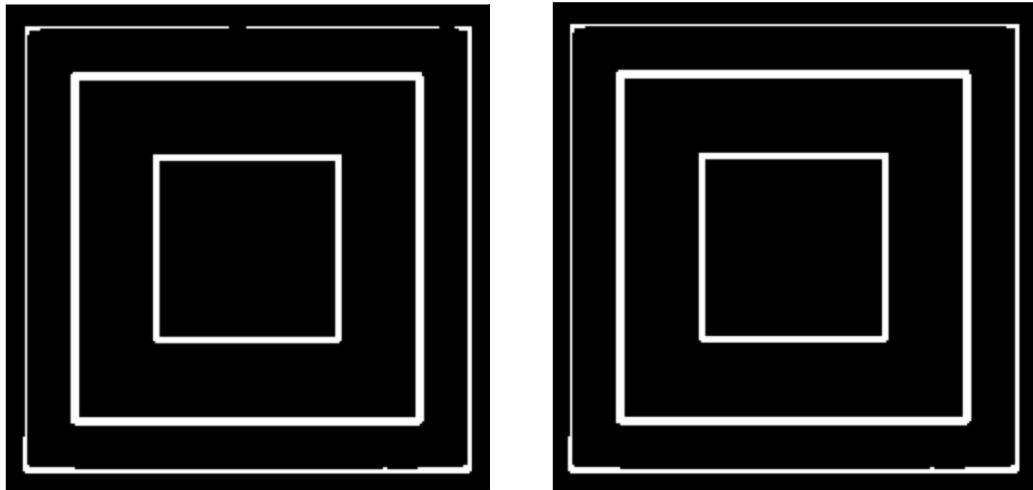


FIG. 4.9: Resultado prática da aplicação do operador morfológico.

4.3.2 SEGMENTAÇÃO POR TÉCNICA BASEADA EM *WATERSHED*

Dentre as duas principais abordagens da técnica de *watershed* mencionadas na seção 4.1 do capítulo 4, optou-se pela abordagem "*rainfalling based watershed*", ilustrado na figura 4.2, a fim de servir como base para a implementação da etapa de segmentação implementada neste algoritmo. Essa etapa é composta por três passos (*steps*).

4.3.2.1 *STEP 1*

O objetivo do passo 1 é encontrar os mínimos locais na imagem. Inicialmente, o *array* $v[p]$ e percorre-se a imagem de cima à esquerda até abaixo à direita e, $v[p]$ assumirá o valor zero se o valor de seu vizinho for menor ou igual ao seu, e o valor 1 caso contrário.

```

STEP1 ( $p$ );
  if  $v[p] \neq 1$  then
    for cada  $n$  vizinho de  $p$  do
      if  $f[n] < f(p)$  then
         $v[p] = 1$ ;
      end
    end
  end

```

Algorithm 1: Algoritmo para o *step 1* da segmentação *watershed*. Ruparelia (2012)

4.3.2.2 *STEP 2*

O fundamento do passo 2 é que se um *pixel* está no *plateau* e seu vizinho apontado para um dos mínimos locais, então o *pixel* aponta para o respectivo vizinho. Para isso, considera-se os *pixels* com $v[p]$ diferente de 1 e com seus *pixels* vizinhos no mesmo *plateau* com $v[p]=1$, ou seja, regiões que não são de mínimo local. Em seguida, calcula-se a menor distância até um mínimo local.

```

STEP2 ( $p$ );
  if  $v[p] \neq 1$  then
     $min = VMAX$ , para cada  $n$  de  $p$ 
    if  $f(n) = f(p)$  and  $v[n] > 0$  and  $v[n] < min$  then
       $min = v[n]$ ;
    end
    if  $min \neq VMAX$  and  $v[p] \neq (min+1)$  then
       $v[p] = min+1$ ;
    end
  end

```

Algorithm 2: Algoritmo para o *step 2* da segmentação *watershed*. Ruparelia (2012)

4.3.2.3 *STEP 3*

O objetivo da terceira etapa do algoritmo é separar os *pixels* em regiões. Para isso, inicializa-se todos os *pixels* com valor zero. Inicialmente, começa-se a definir as regiões a partir dos mínimos locais cujo $v[p]=0$ cujos *pixels* vizinhos com o mesmo valor na escala de cinza ainda não estão associados a uma região definida. Essas regiões são propagadas para seus *pixels* vizinhos de acordo com o valor de $v[p]$ para criar regiões cujo centro é um mínimo local. Em seguida, regiões similares são criadas para todos os mínimos locais

por este mesmo procedimento.

```

STEP3 ( $p$ );
lmin=LMAX, fmin=f( $p$ )
if  $v[p] = 0$  then
  for cada  $n$  vizinho de  $p$  do
    if  $f(n) = f(p)$  and  $l[n] > 0$  and  $l[n] < lmin$  then
      | lmin = l[n]
    end
    if  $lmin = LMAX$  and  $l[p] = 0$  then
      | lmin = New_label+1
    end
  end
else if  $v[p] = 1$  then
  for cada  $n$  vizinho de  $p$  do
    if  $f(n) < fmin$  then
      | fmin = f[n]
    end
  end
  for cada  $n$  vizinho de  $p$  do
    if  $f(n) = fmin$  and  $l[n] > 0$  and  $l[n] < lmin$  then
      | lmin = l[n]
    end
  end
else
  for cada  $n$  vizinho de  $p$  do
    if  $f(n) = f(p)$  and  $v[n] = v[p]-1$  and  $l[n] > 0$  and  $l[n] < lmin$  then
      | lmin = l[n]
    end
  end
end
if  $lmin \neq LMAX$  and  $l[p] \neq LMIN$  then
  | l[p] = lmin
end

```

Algorithm 3: Pseudo código para o *step 3* da segmentação *watershed*.Ruparelia (2012)

4.3.3 PÓS-PROCESSAMENTO

Após as etapas de pré-processamento e segmentação, ainda podem ser necessárias algumas correções relacionadas ao problema de *over-segmentation*. É comum obter diferentes segmentos que fazem parte de uma mesma região como resultado do processo de segmentação. Para tanto pode-se utilizar algum método de agrupamento para mesclar esses segmentos a fim de melhorar a qualidade da segmentação.

A aplicação desenvolvida para esse projeto não contempla a etapa de pós-processamento, melhoria esperada para as futuras versões do aplicativo.

5 AMBIENTE DE DESENVOLVIMENTO

5.1 ANDROID

A plataforma Android foi projetada pela Open Handset Alliance (OHA), um consórcio formado por diversas empresas de tecnologia, incluindo a Google, com o objetivo de fornecer uma estrutura completa e avançada para o desenvolvimento de aplicações para dispositivos móveis. É consituído por um sistema operacional e kits de desenvolvimento (SKD e NDK), apresentados no anexo 1.

