UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DAELN - DEPARTAMENTO ACADÊMICO DE ELETRÔNICA
ENGENHARIA ELETRÔNICA

CHARLES REIS RIBEIRO, MATEUS PRZYSIADA ZEM

APLICATIVO PARA SMARTPHONE PARA MEDIÇÃO DA ÁREA DE FERIDAS DEMARCADAS EM FOLHAS TRANSPARENTES

TRABALHO DE CONCLUSÃO DE CURSO

CURITIBA
2019

CHARLES REIS RIBEIRO, MATEUS PRZYSIADA ZEM

APLICATIVO PARA SMARTPHONE PARA MEDIÇÃO DA ÁREA DE FERIDAS DEMARCADAS EM FOLHAS TRANSPARENTES

Trabalho de Conclusão de Curso apresentado ao Engenharia Eletrônica da Universidade Tecnológica Federal do Paraná, como requisito parcial para a obtenção do título de Bacharel em Engenharia Eletrônica.

Orientador: Gustavo Benvenutti Borba
UTFPR

Coorientador: Adriano Antônio Mehl
UTFPR

CURITIBA
2019

CHARLES REIS RIBEIRO

MATEUS PRZYSIADA ZEM

APLICATIVO PARA *SMARTPHONE* PARA MEDIÇÃO DA ÁREA DE FERIDAS DEMARCADAS EM FOLHAS TRANSPARENTES

Este Trabalho de Conclusão de Curso de Graduação foi apresentado como requisito parcial para obtenção do título de Engenheiro Eletrônico, do curso de Engenharia Eletrônica do Departamento Acadêmico de Eletrônica (DAELN) outorgado pela Universidade Tecnológica Federal do Paraná (UTFPR). Os alunos foram arguidos pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

Curitiba, 12 de julho de 2019.

____

Prof. Dr. Robinson Vida Noronha
Coordenador de Curso
Engenharia Eletrônica

____

Profª. Drª. Carmen Caroline Rasera
Responsável pelos Trabalhos de Conclusão de Curso
de Engenharia Eletrônica do DAELN

BANCA EXAMINADORA

____

Prof. Dr. Gustavo Benvenutti Borba
Universidade Tecnológica Federal do Paraná
Orientador

____

Prof. Me. Adriano Antônio Mehl
Universidade Tecnológica Federal do Paraná
Coorientador

____

Prof. Me. Valfredo Pilla Júnior
Universidade Tecnológica Federal do Paraná

____

Prof. Me. Daniel Rossato de Oliveira
Universidade Tecnológica Federal do Paraná

____

Prof. Dr. Bertoldo Schneider Júnior
Universidade Tecnológica Federal do Paraná

A folha de aprovação assinada encontra-se na Coordenação do Curso de Engenharia Eletrônica.

AGRADECIMENTOS

Agradecemos o Programa de Educação Tutorial (PET) Engenharia Eletrônica da Universidade Tecnológica Federal do Paraná, Curitiba, programa no qual tivemos o primeiro contato com a ideia da criação do aplicativo deste trabalho.

A imaginação muitas vezes nos leva a mundos que nunca sequer existiram. Mas sem ela, não vamos a lugar nenhum (SAGAN, Carl, 1980).

RESUMO

RIBEIRO, Charles Reis; ZEM, Mateus Przysiada. APLICATIVO PARA SMARTPHONE PARA MEDIÇÃO DA ÁREA DE FERIDAS DEMARCADAS EM FOLHAS TRANSPARENTES. 2019. 60 f. Trabalho de Conclusão de Curso – Engenharia Eletrônica, Universidade Tecnológica Federal do Paraná. Curitiba, 2019.

A hiperglicemia em pessoas portadoras de diabetes causa deficiências no processo de cicatrização da pele. Estas deficiências na cicatrização podem resultar em feridas graves, que exigem tratamentos específicos, para os quais é necessário o acompanhamento da área da ferida. Um método bastante difundido e eficaz para a medição da área é sobrepor uma folha transparente na ferida e traçar nesta folha o seu contorno. Posteriormente, realiza-se a medida (cálculo) da área. No entanto, é possível afirmar que o cálculo manual não fornece o valor exato da área da ferida, já que os contornos tendem a ser irregulares. Neste trabalho, apresenta-se o desenvolvimento e a avaliação de um aplicativo para *smartphone* que realiza o cálculo desta área com erros menores se comparados ao cálculo manual. O usuário captura a imagem da folha contendo o contorno da ferida utilizando a câmera do *smartphone* e um algoritmo de segmentação semi-automática deste contorno realiza o cálculo da área. Para os testes, foram criados contornos de áreas conhecidas e os erros relativos médios obtidos no cálculo destas áreas foram entre 0,8 e 3,4%. Estes resultados são comparáveis àqueles relatados em trabalhos similares, o que comprova que a solução desenvolvida neste trabalho é confiável e pode auxiliar os profissionais de saúde na tarefa de acompanhar as alterações nas áreas das feridas durante o tratamento.

Palavras-chave: Feridas em diabéticos, Medição de área, Segmentação semi-automática, Aplicativo Android

ABSTRACT

RIBEIRO, Charles Reis; ZEM, Mateus Przysiada. Smartphone app for wound area measurement using an image of transparent sheet with the borders drawn. 2019. 60 f. Trabalho de Conclusão de Curso – Engenharia Eletrônica, Universidade Tecnológica Federal do Paraná. Curitiba, 2019.

Hyperglycemia in people with diabetes causes deficiencies in the healing process of the skin. Those deficiencies in the healing process may cause severe wounds, which demands specific treatments, requiring constant monitoring of the wound area progression. A very widespread and effective method for measuring the wound area is placing a transparent film over the wound and tracing the wound contour on the sheet. Next, the wound area is measured (calculated). However, it is possible to state that the manual calculation does not provide the exact value of the area of the wound since the contours tend to be irregular. This work presents the development and evaluation of a smartphone application, which calculates the wound area with minor errors when compared to the manual calculation. The user captures the image of the film containing the wound contour using the smartphone camera and a semi-automatic segmentation algorithm of this contour performs the area calculation. For the tests, contours of known areas were created and the mean relative errors obtained in the calculation of these areas were between 0.8 and 3.4 %. These results are comparable to those reported in similar studies, which proves that the solution developed in this work is reliable and can assist health professionals in the task of monitoring changes in wound areas during treatment.

Keywords: Wound in diabetics, Area measurement, Semi-automatic segmentation, Android App

LISTA DE FIGURAS

Figura 1 – Exemplo de feridas nas extremidades do corpo em diabéticos. 21
Figura 2 – Régua para o desenho do contorno da ferida. 23
Figura 3 – Exemplo de medição utilizando o *software* Digimizer. 24
Figura 4 – Resultado da medição utilizando o *software* Digimizer. 24
Figura 5 – Medição de ferida usando Visitrak. Na imagem, observa-se o acetato no qual desenha-se o contorno da ferida, posicionado sobre o *tablet*. 25
Figura 6 – Exemplo de uso da plataforma WoundMatrix. 26
Figura 7 – Diagrama em blocos do algoritmo. 27
Figura 8 – Fluxograma do algoritmo de seleção automática dos quatro vértices da grade. O algoritmo do bloco *binariza com valores de gaussianblurValue e cannyValue* é descrito na Figura 9. 28
Figura 9 – Fluxograma do algoritmo de binarização utilizado pela função de seleção automática dos quatro vértices da grade, apresentado na Figura 8, no bloco denominado *binariza com valores gaussianblurValue e cannyValue* 29
Figura 10 – Operação de convolução em imagens. Exemplo para uma máscara de convolução 3x3. 29
Figura 11 – Afinamento do Canny 30
Figura 12 – Gráfico de histerese 30
Figura 13 – Exemplo de resultado do algoritmo de detecção de bordas de Canny. À esquerda a imagem original e à direita a imagem com as bordas detectadas. 31
Figura 14 – Exemplo de saída da função *findContours* do OpenCV para a imagem de entrada da Figura 13. 32
Figura 15 – Exemplo de resultado da função *approxPolyDP*. À esquerda um exemplo de falha na seleção do maior contorno. À direita, mostra-se um exemplo de sucesso ao detectar o maior contorno 33
Figura 16 – Régua em cores e em tons de cinza. 35
Figura 17 – À esquerda, resultado da binarização usando um único limiar, à direita, resultado da binarização usando o método descrito por (BRADLEY, 2007) 35
Figura 18 – Seleção manual do interior da ferida. 37
Figura 19 – Imagem original e imagem afinada. 37
Figura 20 – Ordem dos pixeis 37
Figura 21 – Ação do erode 38
Figura 22 – Elemento estrutural 3x3 39
Figura 23 – Linhas que cruzam a imagem afinadas e resultado após erosão 39
Figura 24 – Diagrama UML do aplicativo desenvolvido. 40

Figura 25 – Exemplo de zoom usando uma *view* customizada da biblioteca Subsampling Scale Image. View 42
Figura 26 – Diagrama UML do AppService para persistência dos dados e comunição entre as atividades. 42
Figura 27 – Tela principal mostrando a aparência do aplicativo após a sua inicialização. 43
Figura 28 – Tela de delimitação das quatro extremidades da grade da régua. 44
Figura 29 – Tela de seleção de limiar *(threshold)* da operação de Bradley Local Threshold. Ao interagir com a barra da parte inferior, o usuário pode ajustar o nível de binarização, fazendo com que se evite a escolha de imagens com contornos não contínuos. 45
Figura 30 – Fluxograma da classe Flood Fill. 46
Figura 31 – Operação de Flood Fill completada após os toques necessários do usuário . 46
Figura 32 – Tela final exibindo o valor da área obtida da ferida. 47
Figura 33 – Exemplo de amostra seguindo as condições citadas. 48
Figura 34 – Exemplo de amostra simulando o formato de uma ferida. 50
Figura 35 – Exemplo de amostra seguindo as condições citadas 50
Figura 36 – Amostra de má qualidade antes da binarização. 51
Figura 37 – Amostra de má qualidade. 51
Figura 38 – Problema decorrente do contorno com falha. 52

LISTA DE QUADROS

Quadro 1 – Curabilidade de feridas. 20

LISTA DE TABELAS

Tabela 1 – Resultado dos testes com circunferências de raio conhecido e erros de outros aplicativos. 49
Tabela 2 – Resultado dos testes com figuras que simulam feridas. 50
Tabela 3 – Resultado dos testes sem favorecer a qualidade da medição 51

LISTA DE ABREVIATURAS E SIGLAS

API	*Application programming interface*
DAELN	Departamento Acadêmico de Eletrônica
DFU	*Diabetic Foot Ulcer*
DFS	*Diabetic foot syndrome*
DM	*Diabetes Mellitus*
GUI	*Graphical User Interface*
NDK	*Native Development Kit*
OpenCV	*Open source computer vision*
PDI	Processamento digital de imagem

LISTA DE ALGORITMOS

Algoritmo 1 – FloodFill 36

SUMÁRIO

1 – INTRODUÇÃO **15**
1.1 CONTEXTO E JUSTIFICATIVA 15
1.2 OBJETIVO GERAL 16
1.3 OBJETIVOS ESPECÍFICOS 16
1.4 ESTRUTURA DO DOCUMENTO 17

2 – FUNDAMENTAÇÃO TEÓRICA **18**
2.1 FERIDAS 18
2.1.1 CAUSAS DE FERIDAS 18
2.1.2 FASES DA CICATRIZAÇÃO DE FERIDAS 18
2.1.2.1 HOMEOSTASE 18
2.1.2.2 INFLAMAÇÃO 19
2.1.2.3 PROLIFERAÇÃO 19
2.1.2.4 MATURAÇÃO 19
2.2 PROCESSO DE CURA DE FERIDAS 19
2.3 DIABETES 19
2.4 FERIDAS EM DIABÉTICOS 20
2.5 MEDIÇÃO DE FERIDAS 21
2.5.1 PARÂMETROS DIMENSIONAIS DE FERIDAS 21
2.5.2 MEDIÇÕES APROXIMADAS DE FERIDAS 22
2.6 SOLUÇÕES EXISTENTES PARA A MEDIÇÃO DA ÁREA DE FERIDAS . . 22
2.6.1 RÉGUA UTILIZADA NESTE TRABALHO 22
2.6.2 DIGIMIZER 23
2.6.3 VISITRAK 25
2.6.4 WOUNDMATRIX 25

3 – ALGORITMO PARA MEDIÇÃO DA ÁREA DE FERIDAS **27**
3.1 SELEÇÃO DA GRADE DA RÉGUA 27
3.1.1 FILTRO GAUSSIANO 27
3.1.2 DETECÇÃO DE BORDAS DE CANNY (CANNY EDGE DETECTION) 27
3.1.3 FUNÇÃO FINDCONTOURS DO OPENCV 31
3.1.4 FUNÇÃO APPROXPOLYDP DO OPENCV 31
3.1.5 GETPERSPECTIVETRANSFORM E WARPPERSPECTIVE 32
3.1.5.1 GETPERSPECTIVETRANSFORM 33
3.1.5.2 WARPPERSPECTIVE 34
3.2 BINARIZAÇÃO 34

3.3 SELEÇÃO DA REGIÃO DE INTERESSE 36
3.4 CÁLCULO DA ÁREA 39

4 – APLICATIVO ANDROID 40
4.1 OPENCV 4.0 41
4.2 GERENCIAMENTO E EXIBIÇÃO DE IMAGENS 41
4.3 PERSISTÊNCIA DE DADOS E COMUNICAÇÃO ENTRE ATIVIDADES . . 41
4.4 *TELA PRINCIPAL* 43
4.5 CLASSE IMAGEADJUSTMENTACTIVITY 43
4.6 CLASSE *BINARIZATION* 44
4.7 CLASSE *FLOODFILL* 45
4.8 *CLASSE SKELETON* 47

5 – ANÁLISE E DISCUSSÃO DOS RESULTADOS 48

6 – CONCLUSÕES 53
6.1 TRABALHOS FUTUROS 54

Referências 55

Anexos 56

ANEXO A – Implementação em Java do algoritmo Zhang-Suen Thinning . . . 57

1 INTRODUÇÃO

No ano de 2016 o Ministério da Saúde apontou que aproximadamente 8,9% da população brasileira sofria de diabetes, doença caracterizada pela resistência do corpo a insulina, ou problemas com sua produção (VILLINES, 2019). Dentre outras funções, a insulina é um dos hormônios responsáveis pela regulação da quantidade de glicose na corrente sanguínea (glicemia), sua baixa disponibilidade pode levar a hiperglicemia (excesso de glicose no sangue).

Diabéticos possuem um processo de cicatrização mais lento que não portadores dessa doença, isso porque a hiperglicemia cascateia uma série de fenômenos que levam a este retardo, dentre eles: o sistema imunológico tem a resposta prejudicada, tornando feridas mais suscetíveis a infecções; e problemas circulatórios, o que faz com que menos nutrientes cheguem até a área danificada (LIMA, 2013).

A hiperglicemia também causa neuropatias (danos aos nervos), fenômeno que não retarda a cicatrização diretamente, porém leva à perda de sensibilidade (CS, 1990). Por sentir menos dor, o diabético se torna mais propenso a não notar que se machucou e feridas em regiões raramente observadas podem ficar tempo o suficiente sem serem percebidas para que infecções se instaurem no local e gerem complicações. Em situações graves, feridas que podem progredir a ponto de tornarem-se necessárias amputações. Nestes casos, é necessário acompanhamento médico para assegurar que o quadro destas feridas não se agrave a este ponto.

O tratamento envolve a limpeza e proteção da ferida, controle da glicemia e utilização de medicamentos. É importante que o médico responsável pelo tratamento acompanhe o desenvolvimento da ferida, fotografando-a e medindo sua área com certa periodicidade, pois a diminuição da área é considerada o principal indicativo de sucesso do tratamento. A *Wound Healing Society* afirma que pacientes que não apresentam diminuição de pelo menos 40% na área da ferida após quatro semanas de terapia, devem ter seu tratamento reavaliado e outras abordagens devem ser consideradas (SHEEHAN, 2003).

Um método bastante difundido e eficaz para a medida da área é sobrepor uma folha transparente na ferida e traçar nesta folha o seu contorno. Posteriormente, realiza-se a medida (cálculo) da área (KEAST, 2004). No entanto, é possível afirmar que o cálculo manual não fornece o valor exato da área da ferida, já que os contornos tendem a ser irregulares. Neste trabalho, apresenta-se o desenvolvimento e a avaliação de um aplicativo para *smartphone* que realiza a aquisição da imagem da folha contendo o contorno da ferida e o cálculo semi-automático da área.

