CENTRO UNIVERSITÁRIO "EURÍPIDES DE MARÍLIA" – UNIVEM
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

MARCELO ROSSI

FERRAMENTA EM JAVA PARA CONTAGEM DE NECROSE EM TECIDO ANIMAL

MARÍLIA
2005

MARCELO ROSSI

FERRAMENTA EM JAVA PARA CONTAGEM DE NECROSE EM TECIDO ANIMAL

Monografia apresentada à "Fundação de Ensino Eurípides Soares da Rocha", mantenedora do Centro Universitário "Eurípides de Marília", para a obtenção do Título de Bacharel em Ciência da Computação.

Orientador:
Prof. Dr. Márcio Eduardo Delamaro

MARÍLIA
2005

ROSSI, Marcelo

Ferramenta em Java para contagem de necrose em tecido animal / Marcelo Rossi; orientador: Prof. Dr. Márcio Eduardo Delamaro. Marília, SP: [s.n.], 2005.

74 f.

Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Centro Universitário Eurípides de Marília – Fundação de Ensino Eurípides Soares da Rocha.

1. Engenharia de Software

CDD: 005.1

Marcelo Rossi

"FERRAMENTA EM JAVA PARA CONTAGEM DE NECROSE EM TECIDO ANIMAL"

Banca examinadora da monografia apresentada ao Curso de Bacharelado em Ciência da Computação do UNIVEM/F.E.E.S.R., para obtenção do Título de Bacharel em Ciência da Computação.

Nota: 9,5 (Nove e Meio)

Orientador: Márcio Eduardo Delamaro _____

1º Examinador: Fátima L. S. Nunes Marques _____

2º Examinador: Ana Paula Piovesan Melchiori Peruzza _____

Marília, 01 de dezembro de 2005.

Aos meus pais, *Marcos e Fátima*.
Ao meu irmão, *Fernando*.
Aos meus familiares e amigos, em especial
à minha querida amiga *Larissa Prohmann*.

AGRADECIMENTOS

Agradeço aos meus familiares e principalmente aos meus pais, Marcos e Fátima, por me proporcionarem conforto e condições necessárias para minha formação.

Agradeço também aos meus professores e a todas as pessoas que participaram da minha vida, pois direta ou indiretamente me proporcionaram experiências que de uma maneira ou de outra fazem parte da minha formação e do que sou hoje. É impossível lembrar todos os nomes, mas de maneira geral agradeço a todos os: Marcos, Fátimas, Larissas, Alines, Cláudias, Gustavos, Rodrigos, Guilhermes, Naras, Cássias, Lívias, Ritas, Maurícios, Éricos, Júniors, Leandros, Suelis, Denises, Samantas, Fernandos, Carlos, Guilhermes, Antônios, Patricias, Thaíses, Anas, Paulas, Márcios, Renatos, Fabianos, Ilsons, Lourdes, Fábios, Albertos, Paulos, Julios, Alexandres, Williams, Célias, Hélios, Cezárias, Renatas, Julianas, Primos, Luíses, Gilbertos, Joãos, Sônias, Josés, Marias, Brunos, Bernadetes, Andrés, Deolindas, Carolinas, Mários, Camilas, Natálias. Claro que não posso esquecer da Terra, da Natureza, do Sol, das Estrelas e da Lua.

You cannot achieve the impossible
without attempting the absurd.
Autor Desconhecido

ROSSI, Marcelo. **FERRAMENTA EM JAVA PARA CONTAGEM DE NECROSE EM TECIDO ANIMAL.** 2005. Monografia do Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Centro Universitário "Eurípides de Marília", Fundação de Ensino "Eurípides Soares da Rocha", Marília, 2005.

RESUMO

A análise de imagens digitais por meios computacionais é de grande valia para diversas áreas. Áreas como biomedicina, geografia, astronomia, antropologia, entre outras necessitam da análise de imagens para fazer comparações ou gerar diagnósticos. O presente trabalho visa a implementar um software capaz de analisar imagens digitais de tecidos necrosados de animais e gerar um diagnóstico informando a porcentagem do tecido animal que está necrosado. A implementação deste software utiliza, para a manipulação das imagens, alguns recursos da API JAI da linguagem de programação Java.

Palavras-chaves: Análise e tratamento de imagens; Java Advanced Imaging; JAI; Biomedicina; Contagem de Necrose em Tecido Animal.