O Android é uma pilha de software com base em Linux de código aberto criada para diversos dispositivos e fatores de forma. A figura 5.1 mostra os componentes da plataforma Android. Android (2017a)

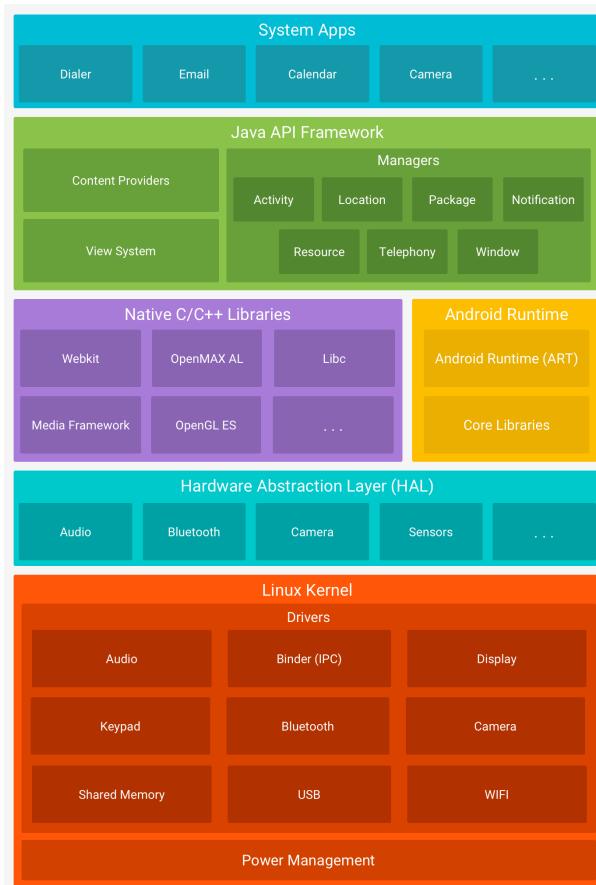


FIG. 5.1: Pilha de software Android.

5.2 NDK

O NDK (*Native Development Kit*) é um conjunto de ferramentas que permite a implantação de aplicações em Android utilizando-se código nativo, como C/C++. A integração do código nativo com o código Java é realizada por chamadas do JNI (*Java Native Interface*).

5.3 JNI

JNI é uma interface de programação nativa, parte do JDK (*Java Development Kit*) e não tem nenhuma relação direta com o Android. JNI permite que o código escrito em Java e executado em uma máquina virtual Java (JVM) execute código e bibliotecas escritos em linguagem nativa como C e C++. O contrário também é permitido, ou seja, incorporar a JVM em aplicações nativas, permitindo chamar o código Java dentro do código nativo.

Como a execução desse projeto trata da portabilidade do algoritmo, apresentado no capítulo 4, escrito em C++ para o ambiente Android, apenas a primeira abordagem é interessante, isto é, chamada do código nativo pelo código Java.

O código nativo é assim chamado, pois é o código escrito em linguagens do próprio sistema operacional, ou seja nativo ao sistema. Isso traz algumas vantagens, como a maior performance da aplicação. Além disso, no caso da aplicação Android, o fato da linguagem Java ser interpretada em tempo de execução pela JVM e o código nativo ser inteiramente compilado, o ganho em performance é ainda maior.

Além disso, a portabilidade para outras plataformas é um fator favorável à utilização de código nativo, uma vez que essas também costumam oferecer suporte para aplicações escritas em linguagens nativas. Assim, para realizar a portabilidade do código basta, em resumo, alterar as APIs nativas. Essa possibilidade faz com que aplicações rodem em ambientes Android e IOS com grande parte do mesmo código - o código nativo -. Isso vale também para aplicações em plataformas Desktop ou web.

5.4 APLICAÇÃO DO JNI

Para o código em Java executar um código nativo por meio do JNI, é necessário o uso de métodos nativos. Esses métodos nativos são declarados em classes Java com auxílio da palavra chave *native* e implementados em código nativo. O JNI também fornece o cabeçalho *jni.h*, que, por sua vez, fornece os métodos de acesso ao ambiente Java (*JavaVM**, *JNIEnv**), a fim de permitir a criação, manipulação e o acesso às primitivas do Java, como *jint* e *jlong*, objetos, como *jobject* e *jclass*, e exceções, como *jthrowable*. No

desenvolvimento desta aplicação o arquivo *GalleryActivity.java*, em Java, é responsável por fazer a chamada do arquivo *native-lib.cpp*, escrito em C++ e onde o algoritmo de segmentação por técnica baseada em *watershed* está implementado, por meio do JNI. Tal chamada ocorre por meio da utilização do método nativo *watershed*, declarado no código Java e implementado no código nativo.

Como a comunicação entre o código escrito em Java e o código nativo é realizada pelo uso de ponteiros, na chamada do método *watershed* são passados dois endereços de objetos da classe *Mat* que representam as matrizes de pixels da imagem a ser segmentada e da imagem resultante da segmentação.

Essas matrizes de pixels são manipuladas como objetos *Mat* no código nativo com auxílio das funcionalidades fornecidas pela biblioteca OpenCV e convertidas em objeto *Bitmap* em Java para ser apresentada na tela da interface da aplicação por meio do método *setImageBitmap* chamado pelo objeto da classe *ImageView*.

5.5 OPENCV JAVA E OPENCV NATIVO

Como descrito no anexo 1, o OpenCV é uma biblioteca de grande suporte ao desenvolvimento de aplicações na área de Visão Computacional. O OpenCV possui interface Java, porém o conjunto de funcionalidades implementadas não é tão completo como as implementadas no OpenCV nativo. Além disso, uma parte significativa dessas funcionalidades em Java são chamadas ao código nativo através da interface do JNI. Junto a isso, a maior performance obtida com seu uso em C++, isto é, a linguagem nativa, completa o conjunto de razões para a utilização do OpenCV nativo nesta aplicação.

5.6 DISPOSITIVOS DE TESTES

Os dispositivos ilustrados pelas figuras 5.2, 5.3 e 5.4, foram utilizados neste projeto para a realização de testes da aplicação desenvolvida.



FIG. 5.2: Motorola Moto G3 - Processador Quad Core de 1.4GHz, 2 GB de RAM.



FIG. 5.3: Samsung Galaxy S7 - Processador Octa Core de 2.3GHz, 4GB de RAM.



FIG. 5.4: Samsung Google Nexus 10 - Processador Dual Core de 1.7GHz, 2 GB de RAM.

Como a capacidade operacional varia significativamente entre os dispositivos utilizados, foi possível notar a diferença de performance durante a execução da aplicação nos mesmos. Inicialmente, os testes foram realizados no dispositivo Motorola Moto G3, ilustrado na figura 5.2, porém, após a implementação de mais recursos, esse já não era mais capaz de atender aos requisitos de desempenho da aplicação. Dessa forma, foram necessários outros dispositivos com capacidade de processamento superiores, como os apresentados nas figuras 5.3 e 5.4, em que pôde-se executar a aplicação corretamente.

Além disso, um dispositivo virtual foi utilizado por meio do emulador fornecido pela plataforma Android Studio, IDE oficial do Android. O dispositivo escolhido para testes foi o Nexus 5 dentre uma série de dispositivos oferecidos como opção. Abaixo as imagens ilustram a seleção desse dispositivo na plataforma bem como as opções de configuração de Hardware oferecidas.

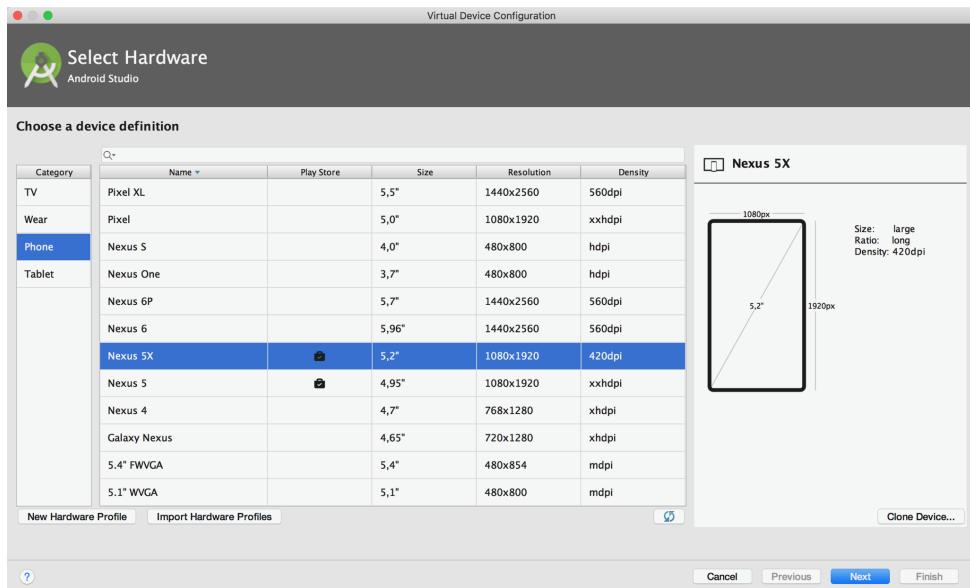


FIG. 5.5: Seleção do dispositivo virtual no Android Studio.