1.1 CONTEXTO E JUSTIFICATIVA

Existem métodos que calculam a área diretamente a partir de uma fotografia da ferida. Uma referência de área conhecida é colocada no plano da ferida (ao lado da ferida) permitindo a obtenção da área dos pixeis da imagem. Assim, calcula-se a área da ferida contando o número de pixeis pertencentes à ferida. No entanto, estes métodos desconsideram a curvatura da superfície, o que resulta em uma medida subestimada da área. Já no método utilizado nesse trabalho, que se baseia no traçado do contorno da ferida em uma folha transparente, este erro é evitado, pois a folha está em contato direto com a ferida (KEAST, 2004).

Este trabalho foi realizado sob a consultoria do Dr. Adriano Mehl, que realiza procedimentos de medida de área de feridas frequentemente. Assim, a folha utilizada nesse trabalho para traçar o contorno da ferida é a mesma utilizada por ele. Trata-se de uma folha transparente feita de acetato, contendo uma área para a identificação do paciente (nome e outros dados) e um campo de $17cm$ de largura por $22cm$ de altura, contendo uma grade com quadrados de $1cm$ x $1cm$, sobre a qual é desenhado o contorno da ferida. Neste trabalho, denomina-se esta folha de *régua*.

A medida da área da ferida desenhada nesta folha pode ser obtida de duas maneiras: (i) contando o número de quadrados presentes dentro do contorno da ferida. (ii) Para medidas mais precisas a régua é escaneada, a imagem é transferida para um computador e, com a utilização de um *software* [1], seleciona-se o contorno da ferida com o mouse e aponta-se uma referência de tamanho conhecido na imagem. O programa calcula a área. Esse processo é demorado e exige um *scanner*.

O aplicativo para *smartphone* desenvolvido neste trabalho torna a obtenção da medida da área da ferida a partir do contorno desenhado na régua mais precisa se comparada ao método (i) e mais prática e rápida se comparada ao método (ii).

Podemos tornar o processo de obtenção de uma medida precisa muito mais rápido e sem a utilização de um *scanner* fazendo uma aplicação para *smartphone*. Por utilizar um contorno que leva em consideração a curvatura da superfície do membro lesionado do paciente, o método pode ser mais preciso que os que utilizam fotos feitas diretamente das lesões.

1.2 OBJETIVO GERAL

Desenvolver um aplicativo para dispositivos móveis baseados em *Android* para medir a área de uma ferida, a partir da foto da régua com o desenho do contorno da ferida.

1.3 OBJETIVOS ESPECÍFICOS

- Desenvolver o algoritmo em ambiente de prototipagem para o processamento da imagem.
- Traduzir o algoritmo desenvolvido para um código em linguagem orientada a objeto.
- Portar o código para execução no sistema operacional *Android*.
- Realizar experimentos para aferir a precisão das medidas obtidas pela aplicação.

[1] Digimizer, <https://www.digimizer.com/>

1.4 ESTRUTURA DO DOCUMENTO

Este relatório de Trabalho de Conclusão de Curso está estruturado da seguinte forma: primeiramente os dados gerais sobre a problemática encontrada são apresentados, discorrendo sobre feridas e diabetes. Em seguida, descreve-se o algoritmo desenvolvido para a medida da área da ferida e, posteriormente, a implementação deste algoritmo para a plataforma *Android*. Finalmente, apresentam-se os resultados de testes de usabilidade e precisão na medição da área.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 FERIDAS

A pele humana é considerada o maior órgão do corpo, possuindo até sete camadas de tecido ectodermal. Ela oferece proteção contra patógenos, o sentido do tato, termorregulação, controle de evaporação, armazenamento de água e lipídios etc. Uma ferida acontece quando as características anatômicas da pele são afetadas (BRASIL, 2013).

2.1.1 CAUSAS DE FERIDAS

As feridas ocorrem em seres humanos por razões como as abaixo (OSTERD, 1998):

- Trauma inicial ou repetitivo
- Queimaduras térmicas ou químicas
- Mordidas ou picadas de animais
- Pressão
- Comprometimento vascular (arterial, venoso ou linfático)
- Imunodeficiência
- Tumores malignos
- Doenças em tecidos conectivos
- Deficiências nutricionais
- Problemas psico-sociais
- Efeitos adversos de medicações
- Doenças metabólicas, como diabetes, foco desse trabalho

2.1.2 FASES DA CICATRIZAÇÃO DE FERIDAS

Tipicamente a cicatrização ocorre em quatro grandes etapas, descritas nas próximas sessões.

2.1.2.1 HOMEOSTASE

Nesta fase, que começa logo após a ferida ser causada – a não ser que o paciente tenha problemas de coagulação – plaquetas começam a aderir a área afetada; tais plaquetas convertem-se numa substância amorfa, e ativam substâncias como a fibrina (oriunda da glicoproteína fibrinogênio) que serve como uma rede e faz as plaquetas aderirem entre si formando uma espécie de coágulo.

As plaquetas também são responsáveis pela liberação dos fatores de crescimento que irão permitir o acontecimento das próximas etapas de cicatrização, ao permitir o recrutamento de neutrófilos, monócitos e fibroblastos, a estimulação de células epiteliais (OSTERD, 1998).

2.1.2.2 INFLAMAÇÃO

Durante esta fase, que dura até dias, as células mortas ou danificadas são removidas junto com patógenos e resíduos. Nesta etapa o tecido afetado acaba recebendo plasma e neutrófilos, sendo estas células responsáveis pelo processo de fagocitose. Após o pico de neutrófilos, os macrófitos entram em cena. É um processo caracterizado por edema, eritema, calor e possível dor (OSTERD, 1998).

2.1.2.3 PROLIFERAÇÃO

É o processo no qual o novo tecido começa a crescer, ocorrendo por volta de 4 dias até pelo menos o dia 21 após a ocorrência da ferida; nesta fase ocorre a angiogênese (criação de novos vasos sanguíneos), contração das bordas da ferida em direção ao seu centro e a epitelização do tecido a fim de preenchê-lo (OSTERD, 1998).

2.1.2.4 MATURAÇÃO

O tecido novo criado adquire força e flexibilidade. Ocorre o realinhamento das fibras de colágeno. As principais células desse processo são os fibroblastos. Devido ao fato de o processo de maturação ser demorado, podendo durar até 2 anos, é necessário manter atenção aos fatores causadores da ferida para não comprometer o processo (OSTERD, 1998).

2.2 PROCESSO DE CURA DE FERIDAS

O processo de cicatrização define o tipo de ferida: em casos em que ele ocorre de forma natural, tem-se uma ferida **aguda**; caso contrário, numa ferida **crônica**, os processos de cicatrização não ocorrem de forma esperada. Um dos resultados pode ser a não retomada das características originais do tecido ferido. Há ainda de se considerar a possibilidade de cura, ou não, das feridas, o que é mostrado no quadro 1.

2.3 DIABETES

Diabetes é uma doença crônica séria que resulta em alta glicose sanguínea devido aos problemas com a produção ou reconhecimento da insulina, um hormônio que regula o nível de glicose no sangue (BRASIL, 2013).

O diabetes tipo 1 é caracterizado por um deficit na produção da insulina. No momento, a causa da doença não é conhecida de forma que ela não é uma doença prevenível (WHO, 2016).

Por outro lado, o diabetes tipo 2 é relacionada ao não reconhecimento correto pelos receptores da insulina no corpo. É o tipo de diabetes com maior número de pacientes pelo mundo. Os sintomas podem ser semelhantes aos do tipo 1, mas podem ser mais brandos ou

Quadro 1 – Curabilidade de feridas.

Tipos de feridas	**Características**	**Exemplos**
Curável	Causas e co-fatores que podem interferir com a cura foram removidos. Cura de ferida acontece de forma previsível. Pode ser aguda ou crônica.	Úlcera de pressão na qual pressão e outros fatores são gerenciados.
Gerenciável	Causas e cofatores que interferem com o tratamento são removíveis, porém não foram removidos por conta de fatores sistemáticos do paciente.	Úlcera de pressão na qual a pressão não é gerenciada.
Incurável	Causas e cofatores que podem interferir com a cura não podem ser removidos.	Pé gangrenoso ou ferida maligna.

Fonte: Osterd (1998)

até ausentes. Por causa disso, o diagnóstico pode ser retardado em vários anos, quando as complicações já são graves. (WHO, 2016)

2.4 FERIDAS EM DIABÉTICOS

O diabetes mellitus (DM) tipo 2 é um distúrbio metabólico de alta mortalidade causado por hiperglicemia e mudanças no metabolismo de sustâncias por conta de falhas, sejam na secreção ou na ação da insulina. (BRASIL, 2013).

DM é uma doença de grande expressão no cenário mundial, especialmente em países emergentes, sendo o tipo de diabetes com mais afetados no mundo. No caso da América Central e do Sul, a prevalência da DM foi estimada em 40 milhões para o ano de 2030 pela Federação Internacional de Diabetes. É esperado, ainda, que o Brasil passe da oitava posição (prevalência de 4,6%) para a sexta posição, 11,3%, em 2030. Quanto às complicações relacionadas a feridas, podem ser problemas de cicatrização, úlceras crônicas e até mesmo amputações de membros (BRASIL, 2013).

No caso da cicatrização de feridas em pacientes diabéticos, não se costuma observar um processo ordenado em nenhuma de suas fases características (OKONKWO, 2019). As ulcerações provocadas pelo diabetes são presentes como feridas doloridas com desintegração da epiderme, derme e em diversos casos, do tecido subcutâneo. A neuropatia diabética é o problema mais comum dos pacientes diabéticos e afeta o sistema nervoso periférico. Ela pode se manifestar de maneira assintomática, o que prejudica a sensibilidade que confere proteção contra lesões nas extremidades do corpo (exemplo na Figura 1) (BRASIL, 2013). De forma geral, por volta de 25% dos pacientes diabéticos terão problemas de cicatrização em feridas nos membros inferiores em algum momento da vida, sendo que a taxa de amputações de grande escala, mundialmente, é de 0,5 a 5 por 1000 afetados por diabetes (JEFFCOATE, 2003).

Figura 1 – Exemplo de feridas nas extremidades do corpo em diabéticos.

Fonte: <https://www.ufrgs.br/lidia-diabetes/2018/04/28/pe-diabetico/>

2.5 MEDIÇÃO DE FERIDAS

A cada consulta de um paciente diabético com feridas, as mesmas devem ser medidas. As medições são parte importante do gerenciamento de feridas, pois fornecem subsídios para se definir e acompanhar o tratamento. Além disso, é sabido que as 4 primeiras semanas de acompanhamento são decisivas no sentido de apontar o sucesso ou não do tratamento do paciente (SHEEHAN, 2003), uma vez que os afetados que não respondam às técnicas convencionais podem precisar de tratamento prolongado.

2.5.1 PARÂMETROS DIMENSIONAIS DE FERIDAS

Diferentes parâmetros podem ser medidos em uma ferida, conforme descrito a seguir (KEAST, 2004):

- Área: a medição repetida de áreas de feridas pode monitorar e predizer o avanço no tratamento, sendo que desenhar o contorno da ferida sobre uma folha transparente proporciona uma medição válida. Além da área, simplesmente a largura e o comprimento da ferida podem trazer certas informações importantes sobre a sua condição. No entanto, é sabido que estas técnicas comumente superestimam o tamanho da ferida, sendo a medição de grandes feridas, ou aquelas de formato irregular, particularmente afetada.
- Profundidade: Pode ser medida, por exemplo, com uma haste de algodão devidamente esterilizada inserida na parte mais funda do ferimento, e está correlacionada com a gravidade do prejuízo ao tecido.
- Volume: Pode ser útil em situações de feridas profundas, nas quais a cicatrização costuma acontecer da base ao topo, de forma que a área acaba não tendo alterações. Entretanto, no caso geral, a medição de volume costuma ser contraindicada devido à falta de métodos padronizados para sua aferição. Diversos problemas estão ligados a esse tipo de medição,

por exemplo, dificuldade de definir as bordas da ferida; mudanças de volume resultantes da simples mudança do posicionamento; impacto da curvatura do corpo etc.

2.5.2 MEDIÇÕES APROXIMADAS DE FERIDAS

É possível utilizar um retângulo para aproximar a área de uma ferida, a partir da medição do seu comprimento e largura (maiores cordas) (equação 1). No entanto, embora prático, este método não é indicado pois pode superestimar em até 44% a área. Caso não se disponha de sistemas para a medida da área que considerem o formato particular da ferida, a área que melhor aproxima grande parte das feridas planas é a de uma elipse (equação 2), que tende a superestimar em 13% a área da ferida (KEAST, 2004).

$$A_{rect} = Comprimento \times Largura \tag{1}$$

$$A_{elipse} = Comprimento \times Largura \times \pi/4 \tag{2}$$

2.6 SOLUÇÕES EXISTENTES PARA A MEDIÇÃO DA ÁREA DE FERIDAS

Técnicas para medir feridas no ambiente médico são diversas e atualmente costumam envolver fotografia digital por conta de sua praticidade e relativo baixo custo. No entanto, há de se considerar que o processo não é trivial, devido ao fato de haver problemas na captação da imagem (contraste, resolução ou iluminação inadequada etc), pelo fato de feridas muitas vezes terem um formato não plano (como no caso de membros) e pela dificuldade de se delimitar automaticamente as fronteiras entre a ferida e a pele sadia (segmentação).

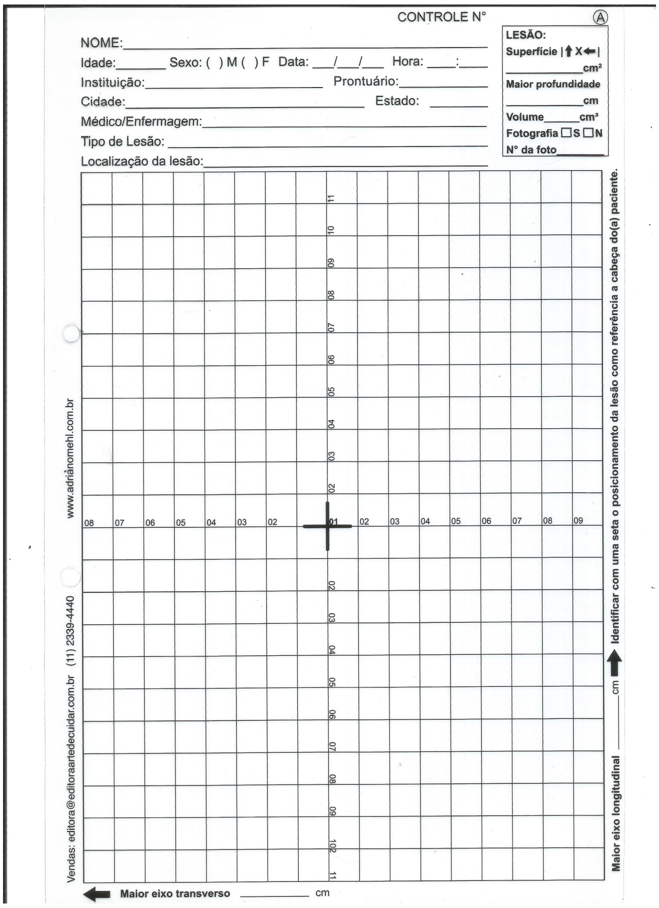
2.6.1 RÉGUA UTILIZADA NESTE TRABALHO

A régua é mostrada na Figura 2. É composta por uma folha transparente feita de acetato, contendo uma área para a identificação do paciente (nome e outros dados) e um campo de $17cm$ de largura por $22cm$ de altura, contendo uma grade com quadrados de $1cm$ x $1cm$, sobre a qual é desenhado o contorno da ferida. É importante mencionar que esta régua não entra em contato direto com a ferida, pois utiliza-se uma película de acetato transparente entre a ferida e a régua. O objetivo desta película é evitar a contaminação do paciente, já que ela é esterilizada, e também da própria régua com o contorno, já que esta será utilizada para a medida da área da ferida e armazenada no prontuário.

A régua para medir feridas foi projetada e desenvolvida pelo médico Adriano Antônio Mehl[1] que protocolou o pedido de registro junto ao INPI (Instituto Nacional da Propriedade Industrial) e teve a Concessão do Registro publicada na Revista da Propriedade Industrial-RPI 2362 página 260 em 12 de abril de 2016 sob o número BR 30 2012 004137-6.

[1] <http://lattes.cnpq.br/7095979310782838>

Figura 2 – Régua para o desenho do contorno da ferida.