ROSSI, Marcelo. **JAVA TOOL FOR COUNTING OF ANIMAL NECROSIS**. 2005. Monograph of the Work of Conclusion of Course (Bachelor in Computer Sciences) – Centro Universitário "Eurípides de Marília", Fundação de Ensino "Eurípides Soares da Rocha", Marília, 2005.

ABSTRACT

The analysis of digital images by computational ways is of great value for many areas. Areas as biomedicine, geography, astronomy, anthropology, among others need the analysis of images to make comparisons or to generate diagnostic. The present work aims at to implement a software capable to analyze digital necrosis images of animals and to generate a diagnosis saying how many percent of the animal's image is necrosis. The implementation of this software uses, for the manipulation of the images, some resources of JAI API of the Java programming language.

Key-Words: Analysis and treatment of images; Java Advanced Imaging; JAI; Biomedicine; Counting of Animal Necrosis.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

FAMEMA: Faculdade de Medicina de Marília

RGB: Vermelho, Verde e Azul

CMY: Ciano, Amarelo e Magenta

YIQ: Luminância e informações das cores

API: Applications Programming Interfaces

JAI: Java Advanced Imaging

2D: Duas Dimensões

3D: Três Dimensões

Pixel: Picture Element

SUMÁRIO

1. INTRODUÇÃO

Este trabalho visa a auxiliar o pessoal da área biológica, tendo como objetivo implementar um software que seja capaz de realizar a contagem da necrose em tecidos de animais.

Por razões de experiências cientificas é necessário realizar um estudo para conseguir identificar qual a porcentagem ocorrida de necrose em um tecido animal, porém esta contagem se torna extremamente difícil e quase que impossível de ser feita sem o auxílio da computação.

Este trabalho manipula imagens cedidas por um grupo de estudos da FAMEMA (Faculdade de Medicina de Marília). Essas fotografias digitais foram tiradas de tecidos animais previamente preparados para o fim específico deste trabalho.

A manipulação destas imagens consiste em uma série de operações gráficas sobre os arquivos das mesmas, que visam a remover o fundo da imagem, ou seja, o cenário de onde a fotografia foi tirada, para que seja feita a análise e também a contagem da necrose apenas sobre o tecido animal e não do tecido com o cenário.

Uma vez feita a retirada do fundo e a limpeza da imagem, a contagem da necrose no tecido animal é iniciada. É sabido que uma imagem digital é formada por pontos (pixels). Portanto, a contagem da necrose consiste em analisar os pixels do tecido animal verificando os seus tons de cores. Os pixels com tons escuros são identificados como necrosados e os com tons claros são identificados como saudáveis. Após esta contagem são feitas algumas contas de onde é obtida a porcentagem de necrose do tecido.

Portanto este trabalho aborda o estudo e a manipulação de imagens digitais em meios computacionais, especialmente com a linguagem de computador Java e suas bibliotecas, e como objetivo final este trabalho visa à implementação de um software em Java que faça as

operações de análise necessárias para a contagem da necrose em fotografias digitais de tecidos de animais.

O capítulo 2 faz um estudo sobre as operações usadas nas imagens para implementar o software, mostrando a teoria necessária para a manipulação das imagens.

O capítulo 3 reporta sobre a linguagem de programação escolhida para a implementação do software e os motivos que levaram à escolha da linguagem. Também explica sobre uma biblioteca contendo a implementação da teoria usada na manipulação das imagens.

O capítulo 4 reporta como a implementação do software foi feita. Passo a passo é explicado o que foi usado, como foi usado e porque foi usado.

O capítulo 5 é um guia mostrando o uso da aplicação desenvolvida, visando a esclarecer dúvidas sobre a operabilidade da aplicação. E também apresenta exemplos da análise de imagens de tecidos animais necrosados.

A conclusão do trabalho é apresentada no capítulo 6.

2. Processamento de Imagens

Imagens são matrizes de valores que representam intensidade de luz. Uma imagem monocromática pode ser representada por f(x,y), onde x e y são as coordenadas espaciais e f o valor destas coordenadas que representa a intensidade de luz neste ponto. Os elementos da matriz são chamados de elementos da imagem, elementos da figura, “pixels” ou “pels”, sendo que o significado destes dois últimos é “picture elements” ou “elementos da figura” (GONZALEZ E WOODS, 2000).