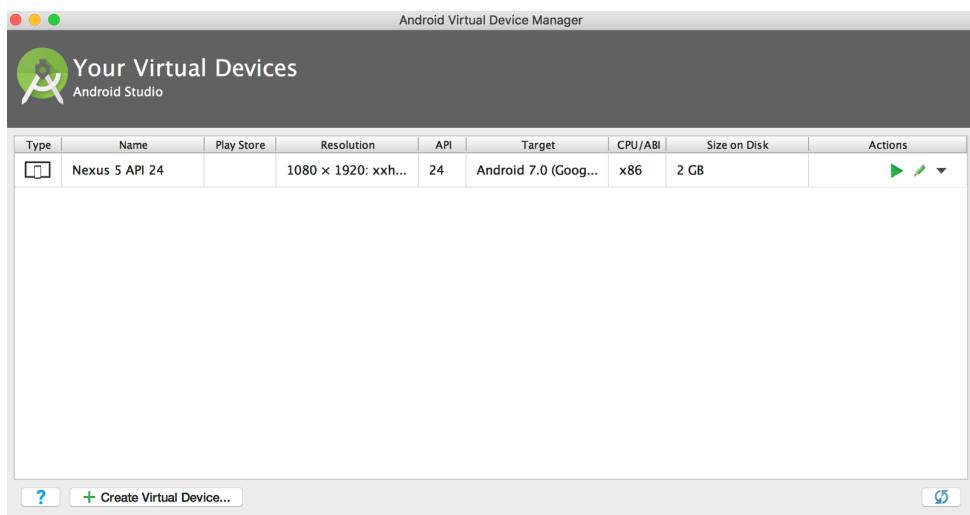


FIG. 5.6: Nexus 5 API 24.

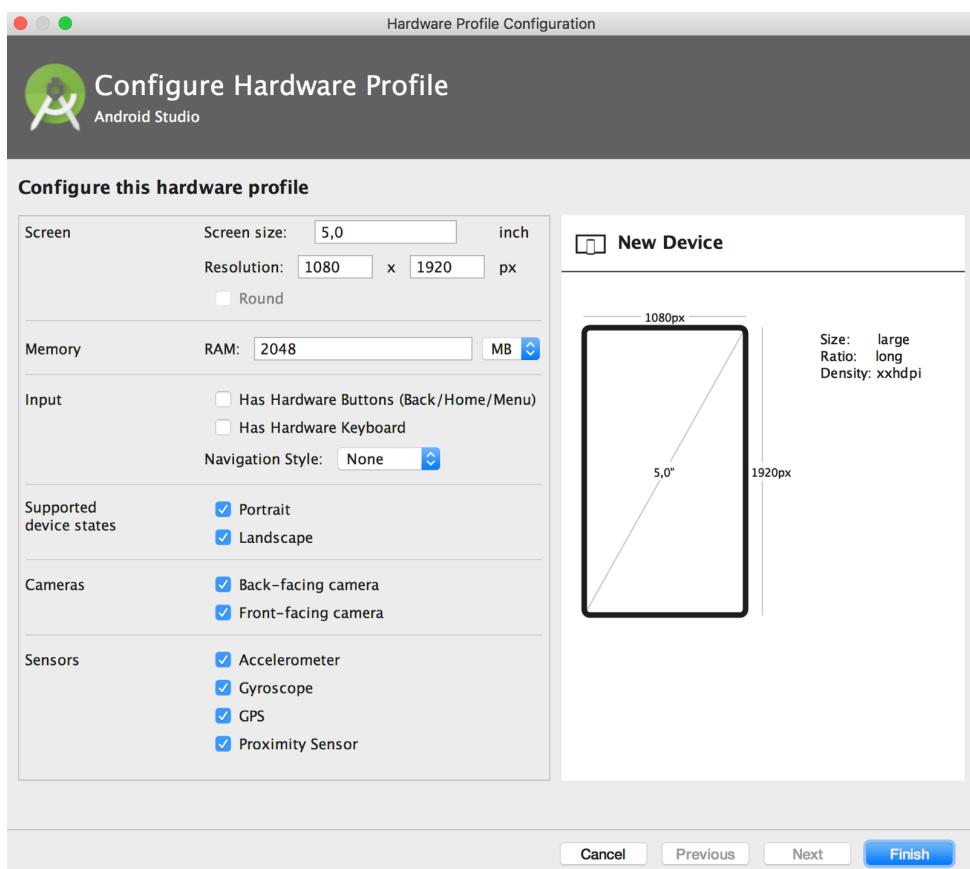


FIG. 5.7: Opções de configuração de hardware do dispositivo virtual.

Mais dispositivos poderiam ter sido testados com auxílio da API Firebase da Google.

5.6.1 FIREBASE

O Firebase é uma plataforma da Google focada no desenvolvimento de aplicativos que provê ferramentas para uma solução *back-end* completa para aplicações mobile (IOS e Android) bem como aplicações *Web*. A plataforma oferece diversas funcionalidades por meio do seu SDK, sendo uma delas um ambiente de testes chamado Firebase Test Lab, que oferece dispositivos físicos e virtuais para executar testes que simulam ambientes de uso real.

O Firebase Test Lab para Android oferece infraestrutura com base em nuvem para testar aplicativos Android e é totalmente integrado ao Android Studio para executar testes instrumentados e analisar os resultados dos testes. Developers (2017)

Além disso, esse produto conta ainda com uma ferramenta de teste integrada, chamada Robo, que analisa a estrutura da interface com o usuário da aplicação e a explora de modo metódico, simulando automaticamente as atividades do usuário.

5.6.1.1 LAYOUT

Além da diferença de performance na execução da aplicação entre os dispositivos utilizados, esses também diferem quanto à apresentação do layout do aplicativo.

Por padrão, o Android faz o redimensionamento do layout do aplicativo para caber na tela do dispositivo que fará a execução da aplicação. Tal redimensionamento tem um resultado satisfatório na maioria dos casos, porém, algumas vezes, é necessário fazer o ajuste da interface com o usuário (IU) para alguns dispositivos com tamanhos de tela específicos. Por exemplo, em telas pequenas, pode acontecer do layout não caber em suas dimensões, e, em telas maiores, pode ser que o layout não faça um uso eficiente do espaço disponível. Uma forma de solucionar esses problemas é por meio da criação de layouts alternativos, que otimizam o layout para os tamanhos de tela propostos e trazem uma experiência mais completa para o usuário.

Por terem diferentes tamanhos de tela, IU da aplicação não se comporta da mesma forma nos dispositivos testados. Enquanto que no dispositivo da figura 5.3, o layout do aplicativo fica bem compacto para caber na tela, no dispositivo da figura 5.4, que possui uma tela consideravelmente maior por se tratar de um *tablet*, sobra espaço não preenchido na tela.

6 APLICAÇÃO

O objetivo da aplicação como descrito na seção 1.2 do capítulo 1 é realizar a segmentação de imagens por meio de um aplicativo móvel na plataforma Android. O processo de segmentação é realizado pelo algoritmo de segmentação por técnica baseada em *watershed*, descrito no capítulo 4, e a imagem segmentada resultante é apresentada na tela do aplicativo.