Fonte: Autoria própria

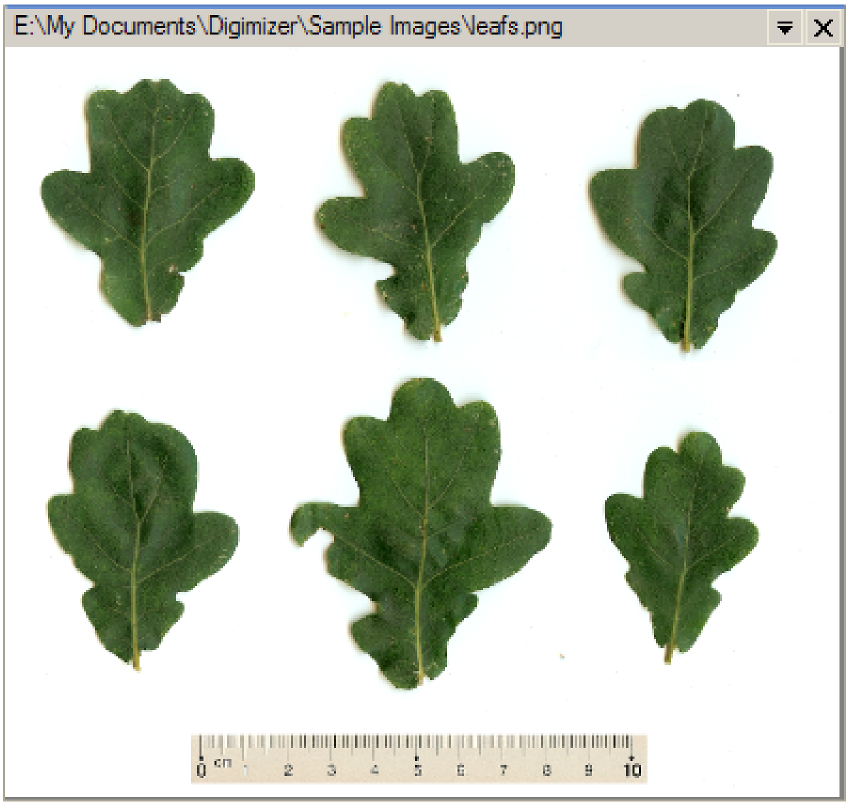
2.6.2 DIGIMIZER

Uma solução utilizada para medir a área de feridas digitalmente é analisar a imagem do acetato escaneada com um programa que sirva pra medir áreas. Um dos mais utilizados é o Digimizer, um *software* pago para *Windows* destinado à detecção de figuras e suas características. Neste processo de medição, pressupõe-se que exista um referencial de escala, como uma régua ilustrada na figura 3, para calibrar a medição.

O processo de medição com o Digimizer de forma semi-automática consiste nas seguintes etapas (mostradas na Figura 3):

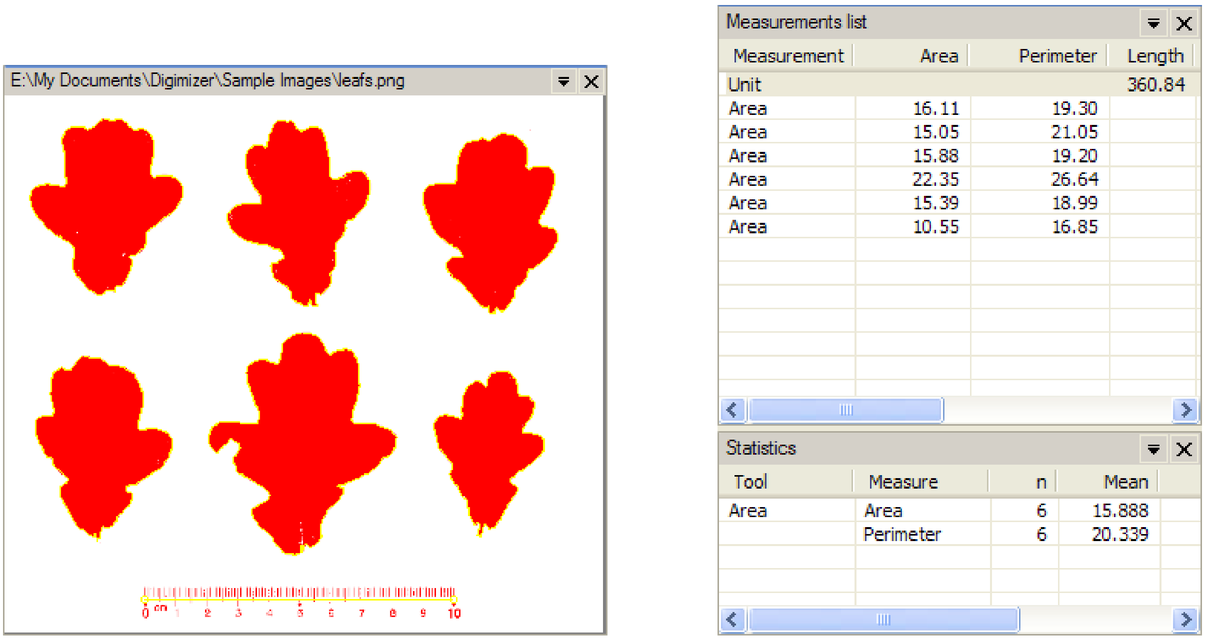
- Carregamento da imagem no computador.
- Correção do contraste.
- Seleção da régua de referência.
- Binarização da imagem e seleção do limiar.
- Obtenção dos resultados de perímetro e área da imagem (Figura 4).

Figura 3 – Exemplo de medição utilizando o *software* Digimizer.



Fonte: <https://www.digimizer.com/manual/example-leafs.php>

Figura 4 – Resultado da medição utilizando o *software* Digimizer.

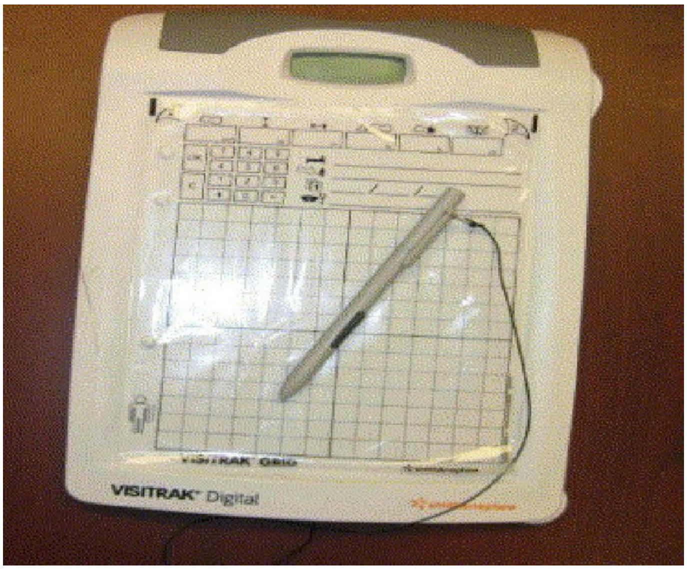


Fonte: <https://www.digimizer.com/manual/example-leafs.php>

2.6.3 VISITRAK

O Visitrak (Figura 5) é um equipamento para medição de feridas composto de um *tablet* com uma caneta especial e de uma régua, de uso único, composta de três camadas, que serve para desenhar um contorno sobre a área de interesse com um marcador permanente. O sistema permite a medida das dimensões da ferida (área, comprimento, largura). O acetato é então colocado sobre o *tablet* e a caneta especial deve ser utilizada para retraçar o contorno da ferida. O *tablet* converte o contorno desenhado em uma área de medida e pode aferir também a mudança percentual da área com relação às últimas medições.

Figura 5 – Medição de ferida usando Visitrak. Na imagem, observa-se o acetato no qual desenha-se o contorno da ferida, posicionado sobre o *tablet*.

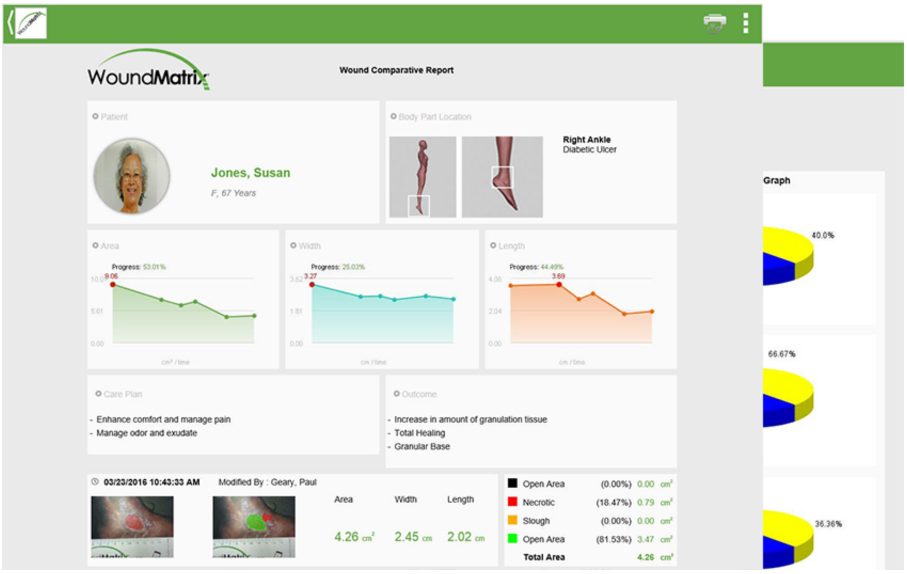


Fonte: <https://www.researchgate.net/figure/Visitrak-digital-device-Intervention-A-brief-questionnaire-was-introduced-to-obtain-each_fig3_258513247>

2.6.4 WOUNDMATRIX

O WoundMatrix é uma plataforma fundada em 2000 focada em tecnologia de documentação de feridas. Ela permite aos usuários capturar, medir e fazer *upload* de fotos tiradas por meio de um *app* para uma interface *web* (figura 6). Nela também é possível manter controle simultâneo de diversos pacientes.

Figura 6 – Exemplo de uso da plataforma WoundMatrix.

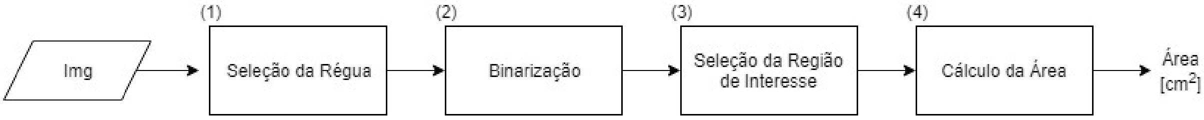


Fonte: <http://www.woundmatrix.com/woundmatrix.aspx>

3 ALGORITMO PARA MEDIÇÃO DA ÁREA DE FERIDAS

Após a traçar o contorno da ferida e descartar a película que entrou em contato com a pele do paciente, o profissional da saúde deve colocar a régua sobre uma superfície de cor clara e adquirir uma imagem. Esta imagem é tratada com o algoritmo (Figura 7) descrito na sequência deste capítulo.

Figura 7 – Diagrama em blocos do algoritmo.



Fonte: Autoria própria

3.1 SELEÇÃO DA GRADE DA RÉGUA

A imagem da régua contendo a grade na qual foi desenhado o contorno da ferida pode já estar presente na galeria do *smartphone* ou pode ser adquirida pela câmera. Esta imagem é submetida à biblioteca *SmartCropper* [1], que sugere as posições dos quatro vértices da grade, conforme o algoritmo do fluxograma da Figura 8. O bloco chamado *binariza com valores de gaussianblurValue[j] e cannyValue[i]* realiza as operações descritas no fluxograma da Figura 9.

3.1.1 FILTRO GAUSSIANO

No processamento de imagens, é comum a utilização de filtros espaciais baseados na convolução. Supondo uma imagem em níveis de cinza, uma das formas de filtrar é alterar o valor de um pixel com base no valor dos pixeis vizinhos. Multiplicam-se os valores dos pixeis vizinhos do pixel alvo por coeficientes e soma-se os resultados para obter o novo valor, conforme ilustrado na Figura 10.

O filtro passa-baixas Gaussiano é utilizado para suavizar o ruído na imagem. A equação 3 permite a construção de uma máscara (*kernel*) Gaussiana 2D.

$$h(x,y) = \exp \frac{-(x^2 + y^2)}{2\sigma^2} \tag{3}$$

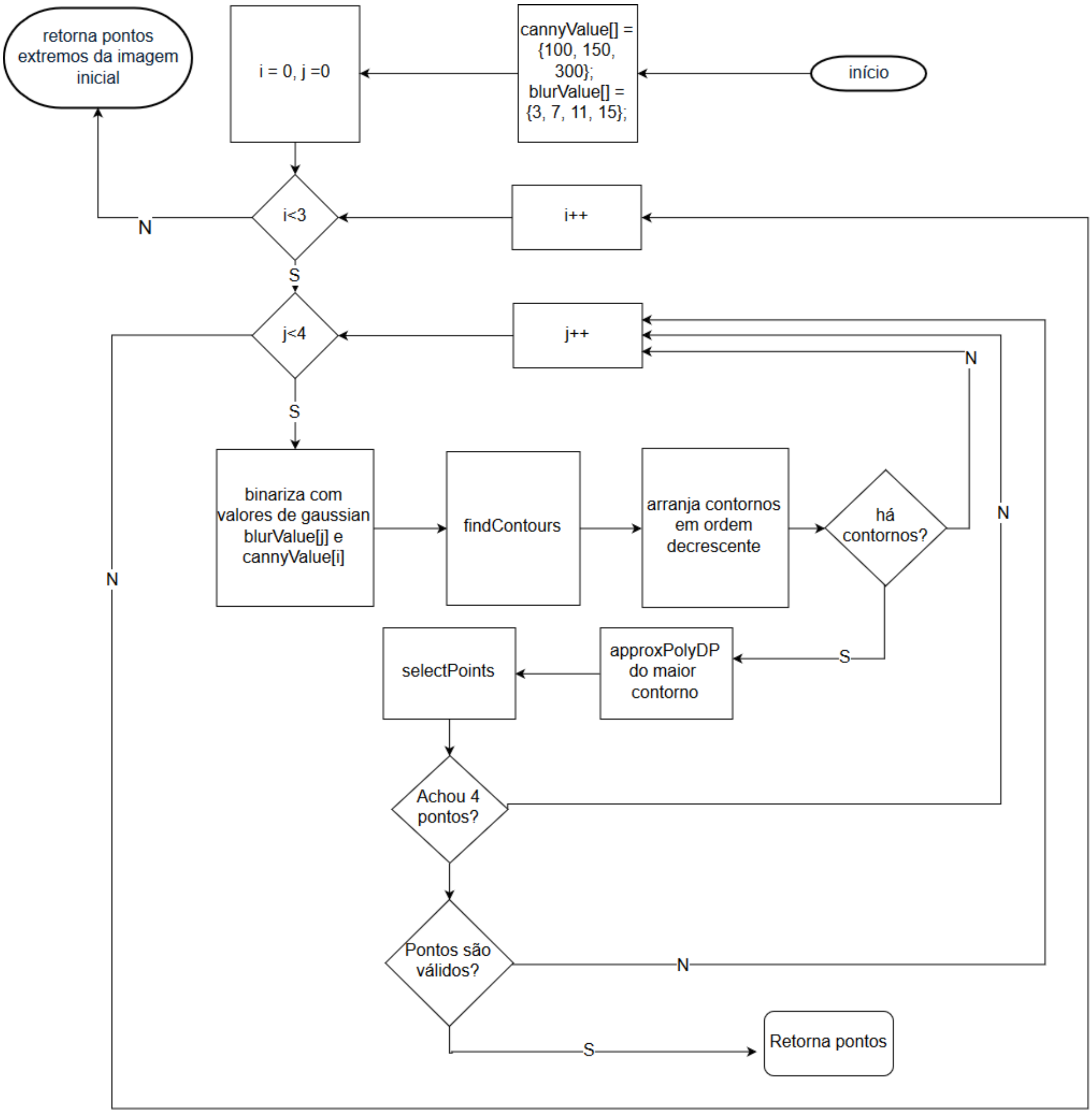
O parâmetro σ (desvio padrão) é ajustável e controla a abertura da Gaussiana. Quanto maior o valor de σ, mais pronunciada é a suavização.

3.1.2 DETECÇÃO DE BORDAS DE CANNY (CANNY EDGE DETECTION)

As etapas a seguir descrevem o algoritmo *Canny Edge Detection* (CANNY, 1986):

[1] <https://github.com/pqpo/SmartCropper/>

Figura 8 – Fluxograma do algoritmo de seleção automática dos quatro vértices da grade. O algoritmo do bloco *binariza com valores de gaussianblurValue e cannyValue* é descrito na Figura 9.