A luz refletida por um objeto enxergado em situações corriqueiras pode ser descrita por:

$0 < r(x,y) < 1$

A equação acima indica que a reflectância está limitada entre 0 (absorção total da luz pelo objeto) e 1 (reflexão total da luz pelo objeto). Reflectância é a luz refletida por um objeto.

A luz é uma forma de energia, representada aqui por f(x, y) e como tal deve ser positiva e finita, sendo que:

$0 < f(x,y) < \text{infinito}$

2.1. Armazenamento

Imagens não são matrizes de apenas duas dimensões, elas têm profundidade. A profundidade da matriz de uma imagem é responsável pelas possibilidades de cores que cada elemento da figura, pixel, pode assumir.

Uma matriz equivalente a uma imagem de TV preta e branca é uma matriz 512 x 512 x 128, onde 512 x 512 são a largura e altura e 128 a possibilidade de tons de cinza que cada pixel pode assumir (GONZALEZ E WOODS, 2000).

Quando é dito: “Uma imagem de 16 bits com resolução de 2048 x 1024”. A imagem referida possui largura de 2048 pixels, altura 1024 pixels e profundidade de cores, ou capacidade de cores, de 16 bits, ou seja, 2^{16} cores por pixels.

O armazenamento destas informações acontece em mídias comuns, como: discos magnéticos, fitas magnéticas ou discos ópticos. Qualquer que for o meio de armazenamento, a quantidade ocupada por uma imagem é de fácil cálculo. Basta multiplicar a largura com a altura e com a profundidade em bits.

No exemplo anterior, da imagem de 16 bits com resolução de 2048 x 1024, a quantidade ocupada por ela é 2048 x 1024 x 16 bits que é equivalente a: 33.554.432 bits, ou 4.194.304 bytes, ou 4.096 Kilobytes ou 4 Megabytes.

Existem técnicas usadas para comprimir imagens, porém este assunto é irrelevante ao propósito deste trabalho.

2.2. Processamento de imagens coloridas

Manipular imagens cinzas é vantajoso no que diz respeito à eficiência. Imagens em uma escala onde só existem representações de cinza possuem menos informações do que imagens coloridas e por isto é menos custoso manipulá-las.

Entretanto imagens coloridas possuem vantagens, principalmente em sistemas automatizados de análise de imagens. As cores podem ser usadas para diferenciar objetos, tornando assim mais fácil o reconhecimento por parte do software implementado.

Outra vantagem das imagens coloridas é referente aos seres humanos. O olho humano é capaz de diferenciar milhares de tons e intensidades de cores, contra apenas algumas dezenas de tons de cinza (GONZALEZ E WOODS, 2000).

2.2.1. Conceitos sobre cores

O conceito básico de cores foi descoberto por Isaac Newton em 1666. Newton descobriu que com um prisma de vidro um feixe de luz solar pode ser dividido em um espectro contínuo de cores do violeta ao vermelho. Este espectro possui seis regiões que podem ser identificadas claramente: violeta, azul, verde, amarelo, laranja e vermelho.

Essas cores podem ser formadas partindo-se de três cores primárias: vermelho (representado por R, do inglês RED), verde (representado por G, do inglês GREEN) e azul (representado por B, do inglês BLUE). Estas cores primárias são adotadas para a representação das outras cores devido ao fato de que a estrutura do olho humano é de tal forma que todas as cores que ele é capaz de captar podem ser obtidas partindo-se dessas três primárias.

É desta maneira que os aparelhos de TV exibem suas cores. O aparelho de TV possui uma matriz de fósforo eletrossensitivo. Esta matriz é excitada por um canhão eletrônico, que pode produzir excitação apenas na faixa de luz correspondente ao vermelho, verde, azul ou a qualquer combinação destas cores.

A potência do canhão também pode ser ajustada, produzindo assim diferentes intensidades das cores primárias e, conseqüentemente, diferentes combinações. É devido à combinação destas cores primárias que é feita a síntese dos milhares de tons de cores que a TV pode produzir (GONZALEZ E WOODS, 2000).

2.2.2. Modelos de cores

Os modelos de cores são especificações que padronizam a sua representação. Os modelos atuais, em sua maioria, são orientados ao hardware, ou seja, a impressoras, monitores, e outros aparelhos que produzem cores.