6.1 AQUISIÇÃO DA IMAGEM

A aquisição da imagem pode ser feita de duas maneiras diferentes:

- Por meio do acesso à galeria de imagens do dispositivo; e
- Por meio da captura de uma imagem através da câmera do dispositivo.



FIG. 6.1: Tela Inicial do aplicativo.

6.1.1 GALERIA DE IMAGENS

Para escolher uma imagem a ser segmentada, deve-se clicar no botão "GALERIA" que levará a uma tela com uma imagem modelo já pré-definida e com três botões. Basta

clicar no botão "LOAD" para acessar as imagens existentes no dispositivo e, depois, clicar sobre a imagem desejada a fim de carregá-la na tela. A imagem escolhida será apresentada na tela, substituindo a imagem anterior.

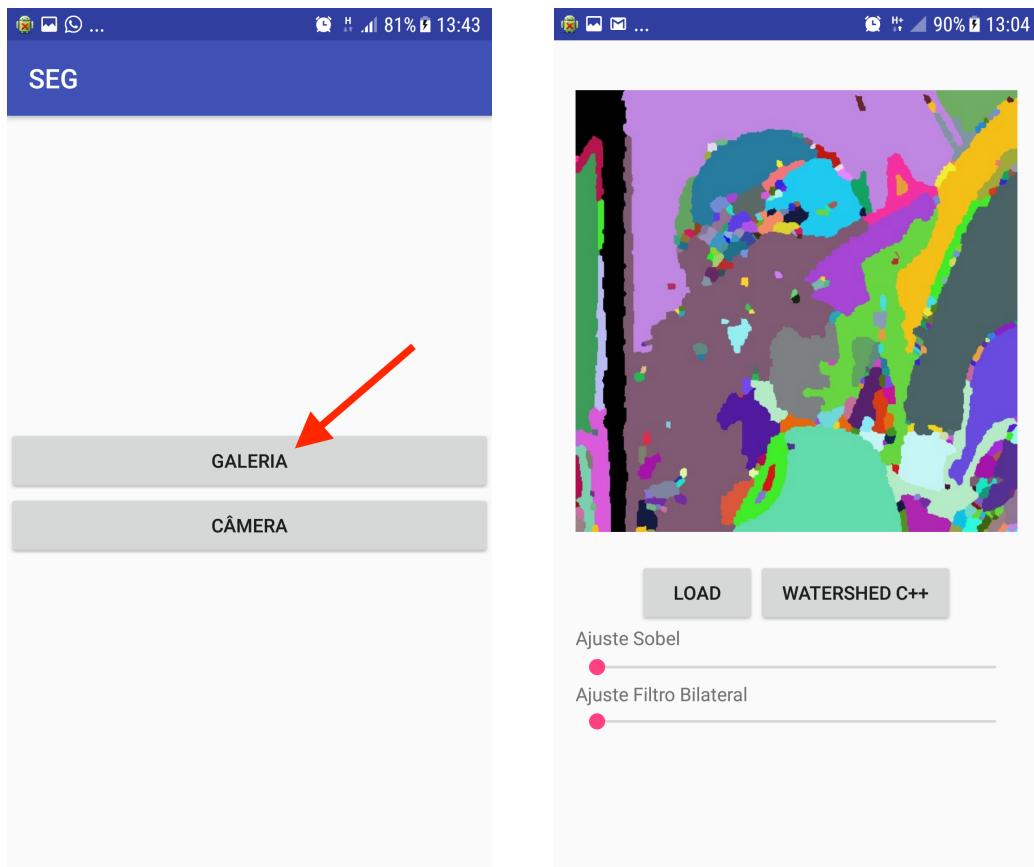


FIG. 6.2: Sequência de atividades para aquisição da imagem por meio da galeria de imagens.

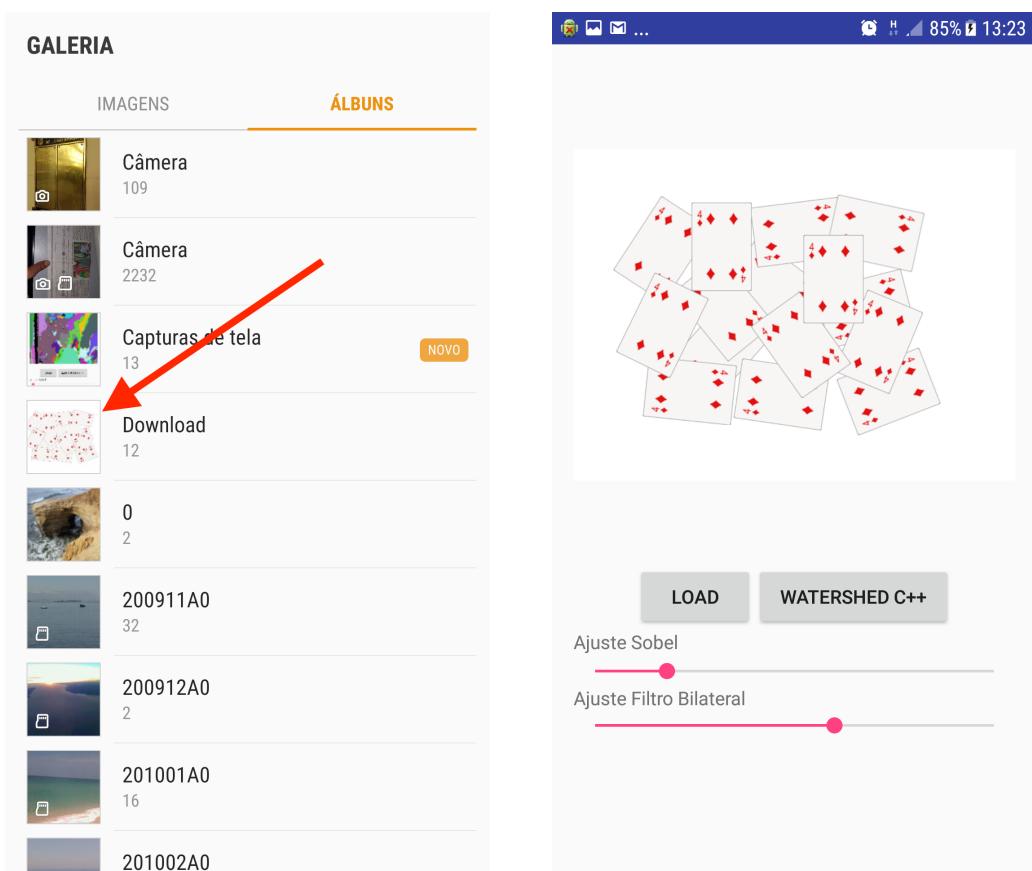


FIG. 6.3: Sequência de atividades para aquisição da imagem por meio da galeria de imagens.

6.1.2 CÂMERA

Uma nova imagem pode ser adquirida a qualquer momento para ser utilizada na segmentação por meio do acesso à câmera do dispositivo. Para acessar a câmera, basta clicar no botão "CÂMERA" apresentado na tela inicial da aplicação. Com a câmera funcionando, pode-se aplicar diversos procedimentos de pré-processamento nas imagens em tempo real:

- Conversão para Cinza;
- Operador Sobel;
- Filtro Biltateral;
- Operação Morfológica;
- SIFT;



FIG. 6.4: Sequência de atividades para aquisição e tratamento das imagens por meio da câmera do dispositivo.