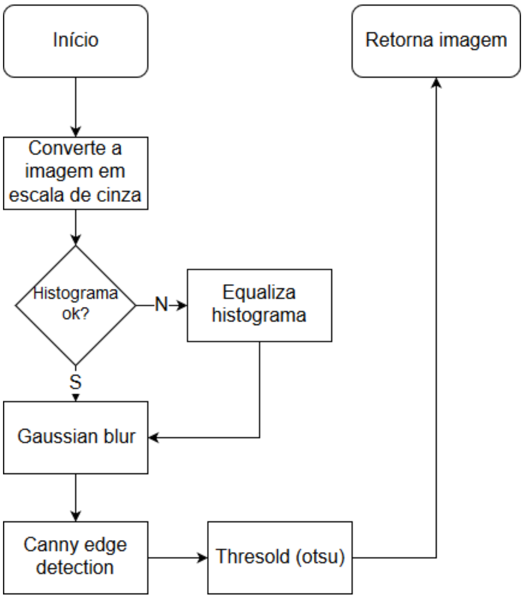


Fonte: Autoria própria

1. Aplicar filtro Gaussiano de 5x5 (como na sub-sessão 3.1.1), pelo fato de a detecção de bordas ser sensível ao ruído.
2. Achar os gradientes de intensidade da imagem por meio de um *kernel* de Sobel tanto na direção horizontal quanto na vertical. Cada pixel de cada uma destas duas imagens (resultados do filtro vertical e horizontal) têm as características de gradiente (equação de borda 4) e ângulo (5).

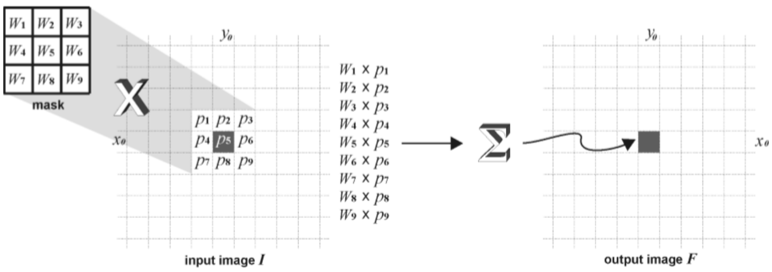
$$G = \sqrt{G_x^2 + G_y^2} \tag{4}$$

Figura 9 – Fluxograma do algoritmo de binarização utilizado pela função de seleção automática dos quatro vértices da grade, apresentado na Figura 8, no bloco denominado *binariza com valores gaussianblurValue e cannyValue*



Fonte: Autoria própria

Figura 10 – Operação de convolução em imagens. Exemplo para uma máscara de convolução 3x3.



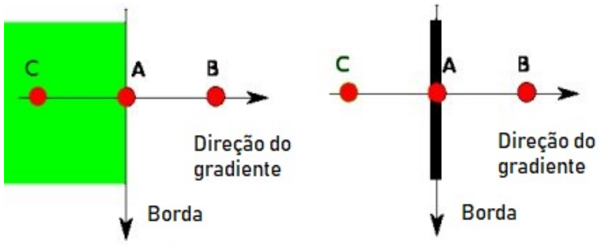
Fonte: Marques, Oge - "PRACTICAL IMAGE AND VIDEO PROCESSING UNSING MATLAB", Figura 10.1

$$\theta = \arctan\left(G_y/G_x\right) \tag{5}$$

A direção do gradiente é sempre perpendicular às bordas, sendo arredondada para um valor representando direção vertical, horizontal ou duas diagonais.

3. Um método de afinamento (supressão de não-máximos) é usado. Consiste em primeiro comparar a força da borda do pixel atual com a força do pixel nas direções de gradiente positivo e negativo. Caso a força do pixel atual seja a maior comparados com os outros pixeis na máscara com a mesma direção, o valor é mantido. Caso contrário, ele é eliminado. Por exemplo, na Figura 11, o ponto A está na borda da direção vertical. O gradiente

Figura 11 – Afinamento do Canny

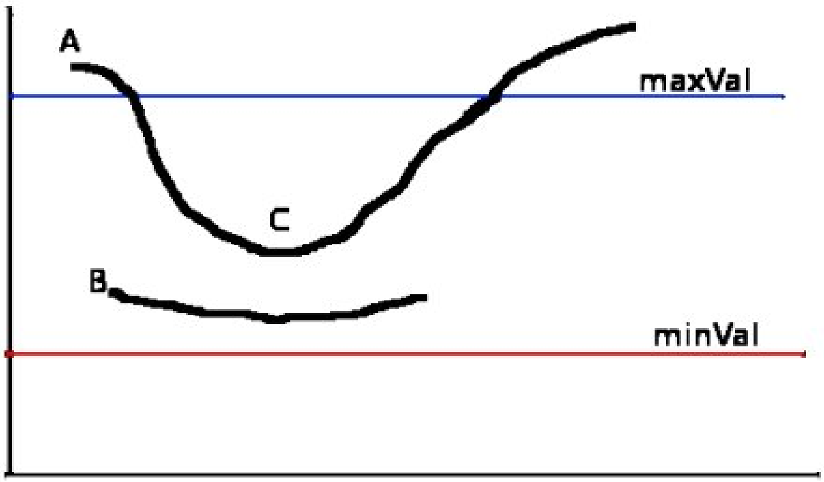


Fonte: Adaptado de <https://docs.opencv.org/3.1.0/da/d22/tutorial_py_canny.html>

de direção é normal à borda. Os pontos B e C são gradientes de direção. O ponto A é comparado com os pontos B e C para verificar se A representa um máximo local. Em caso positivo, ele é considerado para a próxima etapa, caso contrário, é eliminado.

4. Aplicação de limiar duplo e remoção dos contornos fracos por meio de histerese.

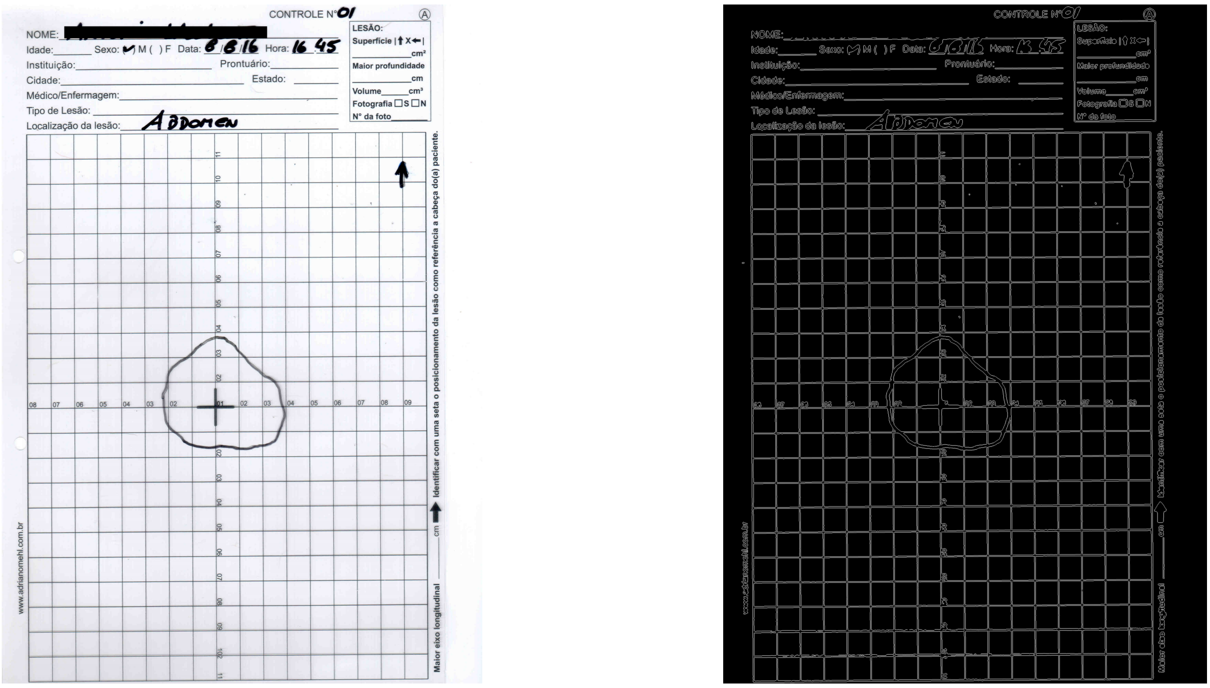
Figura 12 – Gráfico de histerese



Fonte: <https://docs.opencv.org/3.1.0/da/d22/tutorial_py_canny.html>

Conforme mostrado na Figura 12, escolhem-se dois valores de patamar, $maxVal$ e $minVal$. Quaisquer valores de contornos com gradiente maior que $maxVal$ são arbitrados como contornos e os menores que $minVal$ são automaticamente descartados. Aqueles que se encontram entre os patamares são considerados contornos ou não baseados na sua conectividade. Caso estejam conectados, são considerados partes dos contornos. Caso contrário, são descartados.

Esta saída do algoritmo de Canny (mostrada na figura 13) é então submetida a um processo denominado *border following* (SUZUKI, 1985) (função *findContours* do OpenCV).



(a) (b)

Figura 13 – Exemplo de resultado do algoritmo de detecção de bordas de Canny. À esquerda a imagem original e à direita a imagem com as bordas detectadas.

3.1.3 FUNÇÃO FINDCONTOURS DO OPENCV

No contexto do OpenCV, contornos são definidos como uma curva contendo todos os pontos de uma matriz com a mesma intensidade ou cor (o que justifica o processo definido em 3.1.2). A operação *findContours* toma uma matriz e retorna uma lista de vetores contendo os contornos encontrados na imagem por meio de um algoritmo denominado *border following*. Os contornos são ordenados de forma decrescente de quantidade de pixeis. A figura 14 mostra os seis maiores contornos da imagem na Figura 13 em diferentes cores (OPENCV, 2014).

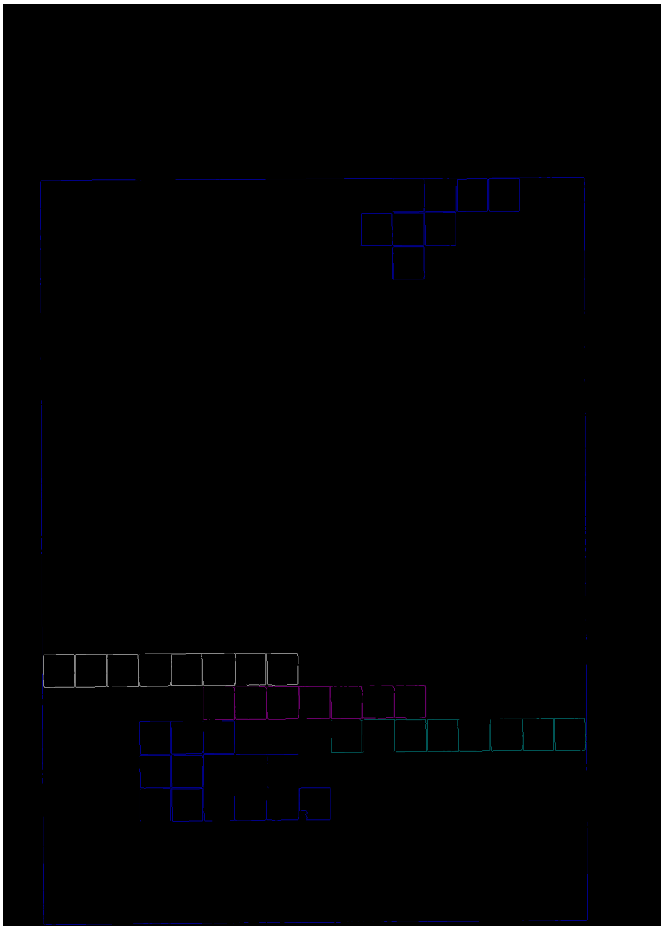
3.1.4 FUNÇÃO APPROXPOLYDP DO OPENCV

Esta função do OpenCV aproxima a curva poligonal fechada obtida anteriormente em um polígono com menos vértices, sendo a precisão desta conversão ajustável. A imagem da Figura 15 apresenta exemplos de saída da função *approxPolyDP*.

O processo de binarização (fluxograma da Figura 9) imagem é executado 12 vezes, utilizando quatro valores de limiar de *blur* gaussiano $3, 7, 11, 15$, e três valores de histerese *canny detection* $100, 150, 300$. Enquanto o processo de obtenção do maior contorno não for satisfatório o processo continua. O maior contorno é aquele que atende ao seguinte critério: o contorno envolve uma área maior que $1/20$ da área total. Na Figura 15 à direita é apresentada a detecção do retângulo principal da imagem.

Neste caso, o vetor contendo estes contornos encontrados é iterado de forma a tentar

Figura 14 – Exemplo de saída da função *findContours* do OpenCV para a imagem de entrada da Figura 13.



Fonte: Autoria própria

encontrar 4 pontos que representem os vértices do retângulo do centro da régua. Os 4 pontos com coordenadas mais extremas são os vértices do retângulo.

Caso contrário, os pontos retornados são os 4 extremos da imagem original, demonstrando que o processo falhou.

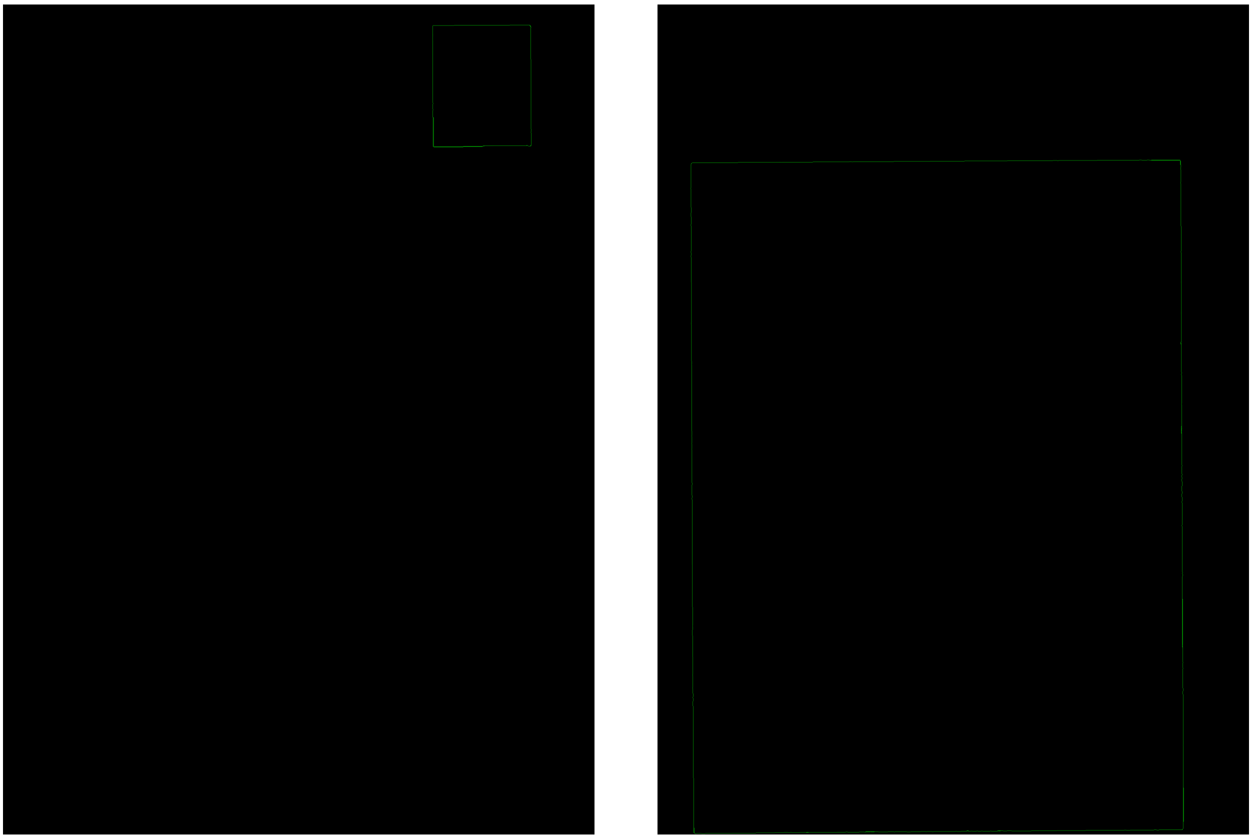
3.1.5 GETPERSPECTIVETRANSFORM E WARPPERSPECTIVE

Caso o processo anterior não tenha determinado as quatro extremidades da grade da régua de forma satisfatória, o usuário pode selecionar manualmente estas quatro posições.

A partir destes quatro pontos é necessário realizar o ajuste de perspectiva da grade da régua. Isto é realizado utilizando-se as funções *getPerspectiveTransform* e *warpPerspective* do OpenCV.