Alguns modelos famosos são:

- O modelo RGB, usado em câmeras de vídeo e monitores.
- O modelo CMY (ciano, amarelo, magenta), usado em impressoras.
- O modelo YIQ (luminância e cores), que é padrão na transmissão de TV.

Também há outros modelos como o HSI e o HSV usados para manipulação de imagens coloridas em computadores.

Neste trabalho o modelo usado como padrão é o RGB que, como dito anteriormente, representa as cores vemelho(R – *RED*), verde (G – *Green*) e azul (B – *Blue*).

2.2.3. Modelo RGB

O modelo RGB é composto por três planos, cada um representando uma cor primária com o seu valor. Portanto uma imagem colorida neste modelo possui largura, altura e profundidade em três planos, o que pode ser visto como uma matriz de tamanho [m, n, 3], onde m e n são, respectivamente, a largura e altura da imagem e 3 cada um dos planos. Este esquema é representado na Figura 1.

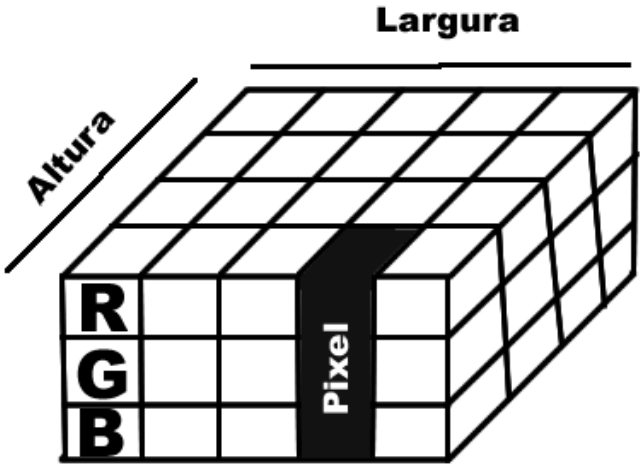


Figura 1: Modelo RGB

2.3. OPERAÇÕES COM IMAGENS

Com exceção da obtenção e visualização de imagens, as operações com imagens digitais são feitas de forma algorítmica, com operações matemáticas e lógicas. E assim sendo podem ser implementadas inteiramente através de softwares, com o uso de hardware específico unicamente para ganho de velocidade.

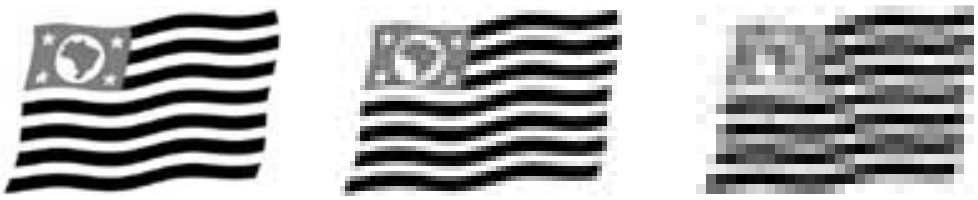
Os sistemas de processamento de imagens disponíveis atualmente é uma mistura de computadores comercialmente disponíveis e hardware especializado para processamento de imagens, com a operação geral sendo comandada por um software no computador (GONZALEZ E WOODS, 2000).

2.3.1. Resolução

É complicado dizer se uma imagem é de "boa" qualidade. Isto depende muito da aplicação na qual a imagem será usada. A forma de medir a qualidade de uma imagem é com a sua resolução espacial.

Resolução espacial é a quantidade de pixels por unidade de espaço. Se uma imagem de 1024 x 512 pixels for impressa em um papel de 16 x 8 centímetros. A configuração de impressão usada para imprimir esta imagem no papel foi de 64 pixels por centímetro, esta configuração é chamada de resolução espacial.

Se a mesma imagem (1024 x 512 pixels) for impressa em uma área maior, digamos de 32 x 16 centímetros, então a resolução espacial será de 32 pixels por centímetro. Estas são grandezas inversamente proporcionais, ou seja, enquanto a área de impressão dobra a resolução espacial é reduzida pela metade. A Figura 2 possui imagens com resoluções espaciais diferentes. As Figuras 2 A, 2 B e 2 C possuem, respectivamente, 28, 14 e 7 pixels por centímetros de resolução espacial. A diferença de qualidade0 entre elas é nítida.