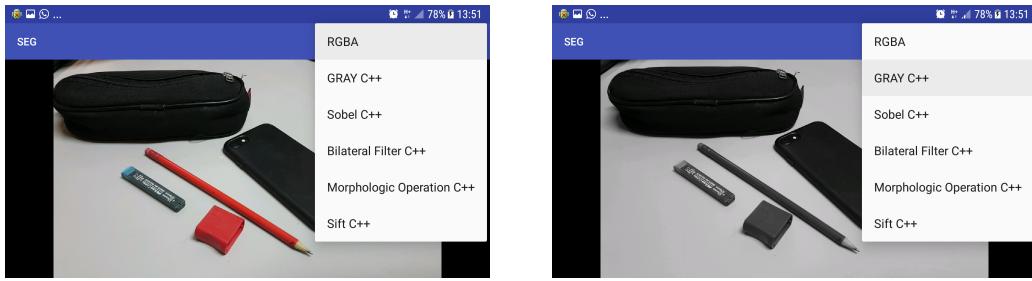


FIG. 6.5: Sequência de atividades para aquisição e tratamento das imagens por meio da câmera do dispositivo.

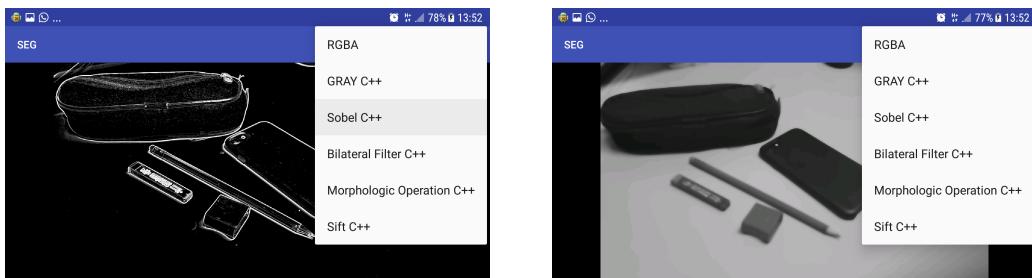


FIG. 6.6: Sequência de atividades para aquisição e tratamento das imagens por meio da câmera do dispositivo.

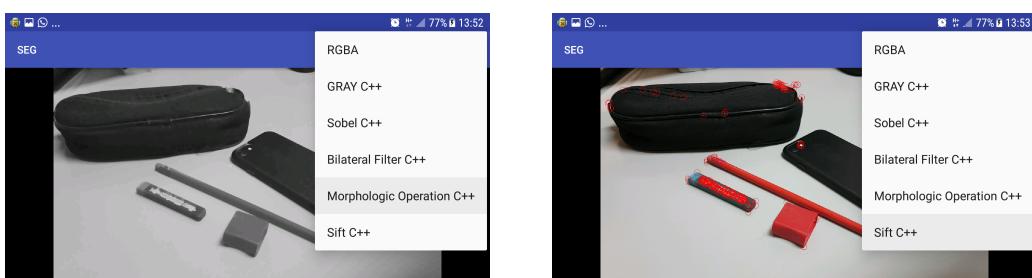


FIG. 6.7: Sequência de atividades para aquisição e tratamento das imagens por meio da câmera do dispositivo.

6.2 SEGMENTAÇÃO DA IMAGEM

A segmentação da imagem, é feita por meio do algoritmo desenvolvido para esta aplicação.

6.2.1 WATERSHED DESENVOLVIDO

Após a escolha da imagem, ao clicar sobre o botão "watershed", a imagem escolhida será segmentada de acordo com o algoritmo desenvolvido neste projeto. Antes disso, pode-se ainda ajustar os parâmetros do Filtro Bilateral e do Operador Sobel por meio de dois *seekbars*. A sequência de *steps* apresentados na seção 4.3 do capítulo 4 é executada pelo processador do dispositivo utilizado e o resultado da imagem segmentada é apresentada na tela.

A seguir são exibidos os resultados da segmentação para algumas das imagens testadas.

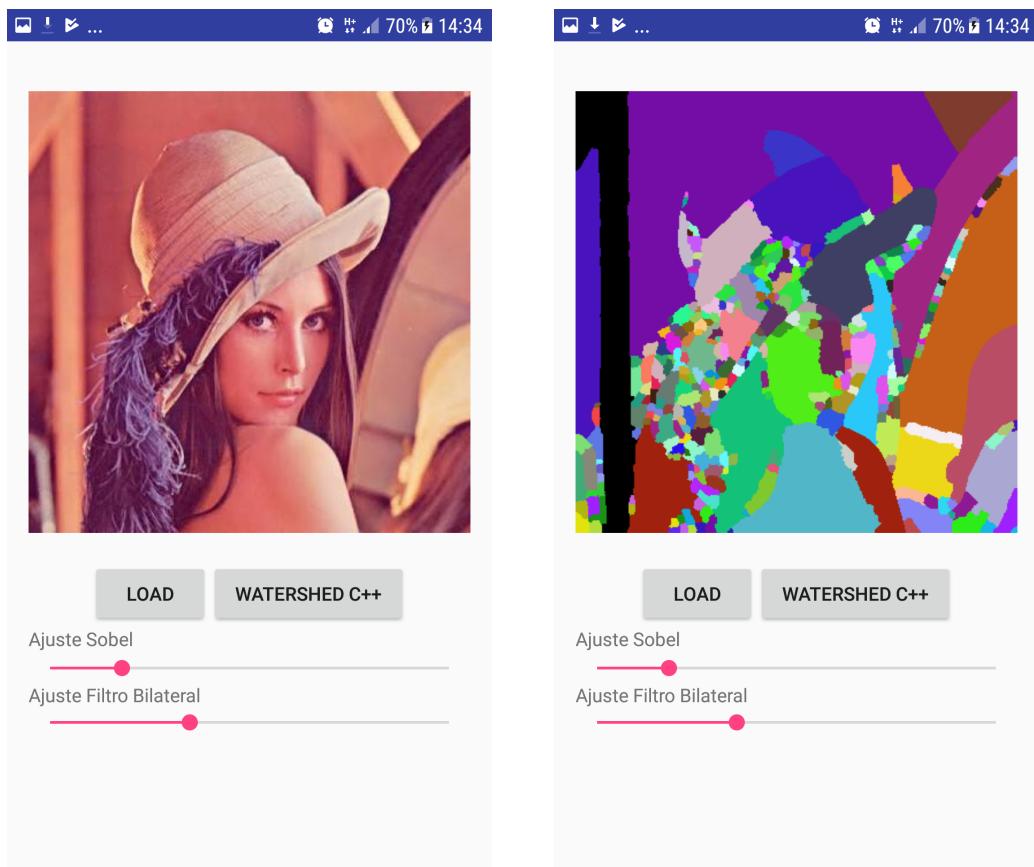


FIG. 6.8: Resultados de segmentação utilizando o algoritmo *watershed* desenvolvido.



FIG. 6.9: Resultados de segmentação utilizando o algoritmo *watershed* desenvolvido.