Estas funções não mudam o conteúdo da imagem, mas sim deformam a matriz de pixeis de forma a remapeá-la para um plano paralelo ao do sensor de imagem. Para evitar problemas de amostragem, o mapeamento é realizado na ordem inversa, ou seja, do destino à origem. Para cada pixel (x,y) da imagem de destino, as funções computam o pixel original na

Figura 15 – Exemplo de resultado da função *approxPolyDP*. À esquerda um exemplo de falha na seleção do maior contorno. À direita, mostra-se um exemplo de sucesso ao detectar o maior contorno



Fonte: Autoria própria

imagem de entrada e copia-se o valor do pixel (equação 6) (OPENCV, 2014).

$$\mathtt{dst}(x, y) = \mathtt{src}(f_x(x, y), f_y(x, y)) \tag{6}$$

3.1.5.1 GETPERSPECTIVETRANSFORM

Feita a devida correção manual (caso necessária) na seleção das extremidades, obtêm-se uma matriz com a função *getPerspectiveTransform*, que cria a chamada transformada de perspectiva (vide equação 7) entre os quatro pontos selecionados e os quatro pontos da nova imagem a ser criada.

$$\begin{bmatrix} t_i x'_i \\ t_i y'_i \\ t_i \end{bmatrix} = \mathtt{map\_matrix} \cdot \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \tag{7}$$

onde a matriz de destino é a da equação 8.

$$dst(i) = (x'_i, y'_i), src(i) = (x_i, y_i), i = 0, 1, 2, 3 \tag{8}$$

3.1.5.2 WARPPERSPECTIVE

A matriz obtida na equação 8 é utilizada como parâmetro M da transformação geométrica *Warp Perspective* via *OpenCV*, que cria uma nova imagem com a perspectiva ajustada. Cada ponto $P(x,y)$ da matriz de resultado é descrito pela equação 9.

$$\texttt{dst}(x, y) = \texttt{src}\left(\frac{M_{11}x + M_{12}y + M_{13}}{M_{31}x + M_{32}y + M_{33}}, \frac{M_{21}x + M_{22}y + M_{23}}{M_{31}x + M_{32}y + M_{33}}\right) \tag{9}$$

3.2 BINARIZAÇÃO

Os pixeis de uma imagem binária só apresentam dois estados, 0 e 1. Esta etapa transforma a imagem obtida após a seleção da grade da régua, até em então colorida, em uma imagem binária. Esta etapa é feita para separar o fundo da imagem (em branco na Figura 17) das linhas da grade e o contorno da ferida (em preto na Figura 17).

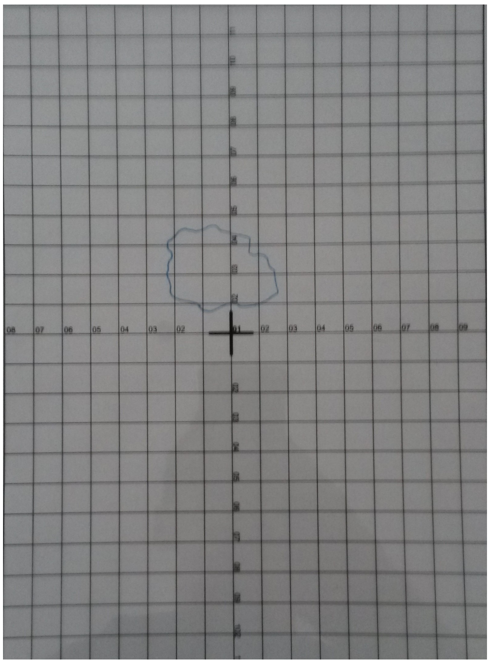
A decisão de transformar um pixel em preto ou branco é feita comparando a intensidade de cada pixel a um limiar, sendo intensidade a medida do quão claro é o pixel. Para obter a intensidade dos pixeis de uma imagem colorida basta transforma-la em uma imagem em escala de cinza. Se a intensidade de um pixel for inferior ao limiar, o pixel resultante será de uma cor, se for superior será da outra. Um pixel de uma imagem colorida geralmente é representado por três valores numéricos, num sistema conhecido como RGB, um representando a intensidade em verde, G, outro em azul, B e outro em vermelho, R. A equação 10 demonstra como calcular o valor da intensidade do pixel, P. Na Figura 16 observa-se o resultado da transformação.

$$P(R,G,B) = 0{,}299R + 0{,}587G + 0{,}114B \tag{10}$$

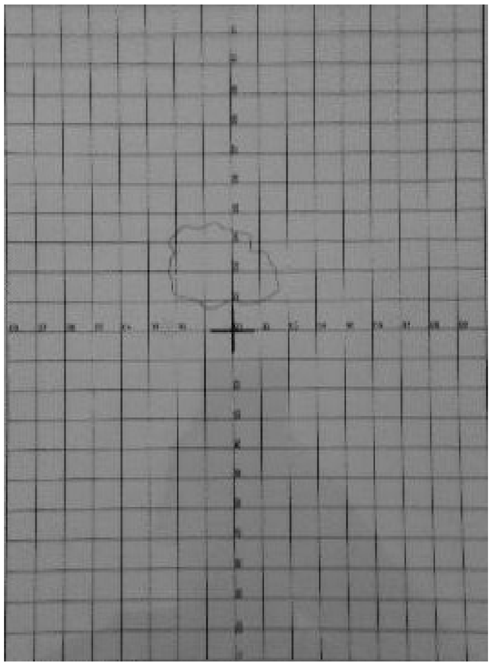
Se um único limiar é utilizado para todos os pixeis da imagem, variações na iluminação ao longo da imagem (sombras) podem gerar resultados não desejados, como pode ser observado na imagem da esquerda da Figura 17.

Para evitar os problemas causados por variações na iluminação ao longo da imagem, o algoritmo "*Adaptive Thresholding using the Integral Image*" (BRADLEY, 2007) é utilizado. Este algoritmo consiste em escolher um limiar particular para cada pixel da imagem e é computacionalmente eficiente. Neste método, a escolha do limiar para um determinado pixel é feita calculando o valor médio dos pixeis em uma janela retangular em torno dele, podendo multiplicar este valor por um fator menor ou igual a um. Na imagem da direita da Figura 17 pode-se observar a eficácia do método.

A primeira etapa deste algoritmo é o cálculo da imagem integral a partir da imagem original, caso a imagem seja colorida é necessário transformá-la em uma imagem em escala de cinza. Representamos como $f(x,y)$ o valor armazenado em cada pixel da imagem em escala



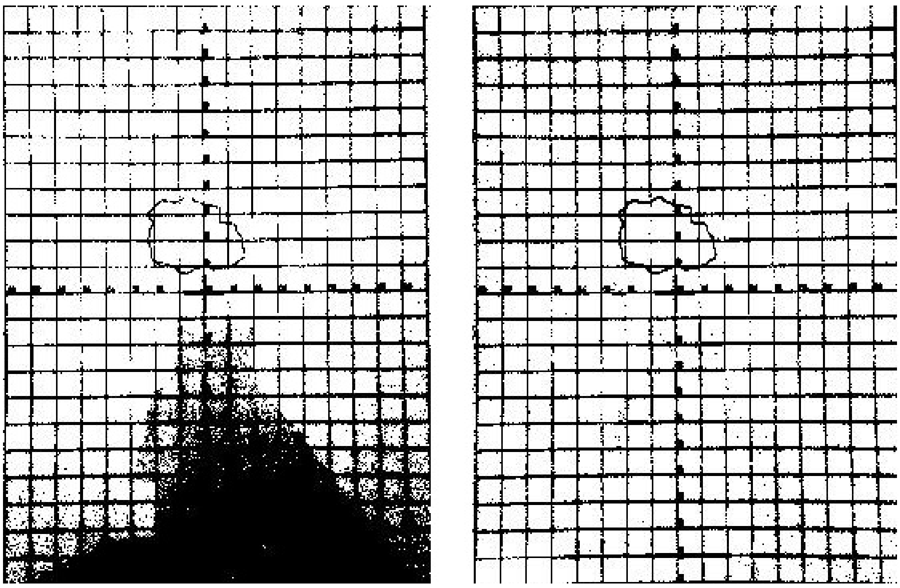
(a)



(b)

Figura 16 – Régua em cores e em tons de cinza.

Figura 17 – À esquerda, resultado da binarização usando um único limiar, à direita, resultado da binarização usando o método descrito por (BRADLEY, 2007)



Fonte: Autoria própria

de cinza. O valor armazenado em cada posição da imagem integral é a soma dos valores a esquerda e acima do correspondente na imagem em tons de cinza (vide equação 11).

$$I(x,y) = f(x,y) + I(x-1,y) + I(x,y-1) - I(x-1,y-1) \tag{11}$$

É possível calcular a soma dos valores de dentro de uma região retangular de uma imagem utilizando os valores das extremidades correspondentes da imagem integral (vide equação 12). Desta forma um número fixo de operações pode ser utilizado para calcular o valor do limiar para um pixel independentemente de quantos pixeis do entorno sejam considerados.

$$\sum_{x=x1}^{x2}\sum_{y=y1}^{y2} f(x,y) = I(x2,y2) + I(x1-1,y1-1) - I(x2,y1-1) - I(x1-1,y2) \qquad (12)$$

A quantidade de pixeis a serem considerados para o cálculo do limiar afeta o resultado da binarização.

3.3 SELEÇÃO DA REGIÃO DE INTERESSE

Esta etapa consiste em selecionar a região correspondente ao interior da ferida e retirar as linhas que cruzam a imagem. A seleção é manual (exibida na Figura 18) e todas as regiões brancas no interior do contorno da ferida devem ser selecionadas. Para que o usuário saiba quando ele selecionou todo o interior do contorno que representa a ferida, a região selecionada é "pintada".

A pintura do interior é feita utilizando uma adaptação do algoritmo 1 denominado "*Flood Fill*". A função deste algoritmo é selecionar uma área em uma imagem que tenha coloração similar. No caso da versão adaptada para este trabalho, ao selecionar um pixel branco, sua cor é alterada para preto e a função é aplicada nos pixeis ao redor.

Algoritmo 1: FloodFill

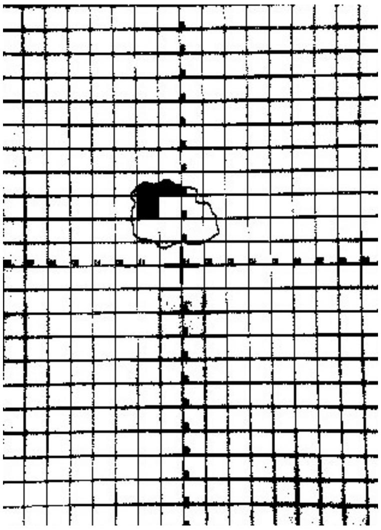
Input: a posição do pixel branco de início x y
if *pixel é preto* **then**
    return
end
FloodFill($x+1$, y)
FloodFill($x-1$, y)
FloodFill(x, $y+1$)
FloodFill(x, $y-1$)
return

Em paralelo, uma cópia da da imagem obtida após a binarização passa por um processo chamado "afinamento", onde todas os traços em uma imagem binária são afinados e passam a apresentar um pixel de "espessura". Este processo preserva a estrutura da imagem original, como "buracos" e "ramificações", o resultado é como o "esqueleto" da imagem (Figura 19).

O algoritmo de afinamento usado é o "*Zhang-Suen Thinning Algorithm*" (ZHANG, 1984), ele atua nos pixeis pretos da imagem, que representaremos com valor 1, transformando-os em brancos, valor 0, se certas condições forem presentes. Tomemos por P_1 o pixel a ser analisado e os ao seu redor P_2, P_3, P_4, P_5, P_6, P_7, P_8 e P_9 na ordem especificada na Figura 20.

Antes dos critérios, duas funções devem ser definidas: $A(P_1)$ é o número de transições de 0 para um nos pixeis em torno de P_1 seguindo a sequência P_2, P_3, P_4, P_5, P_6, P_7, P_8, P_9

Figura 18 – Seleção manual do interior da ferida.



Fonte: Autoria própria

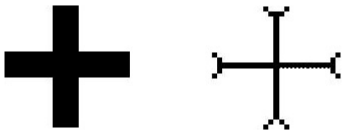


Figura 19 – Imagem original e imagem afinada.

Fonte: Autoria própria

Figura 20 – Ordem dos pixeis

P9	**P2**	**P3**
P8	**P1**	**P4**
P7	**P6**	**P5**

Fonte: <https: //nayefreza.wordpress.com/2013/05/11/zhang-suen-thinning-algorithm-java-implementation/>

e P_2, como na figura 20; $B(P_1)$ é a soma dos valores dos pixeis ao redor de P_1, ou seja, o número de pixeis pretos ao redor.

O passo 1 é analisar todos os pixeis pretos e verificar quais atendem as seguintes condições: $P_1 = 1$ e apresenta oito pixeis vizinhos (não é um pixel da borda da imagem; $2 \leq B(P_1) \leq 6$; $A(P_1) = 1$; P_2 ou P_4 ou P_6 é branco; P_4 ou P_6 ou P_8 é branco. Após todos os pixeis serem analisados os que atenderem as quatro condições deverão ser brancos.

No passo 2 todos os pixeis são testados novamente, há uma mudança na terceira e quarta condição apenas: $P_1 = 1$ e apresenta oito pixeis vizinhos (não é um pixel da borda da imagem; $2 \leq B(P_1) \leq 6$; $A(P_1) = 1$; P_2 ou P_4 ou P_8 é branco; P_2 ou P_6 ou P_8 é branco. Após todos os pixeis serem analisados os que atenderem as quatro condições deverão ser brancos.

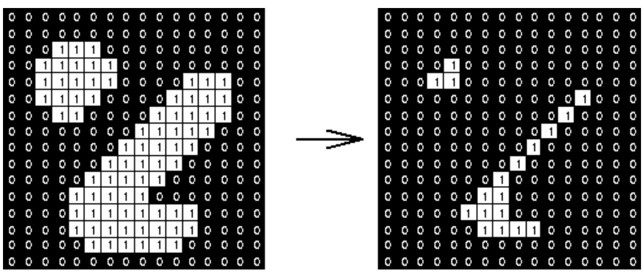
Enquanto pelo menos um pixel for transformado de preto para branco no passo 1 ou no 2, ambos passos devem ser refeitos.

A imagem na qual foi aplicado o afinamento agora tem aplicado o "*Flood Fill*" nas mesmas coordenadas selecionadas pelo usuário, resultando em uma imagem composta pela área da ferida com o interior da mesma cor e pelas linhas que cruzam a imagem apresentando apenas um pixel de espessura.

Como a medição da área requer contar os pixeis pertencentes à delimitação da ferida, é necessário eliminar as linhas que estão cruzando a imagem. Existem uma série de operações básicas que podem ser feitas com imagens binárias conhecidas como operações morfológicas, nelas usa-se um elemento estruturante para comparar com a vizinhança de cada pixel da imagem e decidir se seu valor será mudado ou não se um elemento estruturante para comparar com a vizinhança de cada pixel da imagem e decidir se seu valor será mudado ou não. Dentre estas operações encontramos a "erosão" como possível solução para retirar as linhas que estão cruzando a imagem após de afinamento.

A erosão retira pixeis das bordas dos elementos conectados (objetos de interesse). No caso da imagem binarizada da grade, o fundo é branco e os objetos de interesse pretos (vide figura 21). Essencialmente compara-se o um elemento estruturante (uma matriz como da figura 22) com o arredor do pixel analisado, se todos os elementos tiverem o mesmo valor, mantém-se o valor do pixel; caso o contrário, o pixel é transformado em um pixel branco. O resultado é que o interior dos objetos dentro da imagem são mantidos intactos e, as bordas, erodidas. Uma vez que as linhas que cruzam a imagem apresentam um pixel de espessura apenas, elas são todas retiradas pela erosão (figura 23). Quanto maior o elemento estruturante, mais pixeis da borda serão retirados, como só precisa-se corroer as linhas e a área de interesse é interior ao contorno da ferida, um elemento estruturante de dimensões 3x3 é utilizado.

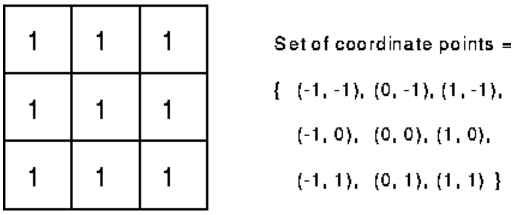
Figura 21 – Ação do erode



Fonte: <https://homepages.inf.ed.ac.uk/rbf/HIPR2/erode.htm>

O resultado é uma imagem que contém apenas o contorno da ferida preenchido, sem as linhas que antes cruzavam a imagem.

Figura 22 – Elemento estrutural 3x3



Fonte: <https://homepages.inf.ed.ac.uk/rbf/HIPR2/erode.htm>

Figura 23 – Linhas que cruzam a imagem afinadas e resultado após erosão

Fonte: Autoria Própria

3.4 CÁLCULO DA ÁREA

Como a área da região limitada pelo retângulo selecionado na primeira etapa (grade da régua) tem área conhecida, a área da ferida é obtida multiplicando a proporção entre número de pixeis da região da ferida e da imagem total, pela área que esta imagem representa. A grade tem dimensões 17 cm x 22 cm, área de 374 cm^2 , sendo A a área em cm^2 da região que representa a ferida, P o número total de pixeis da imagem e p o número de pixeis que preenchem a região correspondente a ferida, tem-se a relação da equação 13:

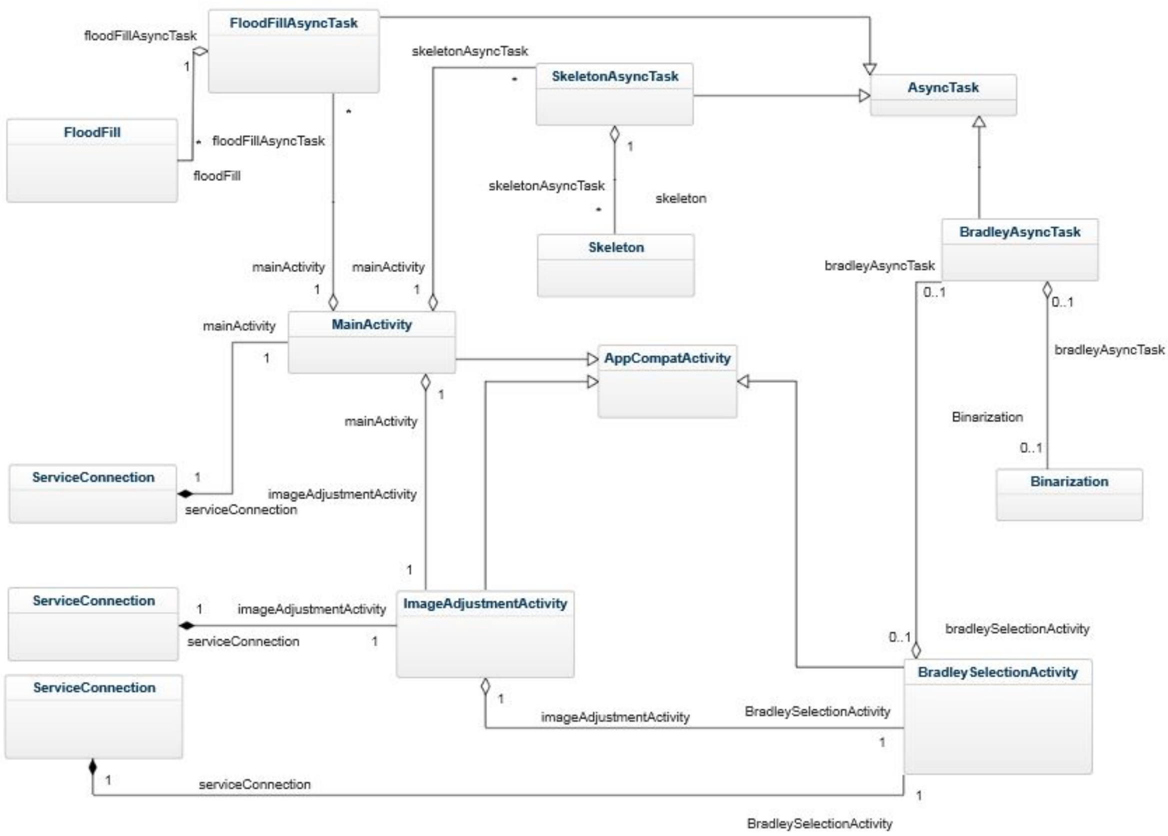
$$A = 374\frac{p}{P} \tag{13}$$

4 APLICATIVO ANDROID

Para se obter um aplicativo *Android* que cumprisse os requisitos de projeto planejados, primeiramente o algoritmo de processamento de imagem foi testado em ambiente de prototipagem. Uma vez que este código inicial foi concebido e testado, pensou-se numa modelagem adequada do algoritmo em código voltado para *Android*. Desta forma, foi então criado um código intermediário para *desktop* que permitisse *debug* rápido e, ao mesmo tempo, oferecesse uma transição escalável para o desenvolvimento móvel, além do suporte a características como processamento paralelo (*threads*) e orientação a objeto.

Foi escolhida a linguagem de programação Java, que é facilmente portável para o sistema operacional *Android* e permite os testes iniciais com a biblioteca OpenCV no *desktop*. A Figura 24 um diagrama UML das classes do aplicativo Android desenvolvido.

Figura 24 – Diagrama UML do aplicativo desenvolvido.



Fonte: Autoria própria

4.1 OPENCV 4.0

O OpenCV é uma biblioteca criada originalmente em C e C++, voltada para visão computacional, disponível para diversas plataformas, como Windows, Linux e Android.

Para a implementação da GUI no *desktop*, foi empregada o *toolkit Swing*.

A versão utilizada no aplicativo é a OpenCV 4.1.0[1], liberada ao público em abril de 2019. A desenvolvedora original do OpenCV foi a Intel como parte de uma iniciativa para viabilizar aplicações de *ray tracing*[2] e telas 3D. Os objetivos gerais do projeto OpenCV são promover a visão computacional multiplataforma com código otimizado ao criar uma infraestrutura comum que pode ser utilizada e reescrita por programadores de diversos contextos Bradski (2008).

4.2 GERENCIAMENTO E EXIBIÇÃO DE IMAGENS

O sistema operacional Android traz desafios relacionados ao carregamento de imagens (*bitmaps*) por conta dos seguintes problemas:

- *Bitmaps* são descompactados na memória do dispositivo, de forma que uma imagem oriunda da câmera de alta resolução (16 Megapixeis no caso do dispositivo usado nos testes, um Asus Zenfone 3) com 4 bytes de profundidade ocupa 64 Mb, restringindo drasticamente a memória para outros processos.
- Carregar imagens na *thread* de UI degrada severamente a *performance* do *app* e obriga a criação de *threads* especiais para lidar com os *bitmaps*.
- Caso o aplicativo carregue simultaneamente diversas imagens, como é o caso deste aplicativo, é necessário desenvolver um método eficiente de carga e descarga de imagens de e para o disco.

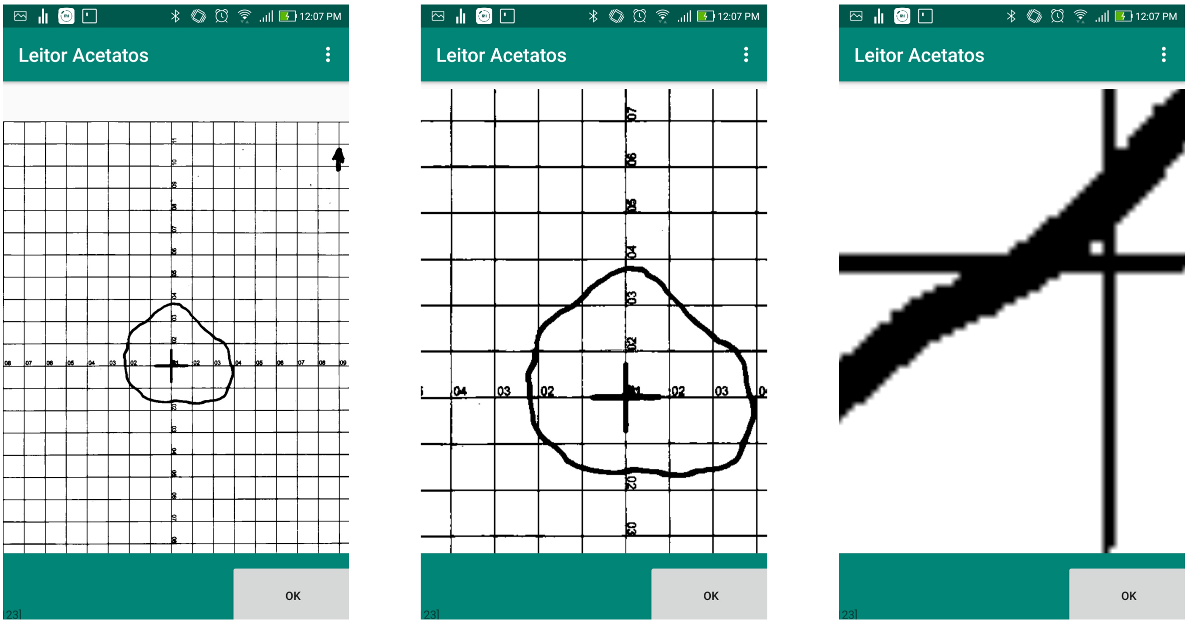
A primeira solução adotada foi utilizar a biblioteca *open-source Subsampling Scale Image View*[3], que é uma *view* customizada projetada para exibir imagens de alta resolução, como exemplificado na Figura 25 (o autor da biblioteca testou o carregamento de imagens de até 20000 pixeis x 20000 pixeis). Ela funciona de forma que uma imagem inicial de baixa resolução é carregada e quando se dá zoom, por movimento de pinça, a *view* recupera imagens menores de maior resolução.

4.3 PERSISTÊNCIA DE DADOS E COMUNICAÇÃO ENTRE ATIVIDADES

Os dados são salvos num serviço chamado *AppService*, ilustrado na Figura 26. A opção de se usar um serviço nativo do *Android* foi pelo fato de ele evitar a escrita e leitura desnecessárias dos dados da memória do dispositivo. A abordagem tradicional dos *intents* [4]

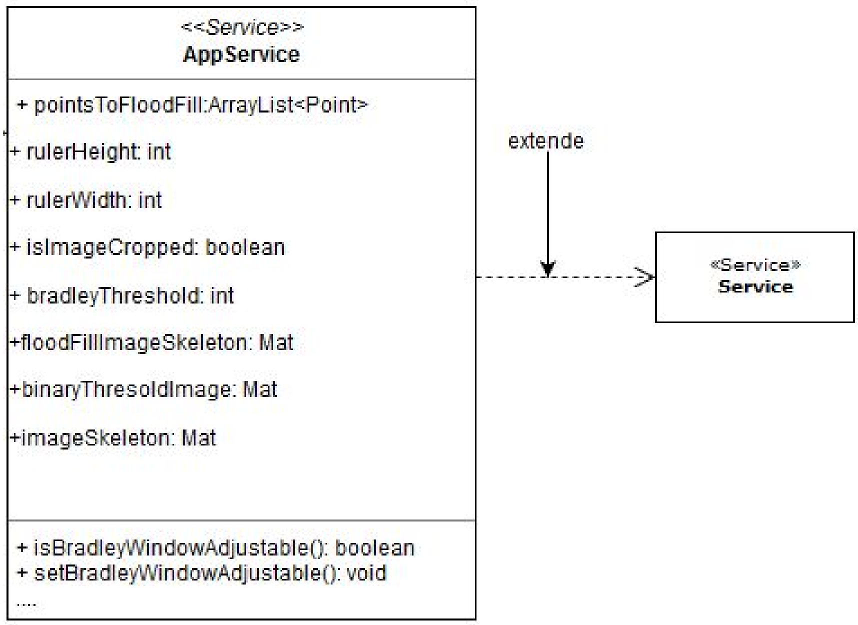
[1] <https://docs.opencv.org/4.1.0/d1/dfb/intro.html>
[2] *Ray tracing* é uma técnica utilizada para criação de imagens tridimensionais baseada na simulação do percurso que raios de luz teriam no mundo real.
[3] <https://github.com/davemorrissey/subsampling-scale-image-view>
[4] *Intents são a forma padrão de comunicação entre atividades no sistema operacional Android*

Figura 25 – Exemplo de zoom usando uma *view* customizada da biblioteca Subsampling Scale Image. View



não era totalmente satisfatória por conta da grande quantidade de dados e do seu tamanho. Tampouco o padrão de *software Singleton*, nativo do desenvolvimento do *Android*, era viável, por conta dos possíveis vazamentos de memória.

Figura 26 – Diagrama UML do AppService para persistência dos dados e comunição entre as atividades.



Fonte: Autoria própria

4.4 TELA PRINCIPAL

Na tela principal (Figura 27) o usuário interage com um menu onde escolhe a origem da imagem a ser analisada.

Figura 27 – Tela principal mostrando a aparência do aplicativo após a sua inicialização.

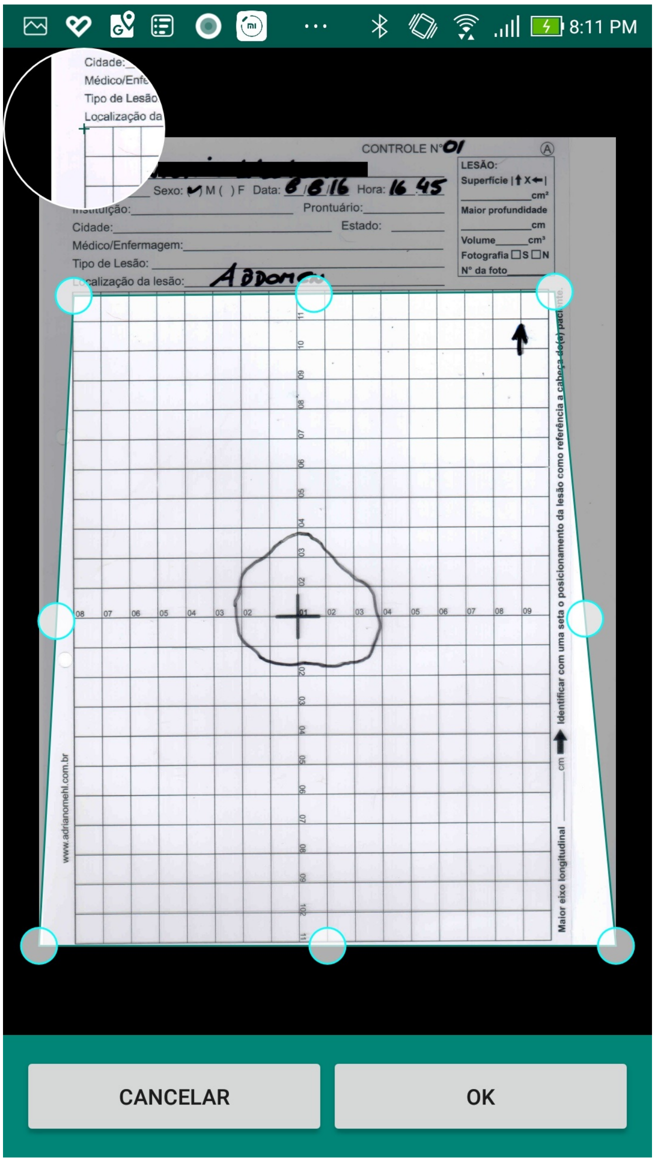
Fonte: Autoria própria

4.5 CLASSE IMAGEADJUSTMENTACTIVITY

É a atividade onde acontece a delimitação das bordas a imagem a ser planificada. Nela, o usuário interage com uma *CropImageView*, objeto da biblioteca SmartCropper que extende a *view* padrão do Android.

A biblioteca executa uma pré-seleção dos quatro pontos que delimitam a régua central do usuário. Pode-se a partir desta pré-seleção fazer alguns ajustes de posicionamento, como mostrado na Figura 28. O usuário tem acesso a uma lupa de ajuste onde pode ter visão melhorada da extremidade que está selecionando.

Figura 28 – Tela de delimitação das quatro extremidades da grade da régua.