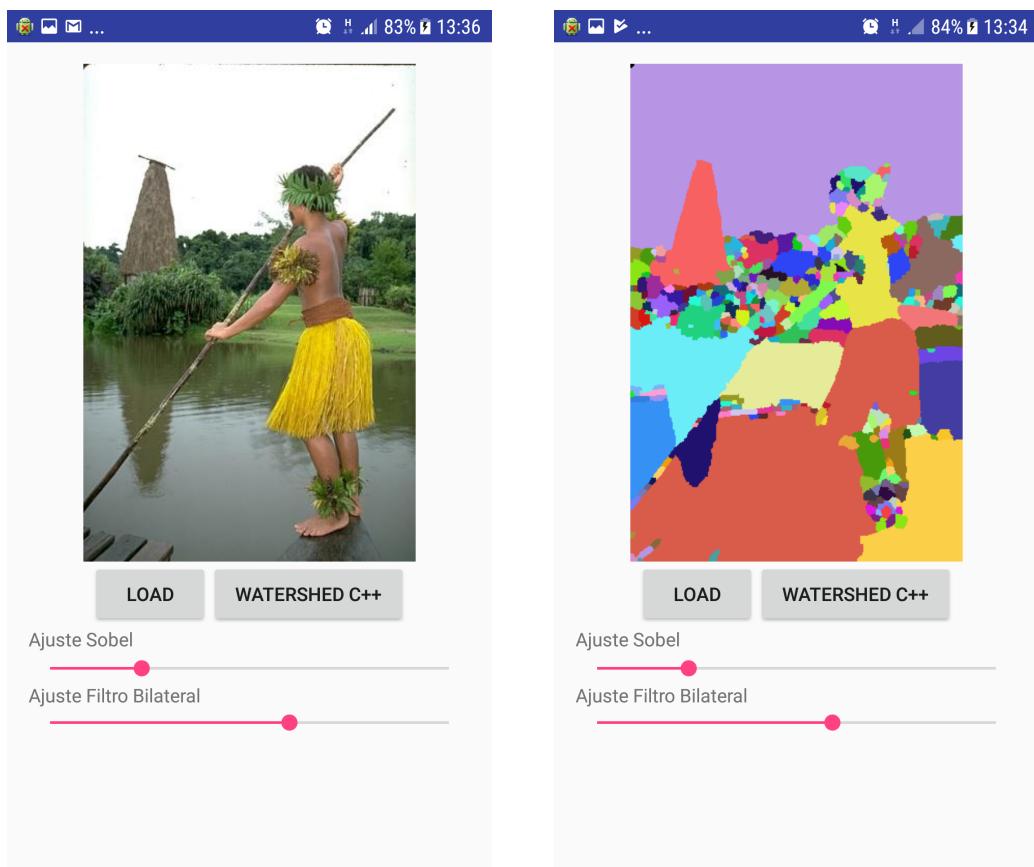


FIG. 6.10: Resultados de segmentação utilizando o algoritmo *watershed* desenvolvido.

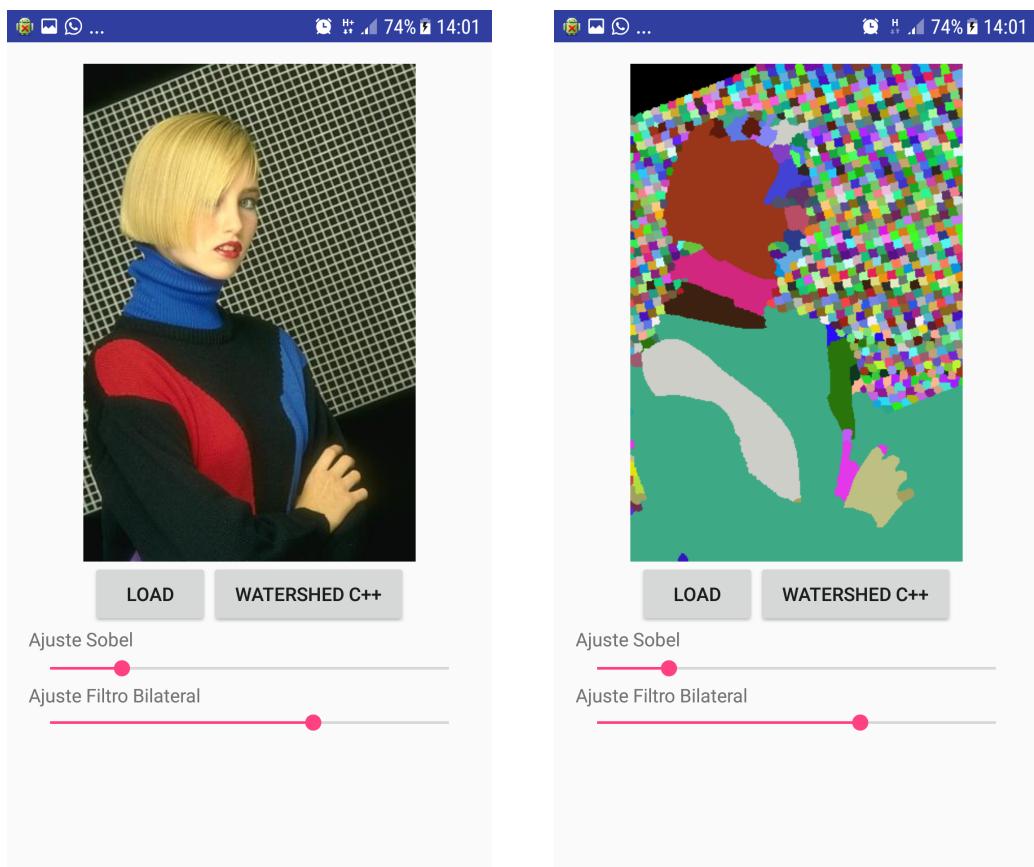


FIG. 6.11: Resultados de segmentação utilizando o algoritmo *watershed* desenvolvido.

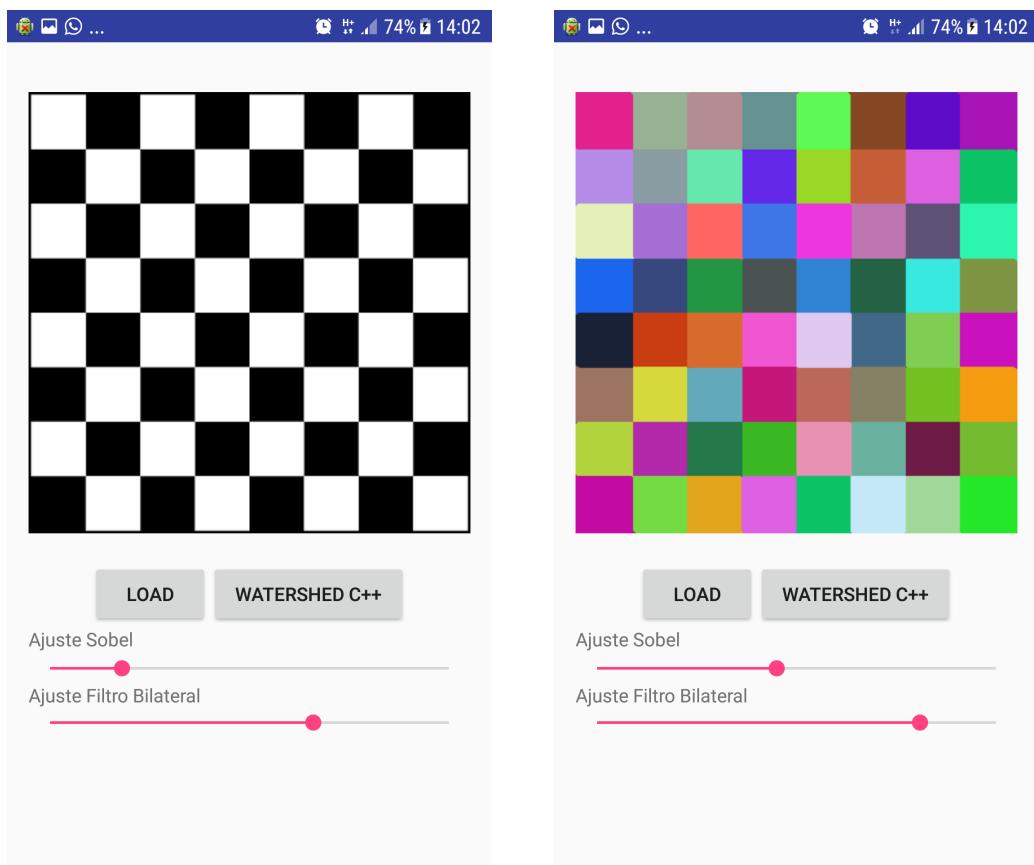


FIG. 6.12: Resultados de segmentação utilizando o algoritmo *watershed* desenvolvido.