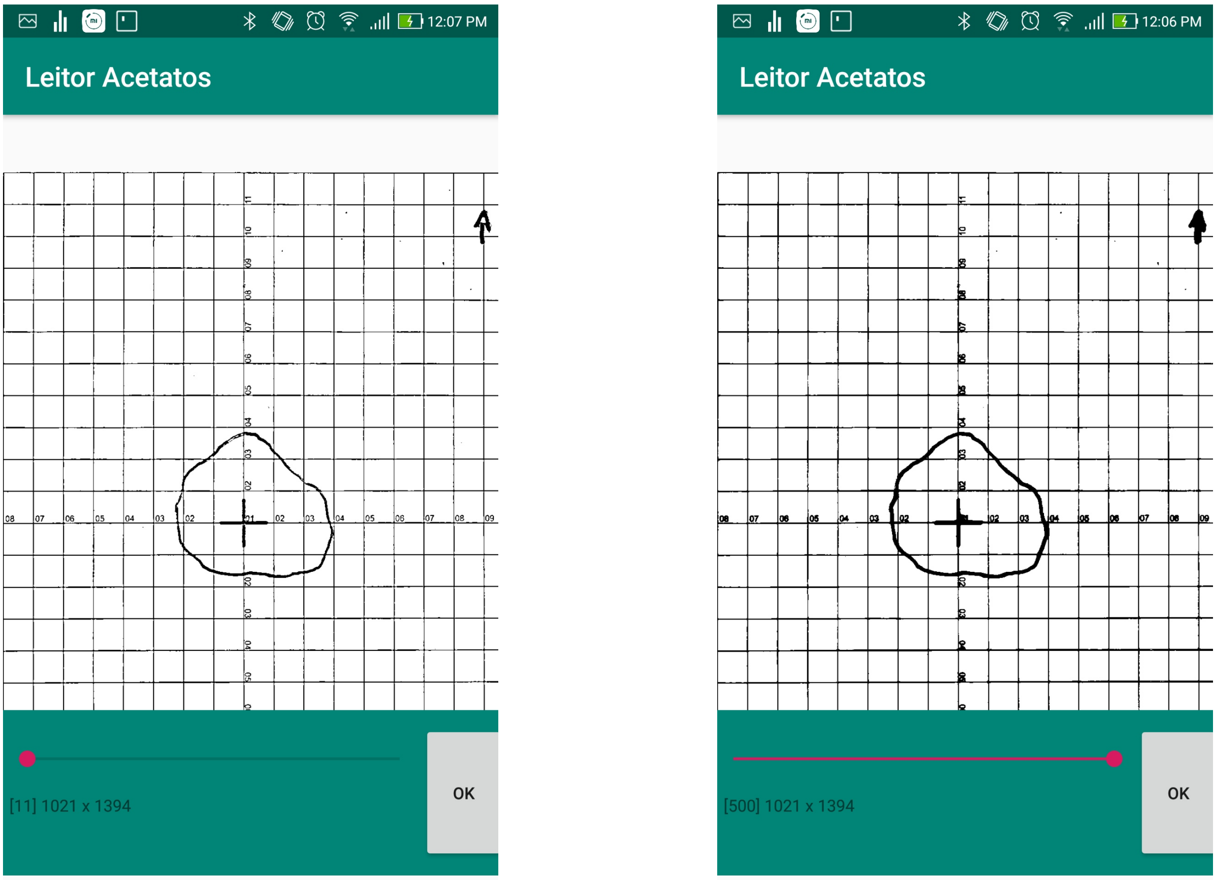
Fonte: Autoria própria

4.6 CLASSE *BINARIZATION*

Esta classe é responsável pela binarização da imagem escolhida. Pelo fato de a biblioteca OpenCV ter sido originalmente feita em C++, determinadas operações do código portado em Java podem ser lentas. Assim optou-se por usar a biblioteca de PDI *Catalano Framework*, desenvolvido por Diego Catalano [5], disponível para *desktop* e *Android*, desenvolvida nativamente em Java. A binarização utiliza a função *BradleyLocalThreshold*, que tem como parâmetro a janela de binarização. Esta janela pode ser ajustada pelo usuário (Figura 29), o que requer que o processo de binarização seja refeito cada vez que o usuário utilize um componente da interface (*seekBar*).

[5] <http://catalano-framework.com/>

Figura 29 – Tela de seleção de limiar *(threshold)* da operação de Bradley Local Threshold. Ao interagir com a barra da parte inferior, o usuário pode ajustar o nível de binarização, fazendo com que se evite a escolha de imagens com contornos não contínuos.



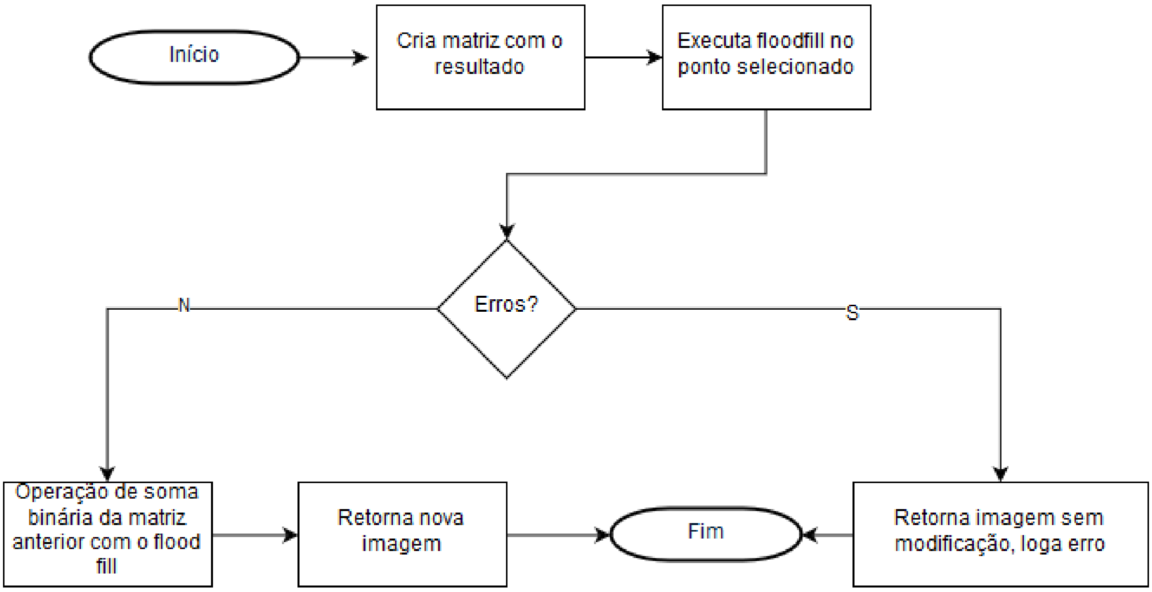
Como o processo é potencialmente pesado, a classe Binarization é chamada por uma AsyncTask, ou seja, uma tarefa assíncrona independente da *UI Thread*.

4.7 CLASSE *FLOODFILL*

Esta classe tem em sua construtora dois parâmetros, a imagem original com a matriz a ser preenchida e uma coordenada, que funciona como *seed*. A classe executa a operação chamada do OpenCV de *FloodFill*, constante na classe *ImgProc* na sua versão em Java, que preenche um componente conexo com determinada cor (no caso com pixeis pretos), como mostrado no fluxograma de Figura 30.

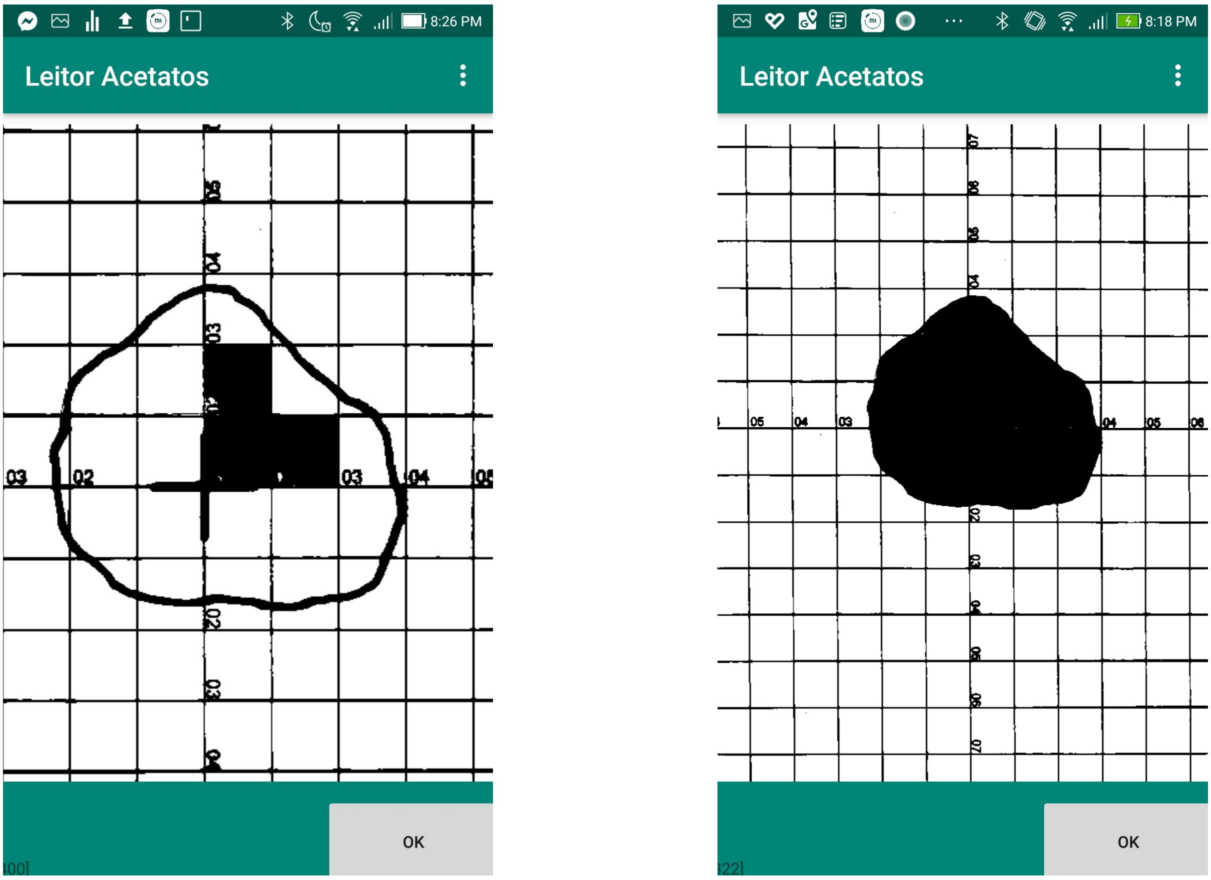
A cada toque do usuário no interior da ferida ocorre uma operação assíncrona de FloodFill, que uma vez finalizada retorna uma imagem na tela principal. Paralelamente são registrados os pontos em que ocorreu o FloodFill. Um exemplo de imagem após o FloodFill acontecer é a Figura 31.

Figura 30 – Fluxograma da classe Flood Fill.



Fonte: Autoria própria

Figura 31 – Operação de Flood Fill completada após os toques necessários do usuário



4.8 *CLASSE SKELETON*

Esta classe é responsável pela esqueletização. Nativamente, o OpenCV não possui implementação de operações de afinamento[6]. Desta forma, buscou-se uma alternativa que oferecesse um processamento rápido, de forma que optou-se pelo algoritmo de Zheng-Suen Zhang (1984). Vide Anexo A para a implementação do código em Java.

Uma vez que o processo de esqueletização estiver pronto, o usuário poderá finalizar o processo de FloodFill ao clicar no botão de OK (Figura 31). A imagem que foi esqueletizada sofrerá processo de FloodFill em todos os pontos que o usuário selecionou previamente. O usuário poderá visualizar a área total da imagem, assim como seu valor (Figura 32).

Figura 32 – Tela final exibindo o valor da área obtida da ferida.

Fonte: Autoria própria

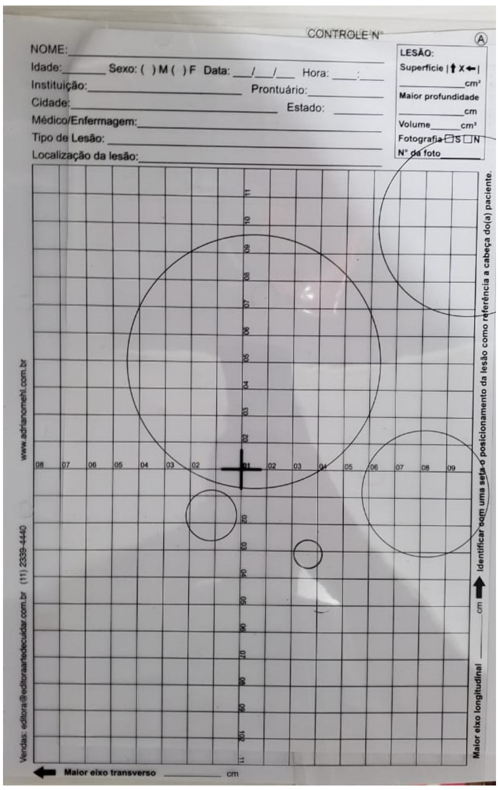
[6] Há uma implementação de uma biblioteca que extende as funcionalidades do OpenCV chamada ximgproc. No entanto, ela apresentou incompatibilidade com o Android.

5 ANÁLISE E DISCUSSÃO DOS RESULTADOS

Para estimar a qualidade das medições feitas pelo aplicativo em feridas, foram utilizadas cinco circunferências impressas em folha sulfite A4, de raios $0{,}5$, 1, $2{,}5$ e $3{,}5$ cm, para comparar os resultados obtidos com a área calculada. A régua foi colocado sobre a folha com as circunferências impressas e fotos foram tiradas buscando as seguintes condições: i) manter celular o mais paralelo possível ao plano no qual a folha estava apoiada e ii) a grade de $1cm$ x $1cm$ ocupando a maior parte da imagem possível. Todas as fotos foram tiradas em cômodos diferentes. Não foram tiradas fotos com reflexos fortes sobre as linhas das circunferências, pois a medição só é possível se o desenho do contorno da área a ser medida não apresentar falhas. A posição da régua em relação às circunferências também foi mudada a cada foto. A Figura 33 é um exemplo de amostra.

É importante observar que testes foram feitos com contornos conhecidos desenhados na própria régua, e os resultados são similares àqueles obtidos desenhando-se o contorno em uma folha sulfite A4 e colocando-se a régua sobre a folha. Em outras palavras, o contorno estar desenhado diretamente na régua ou em uma folha sulfite abaixo da régua não afeta os resultados.

Figura 33 – Exemplo de amostra seguindo as condições citadas.



Fonte: Autoria própria

No total 100 medições foram feitas, 20 para cada uma dos 5 circunferências. Destas

20, metade das medições usou fotos armazenadas no dispositivo celular e a outra metade fazendo a aquisição da imagem diretamente pelo aplicativo. As fotos correspondentes a essas 100 medições foram feitas utilizando a câmera de um Samsung Galaxy S9, as imagens tinham resolução de 4032x1960 pixeis, com tamanho em torno de 2 MB. Segue na Tabela 1 a média do erro relativo e o coeficiente de variação para as medidas de cada uma das circunferências:

Tabela 1 – Resultado dos testes com circunferências de raio conhecido e erros de outros aplicativos.

Área da circunferência em cm^2	0,79	3,14	19,63	38,48	78,54
Erro relativo médio deste trabaho	3,4%	0,8%	1,0%	1,0%	1,5%
Coeficiente de Variação	0,9%	0,9%	1,2%	1,0%	0,8%
Erro relativo do Visitrak	6,9%	2,4%	3,6%	3,6%	3,6%
Erro relativo do AreaMe	3,9%	1,2%	0,43%	0,43%	0,43%
Erro relativo do SilhoutteMobile	7,8%	0,9%	0,45%	0,45%	0,45%

Fonte: Autoria própria

O erro relativo é obtido a partir da equação 14, onde E_R é o erro relativo, M_O a medida obtida e M_E a medida esperada. Os valores expostos na tabela são as médias dos erros relativos.

$$E_R = \left| \frac{M_E - M_O}{M_E} \right| \tag{14}$$

O coeficiente de variação é a razão entre o desvio padrão e a média das medidas das amostras, o baixo valor obtido indica pouca variação nos valores obtidos durante as medições.

As medidas da circunferência menor apresentaram um erro relativo médio superior às outras. O mesmo observa-se em outras aplicações para medição de área de feridas (FOLTYNSKI PIOTR LADYZYNSKI, 2014). Os métodos comparados no artigo citado apresentam desempenho similar ao método utilizado neste trabalho. A Tabela 1 apresenta os erros relativos médios obtidos neste trabalho e também os relatados em (FOLTYNSKI PIOTR LADYZYNSKI, 2014).

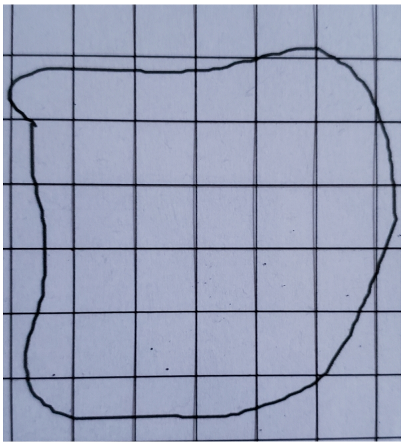
Para verificar se os contornos irregulares afetariam a precisão foram geradas, com o *software* Geogebra[1] (para que a área seja conhecida), cinco figuras de área conhecida e de formato similar às encontradas nas imagens fornecidas pelo médico Adriano Antônio Mehl (exemplo na Figura 34). Os resultados expressos na tabela 2 mostram que a precisão não é afetada.