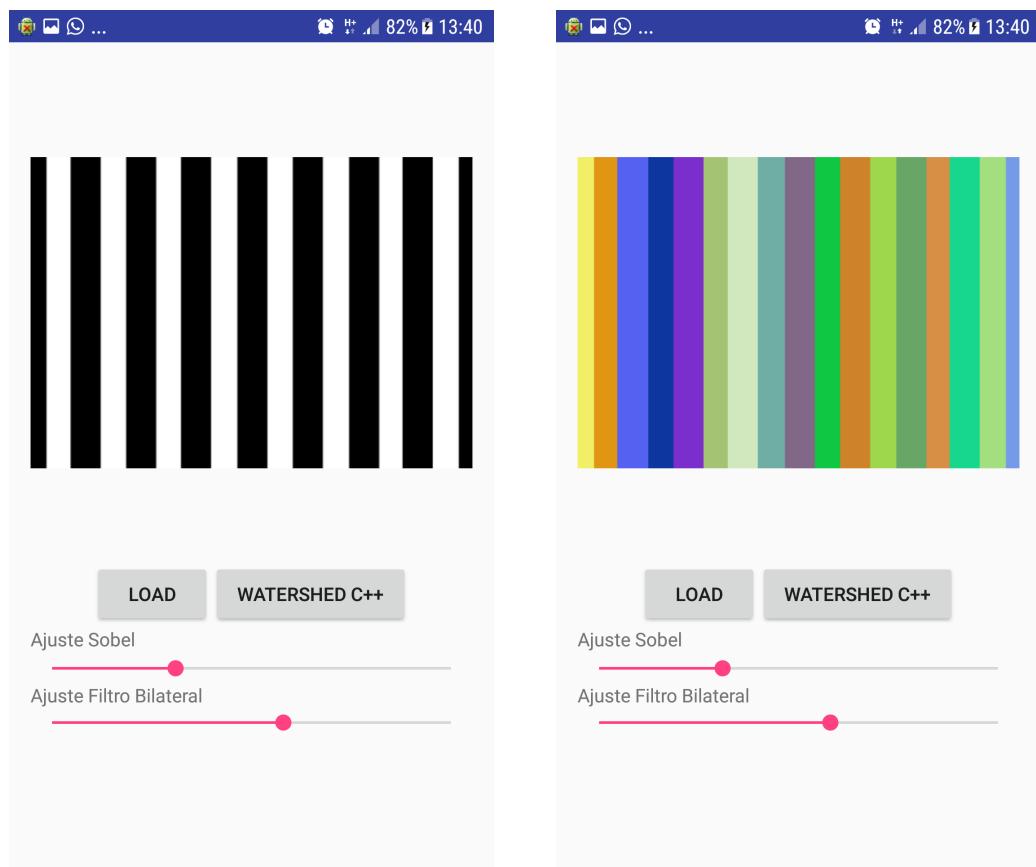


FIG. 6.13: Resultados de segmentação utilizando o algoritmo *watershed* desenvolvido.

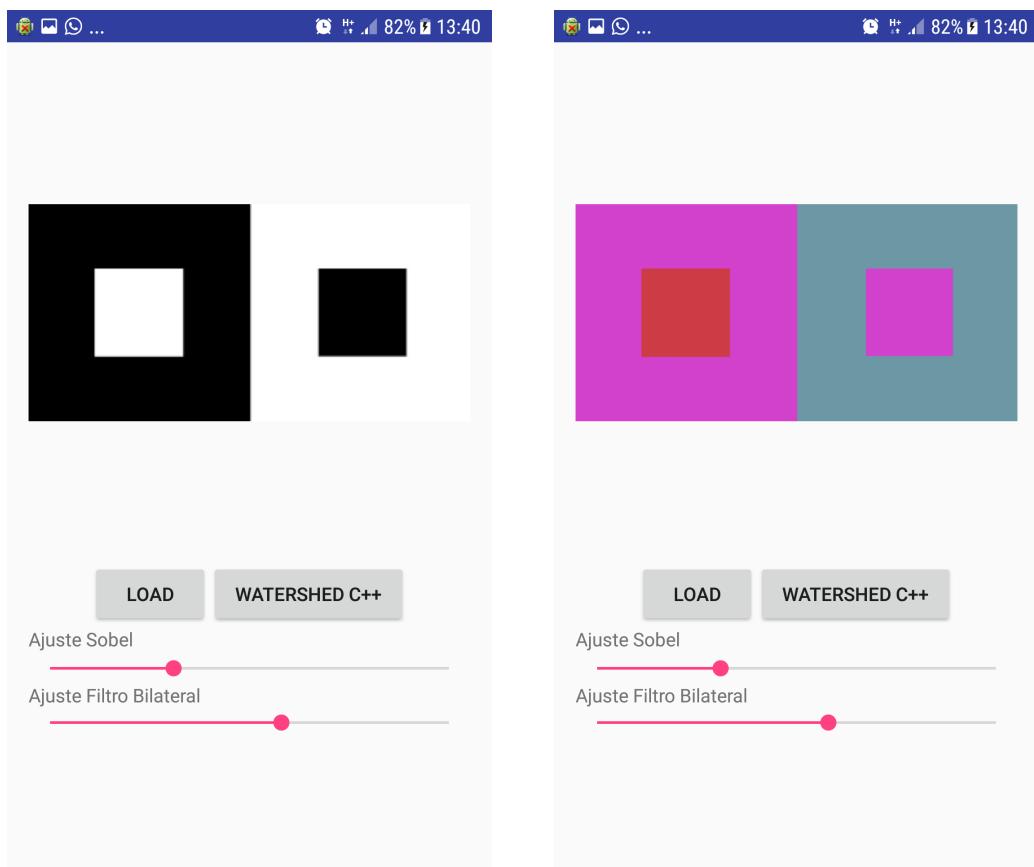


FIG. 6.14: Resultados de segmentação utilizando o algoritmo *watershed* desenvolvido.

7 CONCLUSÃO

O projeto atingiu o objetivo principal de desenvolver uma aplicação móvel em ambiente Android para a segmentação de imagens. Todo o estudo sobre o processamento digital de imagens e as diversas técnicas de segmentação formaram a base do conhecimento empregada na concepção desse produto. Além disso pôde-se perceber a importância crescente de sistemas de visão computacional no tratamento de imagens a medida que se busca aproximar da visão e da análise humana bem como suas aplicações.

Como trata-se de um problema mal colocado, isto é, sem solução universal, foi implementado uma técnica de segmentação baseada no algoritmo *watershed* a partir da qual foi possível obter resultados bastante satisfatórios. Certamente, o desenvolvimento de novas técnicas de segmentação e suas implementações tornarão a aplicação mais completa, além de permitir uma análise comparativa dos resultados obtidos, levando a uma ferramenta mais útil para novos projetos de pesquisa na área da visão computacional. Para isso o aplicativo desenvolvido está disponível na plataforma do GitHub, no *link* <https://github.com/brunoavelino/SEG>, a fim de permitir a colaboração de estudantes e pesquisadores interessados. A possível integração dessa aplicação com sistemas de manipulação de imagens que necessitam da segmentação como etapa de seus processos é também outra forma de criação de valor por meio deste projeto.

O aprendizado obtido nas etapas de desenvolvimento desse projeto e a criação de um produto com o potencial de impactar positivamente a comunidade científica superaram as expectativas iniciais e satisfizeram seus objetivos.

8 CRONOGRAMA

8.1 DEFINIÇÃO DE ETAPAS

8.1.1 ESCOLHA DO TEMA E ESTUDO DE VIABILIDADE

A escolha do tema foi a primeira etapa do projeto. O uso de dispositivos móveis, bem como de suas respectivas câmeras tem sido cada dia mais frequentes. Essas câmeras captam informações do ambiente que os cercam e a segmentação de imagem pode ser usado no processamento de imagens de forma a desenvolver soluções computacionalmente automatizáveis.

Após a escolha do tema, foi feito o estudo de viabilidade, de forma a se verificar a possibilidade da cumprimento do objetivo do tema em tempo aceitável.

8.1.2 REVISÃO BIBLIOGRÁFICA

Referências tais como livros, artigos e outras referências foram estudadas durante o período de revisão bibliográfica, de forma a permitir um embasamento bibliográfico do projeto.

8.1.3 ELABORAÇÃO DA MONOGRAFIA

Esta fase do projeto se estende até o término do projeto de fim de curso. Nesta última, confecciona-se um relatório utilizando-se os conhecimentos adquiridos desde o início do projeto.

8.1.4 ESTUDO E ANÁLISE DOS ALGORITMOS DE SEGMENTAÇÃO DE IMAGEM

Na aplicação proposta, alguns algoritmos de segmentação de imagem serão implementados no dispositivo móvel, de modo a estudá-los e verificar o mais apropriado à aplicação.

8.1.5 IMPLEMENTAÇÃO

A implementação ocorrerá após o estudo dos algoritmos e da definição de quais deles são mais indicados à proposta. Em uma primeira fase os algoritmos serão implementados em outros ambientes de desenvolvimento e em uma segunda etapa será realizada a portabilidade para o ambiente Android.

8.1.6 TESTE

Os testes com as imagens em diferentes algoritmos serão realizados e, com a implementação pronta, pode-se observar e comparar os resultados dos algoritmos.

8.1.7 ENTREGA DO RELATÓRIO FINAL E APRESENTAÇÃO

A partir das análises e de todo conteúdo, implementação, análises e testes, será feito o relatório e a apresentação à banca.

8.2 ENTREGÁVEIS

Os entregáveis serão:

- Projeto do Aplicativo
- Escolha dos Algoritmos
- Aplicativo em Ambiente Android de Segmentação de Imagens implementado
- Testes e análises comparativas

Etapa	Meses								
	FEV	MAR	ABR	MAI	JUN	JUL	AGO	SET	OUT
Escolha do Tema e Estudo de Viabilidade	X	X	X						
Revisão Bibliográfica	X	X	X	X					
Elaboração da Monografia	X	X	X	X	X	X	X	X	X
Estudo e Análise dos Algoritmos de Segmentação de Imagem						X	X	X	
Implementação						X	X	X	
Teste								X	
Entrega do Relatório Final e Apresentação									X

TAB. 8.1: Cronograma das atividades previstas

9 REFERÊNCIAS BIBLIOGRÁFICAS

DEVELOPERS ANDROID. Arquitetura da plataforma. Disponível em: <<https://developer.android.com/guide/platform/index.html?hl=pt-br>>. Acesso em: 24 set. de 2017.

DEVELOPERS ANDROID. Primeiros passos com o NDK. Disponível em: <<https://developer.android.com/ndk/guides/index.html?hl=pt-br>>. Acesso em: 10 mai. de 2017.

DEVELOPERS ANDROID. Tudo de que você precisa para criar aplicativos no Android. Disponível em: <<https://developer.android.com/studio/features.html>>. Acesso em: 10 mai. de 2017.

ARBELAEZ, P.; MAIRE, M.; FOWLKES, C. ; MALIK, J. Contour detection and hierarchical image segmentation. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 33, n. 5, p. 898–916, 2011.

BANTERLE, F.; CORSINI, M.; CIGNONI, P. ; SCOPIGNO, R. A low-memory, straightforward and fast bilateral filter through subsampling in spatial domain. **Computer Graphics Forum**, v. 31, n. 1, p. 19–32, 2012.

THE BERKELEY SEGMENTATION DATASET AND BENCHMARK. Berkeley Segmentation Dataset. Disponível em: <<https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/BSDS300>>. Acesso em: 10 mai. de 2017.

GOOGLE DEVELOPERS. Firebase Test Lab para Android. Disponível em: <<https://firebase.google.com/products/test-lab/?hl=pt-br>>. Acesso em: 23 set. de 2017.

ABEL GOMES. Computação Visual e Multimídia. Disponível em: <<http://www.di.ubi.pt/~agomes/cvm/teoricas/07-regionsegmentation.pdf>>. Acesso em: 10 mai. de 2017.

OPENCV. More Morphology Transformations. Disponível em: <<http://docs.opencv.org/2.4/doc/tutorials/imgproc/openingclosinghats/openingclosinghats.html>>. Acesso em : 26 jul. de 2017.

OPENCV. OpenCV About. Disponível em: <<http://opencv.org/about.html>>. Acesso em: 10 mai. de 2017.

OPENCV. Sobel Derivatives. Disponível em: <http://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/sobel_derivatives/sobel_derivatives.html>. Acesso em : 26 jul. de 2017.

RUPARELIA, S. **Implementation of watershed based image segmentation algorithm in FPGA.** 2012. Dissertação (M.Sc. Information Technology) – Universidade de Stuttgart, Alemanha, 2012.

STANFORD. The Stanford Center for Image Systems Engineering. Disponível em: <<https://scien.stanford.edu/index.php/test-images-and-videos/>>. Acesso em: 10 mai. de 2017.

ROBIN STRAND. Segmentation. Disponível em: <www.it.uu.se/edu/course/homepage/bild1/ht14/L6_segmentation.pdf> . Acesso em : 10 mai. de 2017.

10 ANEXOS

ANEXO 1: FERRAMENTAS

10.2 ANDROID STUDIO

Plataforma de desenvolvimento de aplicativos para sistemas Android. Criado especificamente para esse propósito, a IDE oficial do Android oferece diversos recursos que aceleram o desenvolvimento e aumentam a qualidade dos aplicativos criados.

Tais recursos como ferramentas de edição, depuração, testes e geração de perfis de código motivaram a escolha dessa plataforma para o desenvolvimento da aplicação dessa pesquisa. Além da programação na linguagem Java, é permitido o desenvolvimento em linguagem nativa, C/C++, por meio da ferramenta Android NDK, que será utilizada na implementação dos algoritmos de segmentação presentes nessa aplicação. (ANDROID, 2017c).

10.3 OPENCV

Biblioteca *open source* para o desenvolvimento de aplicativos na área de Visão Computacional. Conta com mais de 2500 algoritmos otimizados com abordagens clássicas e no estado da arte nas áreas de visão computacional e aprendizado de máquina, sendo, portanto, uma ferramenta amplamente usada em aplicações que envolvem o processamento de imagens (mais de 14 milhões de *downloads*). Tem interfaces para linguagens como C++, C, Python e Java e suporta diferentes plataformas como Windows, Linux, Mac OS e Android. A maior performance é obtida com seu uso em C++ pelo fato de ser sua linguagem nativa. (OPENCV, 2017b)

10.4 ANDROID SDK

O *Android Software Development Kit* (SDK) é um conjunto de ferramentas disponibilizados pela Google para o desenvolvimento de aplicações para a plataforma Android.

10.5 ANDROID NDK

O *Android Native Development Kit* (NDK) é um conjunto de ferramentas que permite a codificação de arquivos de projeto em linguagens nativas como C/C++. Essa ferramenta auxiliará na implementação dos algoritmos de segmentação de imagens que farão parte de nossa aplicação. (ANDROID, 2017b)