Outras cinco medições foram realizadas com o mesmo celular já citado, porém não respeitando as condições que favorecem a qualidade da medição. As fotos foram tiradas da seguinte forma: a régua foi apoiada sobre bancadas com altura próxima da cintura e o usuário não se preocupou em tirar as fotos com a câmera paralela à régua. A Figura 35 é um exemplo destas amostras. A circunferência usada para as medidas foi a de $19{,}63\ cm^2$ apenas.

Nota-se nos resultados deste teste, apresentados na Tabela 3, que os erros relativos encontrados com fotografias retiradas sem respeitar as condições que favorecem a qualidade da

[1] <https://www.geogebra.org/?lang=en>

Figura 34 – Exemplo de amostra simulando o formato de uma ferida.



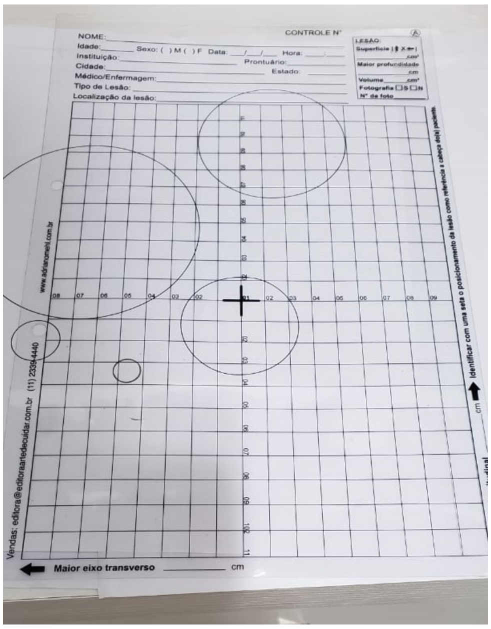
Fonte: Autoria própria

Tabela 2 – Resultado dos testes com figuras que simulam feridas.

Área esperada	12,34	16,11	18,95	23,96	32,29
Área medida	12,19	15,91	18,81	23,64	31,93
Erro Relativo	1,2%	1,2%	0,7%	1,3%	1,1%

Fonte: Autoria própria

Figura 35 – Exemplo de amostra seguindo as condições citadas



Fonte: Autoria própria

medição apresentam erros relativos pouco superiores à média dos testes seguindo os critérios que favorecem a qualidade da medida.

Mais cinco medidas foram realizadas, utilizando a circunferência de área 19,63 cm^2

Tabela 3 – Resultado dos testes sem favorecer a qualidade da medição

Medida	1	2	3	4	5
Erro Relativo	2,7%	1,2%	1,8%	4,9%	2,4%

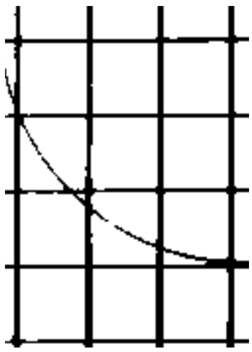
Fonte: Autoria própria

e seguindo os critérios que favorecem a qualidade da medição, porém utilizando um celular com câmera de resolução inferior à do celular usado anteriormente. As imagens foram então enviadas por aplicativo para outro *smartphone*, o que diminuiu a resolução das imagens para 1280x720 pixeis ficando com tamanho médio de 100 kB. Estas imagens (exemplo na Figura 36) podem apresentar problemas na medição da área, pois o processo de binarização pode resultar em descontinuidades no contorno da ferida, como pode ser observado na Figura 37. Isso afeta a seleção da ferida pois o preenchimento "vaza" para fora do contorno da região de interesse, como pode ser observado na Figura 38. Não foram feitos testes para descobrir a mínima resolução aceitável para o correto funcionamento do algoritmo.

Figura 36 – Amostra de má qualidade antes da binarização.

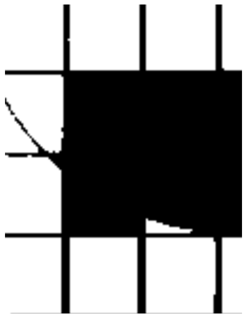
Fonte: Autoria própria

Figura 37 – Amostra de má qualidade.



Fonte: Autoria própria

Figura 38 – Problema decorrente do contorno com falha.



Fonte: Autoria própria

Considerando as cinco imagens mencionadas neste teste, três apresentaram descontinuidades nos contornos. Os erros relativos encontrados para as duas que permitiram medidas foram similares aos encontrados nas medidas dos testes anteriores: 3,6% e 2,1%. Isso indica que a resolução afeta muito mais na possibilidade de fazer a medição do que no erro obtido. Durante os testes foi observado que a experiência do usuário é prejudicada conforme maior é a área a ser medida, isso porque uma área de $n\ cm^2$ requer no mínimo n toques no seu interior para ser medida. Assim, o tempo necessário para executar a medição aumenta conforme a área a ser medida aumenta. Uma possível forma de amenizar este problema seria implementar um modo de pintura manual para selecionar a maior parte da ferida, utilizando o dedo como "pincel", e usar a pintura por toque próximo das bordas apenas.

6 CONCLUSÕES

Este trabalho foi embasado na necessidade médica de acompanhar o desenvolvimento de lesões cutâneas em diabéticos de forma a conduzir, da melhor forma possível, seu tratamento. Por estas lesões serem uma causa comum de amputação em portadores dessa doença, devido à baixa sensibilidade nas extremidades dos mesmos bem como sua cicatrização demorada, é importante que o plano de tratamento seja mudado para um mais eficaz caso o paciente não esteja respondendo ao mesmo, sendo a checagem do aumento ou diminuição da área da lesão a forma mais utilizada para verificar a resposta do paciente.

A equipe entrou em contato com o médico Dr. Adriano Mehl e notou que, para medir com precisão a área das feridas, o método utilizado por ele é demasiadamente demorado, e o modo alternativo e mais rápido para realizar esse processo de medição é pouco preciso. Dado isso, a equipe se propôs a criar uma solução para *smartphone* que tornaria o processo de obter uma medida precisa da área da ferida extremamente mais rápido. O instrumento utilizado pelo Dr. Adriano é uma régua, um acetato transparente com uma grade centi-metrada de medidas $17cm$ x $22cm$ o qual é utilizado para demarcar à caneta o contorno da ferida dos seus pacientes.

A solução idealizada, portanto, foi a criação de um aplicativo que, por meio de uma foto tirada dessa régua com o contorno da ferida já desenhado, pudesse disponibilizar rapidamente para o profissional da saúde a área da lesão.

A solução exige que o usuário apoie a régua sobre uma superfície plana e clara para tirar a foto e selecione manualmente na imagem adquirida as delimitações da régua e o interior da região demarcada pelo contorno da ferida. A aplicação é bem robusta contra sombras na imagem, mas é necessário evitar reflexos na régua.

A precisão alcançada em medições feitas com fotos de uma câmera de alta qualidade tiradas de acordo com as recomendações citadas no capítulo 5 (Análise e Discussão dos Resultados) foi considerada extremamente satisfatória. O tempo levado para executar as medições aumenta conforme aumenta a área a ser medida. Contudo, ainda é mais rápido que o método de boa precisão utilizado pelo Dr. Adriano Mehl, baseado no *software* Digimizer, de forma que esse quesito também foi considerado satisfatório.

A solução apresentada neste trabalho também pode medir áreas delimitadas por contornos em outras folhas que não a régua usada nesse trabalho. Isso porque o cálculo da área é baseado na área total da folha, a qual é selecionada manualmente. Como é possível esse ajuste manual no aplicativo dessa área total, pode-se calcular áreas em qualquer folha desde que sua área seja conhecida.

Finalmente, pode-se afirmar que a ferramenta desenvolvida, implementada e testada neste trabalho é capaz de auxiliar no trabalho diário do profissional da saúde que opte por usar a régua, trazendo o benefício de uma medida precisa que pode ser obtida rapidamente com dispositivos comuns.

6.1 TRABALHOS FUTUROS

Para tornar mais rápida e prática a seleção do interior de feridas de áreas grandes, pode-se desenvolver uma método de preenchimento no qual o usuário arrastaria o dedo no interior da ferida ao invés de dar um toque por quadrado dentro da área delimitada.

Trabalhos futuros também podem ser realizados no sentido de deixar automática a seleção da área. Uma sugestão seria permitir que o usuário tocasse apenas uma vez no interior de cada contorno de ferida desenhado na régua, ao invés de requisitar vários toques como no método proposto.

Outra melhoria seria o desenvolvimento independente de uma biblioteca similar à SmartCropper, que encontra-se atualmente com documentação limitada, poucas opções de manipular a API e em processo de depreciação.

Referências

BRADLEY, G. R. D. Adaptive thresholding using the integral image. p. 13–21, 2007. Citado 3 vezes nas páginas 8, 34 e 35.

BRADSKI, A. K. G. **Learning OpenCV**: Computer vision with the opencv library. [S.l.], 2008. 555 p. Citado na página 41.

BRASIL. **Caderno de Atenção Básica. Diabetes Mellitus**: Estratégias para o cuidado da pessoa com doença crônica. [S.l.], 2013. 162 p. Citado 3 vezes nas páginas 18, 19 e 20.

CANNY, J. A computational approach to edge detection. p. 679–698, 1986. Citado na página 27.

CS, R. **Wound healing in the patient with diabetes mellitus**. [S.l.], 1990. 14 p. Citado na página 15.

FOLTYNSKI PIOTR LADYZYNSKI, J. M. W. P. A new smartphone-based method for wound area measurement. p. 6, 2014. Citado na página 49.

JEFFCOATE, K. G. H. W. J. Diabetic foot ulcers. p. 7, 2003. Citado na página 21.

KEAST, C. K. B. D. H. **MEASURE: A proposed assessment framework for developing best practice recommendations for wound assessment.** [S.l.], 2004. 17 p. Citado 4 vezes nas páginas 15, 16, 21 e 22.

LIMA, E. P. A. Maria Helena de M. **DIABETES MELLITUS E O PROCESSO DE CICATRIZAÇÃO CUTÂNEA**. [S.l.], 2013. 3 p. Citado na página 15.

OKONKWO, L. A. D. U. A. Diabetes and wound angiogenesis. p. 4, 2019. Citado na página 20.

OPENCV. **OpenCV 2.4.13.7 Documentation, https://docs.opencv.org/2.4/index.html**. [S.l.], 2014. Citado 2 vezes nas páginas 31 e 33.

OSTERD, D. K. H. L. The basic principles of wound care. p. 4, 1998. Citado 3 vezes nas páginas 18, 19 e 20.

SHEEHAN, P. Percent change in wound area of diabetic foot ulcers over a 4-week period is a robust predictor of complete healing in a 12-week prospective trial. p. 4, 2003. Citado 2 vezes nas páginas 15 e 21.

SUZUKI, K. A. S. Topological structural analysis of digitized binary images by border following. p. 15, 1985. Citado na página 30.

VILLINES, Z. **How does diabetes affect wound healing?** [S.l.], 2019. 1 p. Citado na página 15.

WHO. Global report on diabetes. p. 88, 2016. Citado 2 vezes nas páginas 19 e 20.

ZHANG, C. S. T. **A fast parallel algorithm for thinning digital patterns**. [S.l.], 1984. 236-239 p. Citado 2 vezes nas páginas 36 e 47.

Anexos

ANEXO A – Implementação em Java do algoritmo Zhang-Suen Thinning

Esta implementação em Java foi criada por Nayef Reza[1] e posteriormente adaptada para Android usando OpenCV.

```java
package imageProcessing.service;

import imageProcessing.model.Point;

import java.util.LinkedList;
import java.util.List;

/**
 * Created by nayef on 1/26/15.
 */
public class ThinningService {
    /**
     * @param givenImage
     * @param changeGivenImage decides whether the givenArray should be modified or a clone should be used
     * @return a 2D array of binary image after thinning using zhang-suen thinning algo.
     */
    public int[][] doZhangSuenThinning(final int[][] givenImage, boolean changeGivenImage) {
        int[][] binaryImage;
        if (changeGivenImage) {
            binaryImage = givenImage;
        } else {
            binaryImage = givenImage.clone();
        }
        int a, b;
        List<Point> pointsToChange = new LinkedList();
        boolean hasChange;
        do {
            hasChange = false;
            for (int y = 1; y + 1 < binaryImage.length; y++) {
                for (int x = 1; x + 1 < binaryImage[y].length; x++) {
                    a = getA(binaryImage, y, x);
                    b = getB(binaryImage, y, x);
                    if (binaryImage[y][x] == 1 && 2 <= b && b <= 6 && a == 1
                            && (binaryImage[y - 1][x] * binaryImage[y][x + 1] * binaryImage[y + 1][x] == 0)
```

[1] https://nayefreza.wordpress.com/2013/05/11/zhang-suen-thinning-algorithm-java-implementation/

```
                            && (binaryImage[y][x + 1] * binaryImage[y + 1][x] * binaryImage[y][x - 1] == 0)) {
                        pointsToChange.add(new Point(x, y));
//binaryImage[y][x] = 0;
                        hasChange = true;
                    }
                }
            }
            for (Point point : pointsToChange) {
                binaryImage[point.getY()][point.getX()] = 0;
            }
            pointsToChange.clear();
            for (int y = 1; y + 1 < binaryImage.length; y++) {
                for (int x = 1; x + 1 < binaryImage[y].length; x++) {
                    a = getA(binaryImage, y, x);
                    b = getB(binaryImage, y, x);
                    if (binaryImage[y][x] == 1 && 2 <= b && b <= 6 && a == 1
                            && (binaryImage[y - 1][x] * binaryImage[y][x + 1] * binaryImage[y][x - 1] == 0)
                            && (binaryImage[y - 1][x] * binaryImage[y + 1][x] * binaryImage[y][x - 1] == 0)) {
                        pointsToChange.add(new Point(x, y));
                        hasChange = true;
                    }
                }
            }
            for (Point point : pointsToChange) {
                binaryImage[point.getY()][point.getX()] = 0;
            }
            pointsToChange.clear();
        } while (hasChange);
        return binaryImage;
    }

    private int getA(int[][] binaryImage, int y, int x) {
        int count = 0;
//p2 p3
        if (y - 1 >= 0 && x + 1 < binaryImage[y].length && binaryImage[y - 1][x] == 0 && binaryImage[y - 1][x + 1] == 1) {
            count++;
        }
//p3 p4
        if (y - 1 >= 0 && x + 1 < binaryImage[y].length && binaryImage[y - 1][x + 1] == 0 && binaryImage[y][x + 1] == 1) {
            count++;
        }
```

```
//p4 p5
        if (y + 1 < binaryImage.length && x + 1 < binaryImage[y].length && binaryImage[y][x + 1] == 0 && binaryImage[y + 1][x + 1] == 1) {
            count++;
        }
//p5 p6
        if (y + 1 < binaryImage.length && x + 1 < binaryImage[y].length && binaryImage[y + 1][x + 1] == 0 && binaryImage[y + 1][x] == 1) {
            count++;
        }
//p6 p7
        if (y + 1 < binaryImage.length && x - 1 >= 0 && binaryImage[y + 1][x] == 0 && binaryImage[y + 1][x - 1] == 1) {
            count++;
        }
//p7 p8
        if (y + 1 < binaryImage.length && x - 1 >= 0 && binaryImage[y + 1][x - 1] == 0 && binaryImage[y][x - 1] == 1) {
            count++;
        }
//p8 p9
        if (y - 1 >= 0 && x - 1 >= 0 && binaryImage[y][x - 1] == 0 && binaryImage[y - 1][x - 1] == 1) {
            count++;
        }
//p9 p2
        if (y - 1 >= 0 && x - 1 >= 0 && binaryImage[y - 1][x - 1] == 0 && binaryImage[y - 1][x] == 1) {
            count++;
        }
        return count;
    }

    private int getB(int[][] binaryImage, int y, int x) {
        return binaryImage[y - 1][x] + binaryImage[y - 1][x + 1] + binaryImage[y][x + 1]
                + binaryImage[y + 1][x + 1] + binaryImage[y + 1][x] + binaryImage[y + 1][x - 1]
                + binaryImage[y][x - 1] + binaryImage[y - 1][x - 1];
    }
}
```

Listing A.1 – "Serviço de Afinamento"

```
package imageProcessing.model;

/**
 * Created by nayef on 1/27/15.
```

```
 */
public class Point {
    private int x;
    private int y;

    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public int getX() {
        return x;
    }

    public void setX(int x) {
        this.x = x;
    }

    public int getY() {
        return y;
    }

    public void setY(int y) {
        this.y = y;
    }
}
```

Listing A.2 – "Ponto customizado"