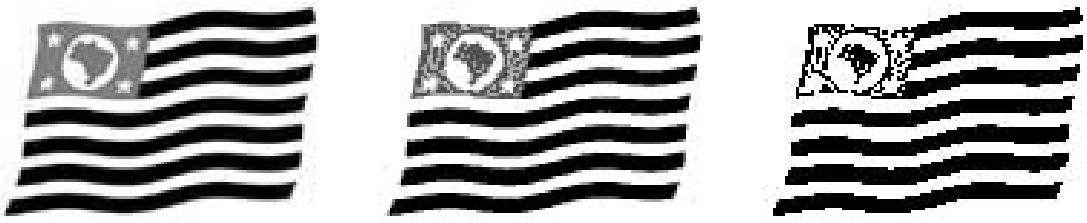
A B C

Figura 2: Resolução espacial

2.3.2. Profundidade

Mesmo a resolução espacial se mantendo fixa, a aparência (qualidade) de uma imagem pode melhorar ou piorar dependendo da quantidade de níveis de cores usada, ou seja, a profundidade de tons que a imagem suporta.

Para uma imagem com profundidade de 1 bit temos o valor 0 indica a cor Preta e o valor 1 indica a cor Branca. Para a profundidade de 3 bits o valor 000 indica a cor Preta e o valor 111 indica a cor branca, sendo que os valores intermediários (001, 010, 011, 100, 101, 110) indicam os tons de cinzas possíveis de serem representados. A Figura 3 apresenta imagens com níveis diferentes de cores. As Figuras 3 A, 3 B e 3 C possuem, respectivamente, 8, 4 e 2 bits de profundidade. É facilmente notada a diferença de qualidade entre as três imagens.



A B C

Figura 3: Profundidade dos pixels

2.3.3. Relacionamento entre pixels

A manipulação de imagens se baseia em operações matemáticas com os valores das matrizes que as representam. Por exemplo, escurecer uma imagem consiste em diminuir o valor de todos os pixels da matriz, ou seja, diminuir a intensidade luminosa, o valor de f(x, y) para toda a matriz (GONZALEZ E WOODS, 2000).

Para a operação citada acima, não é necessário identificar nenhum relacionamento entre os pixels da imagem, porém em uma situação em que é necessário escurecer apenas um elemento da imagem. Para este caso é necessário estipular uma relação entre os pixels da imagem para ser possível identificar quais pixels pertencem ao elemento que deve ser escurecido. Esta relação é chamada de conectividade entre pixels.

Principalmente para a identificação de bordas e componentes de regiões em uma imagem, o conceito de conectividade entre pixels é importante.

Dois pixels são ditos conectados se estes são adjacentes, ou seja, vizinhos e se seus valores (tons de cinza) satisfazem um critério (por exemplo, se os tons de cinza deles estão entre os valores 123 e 126).

2.3.3.1. Pixels vizinhos

Para entender a conectividade é necessário saber como identificar pixels vizinhos. Suponha um pixel p com coordenadas (x, y), este pixel possui quatro vizinhos.

Os vizinhos do pixel p citado acima são:

(x + 1, y); (x – 1, y); (x, y + 1); (x, y – 1).

Este conjunto de quatro pixels é chamado de vizinhança-de-4 do pixel p e é representado por $N_4(p)$. Cada pixel deste conjunto está a uma unidade de distância do pixel p, no eixo cartesiano X ou Y. Claro que se o pixel p estiver na borda da imagem um ou dois dos vizinhos não existirão, pois eles teriam que ficar fora da imagem.

O pixel p também possui mais 4 vizinhos, os vizinhos diagonais que são denotados por $N_D(p)$ e estão duas unidades distantes do pixel p (uma unidade no eixo cartesiano X e outra no Y). O conjunto dos vizinhos diagonais de p(x, y) é dado por:

(x-1, y-1); (x-1, y+1); (x+1, y+1); (x-1, y-1).

O conjunto dos pixels vizinhos-de-4, junto com os vizinhos diagonais do pixel p é chamado de vizinhança-de-8 de p e é representada por $N_8(p)$. Se o pixel p estiver na borda da imagem alguns elementos de $N_8(p)$ podem ficar fora da imagem (GONZALEZ E WOODS, 2000).

2.3.3.2. Conectividade

Sabendo os conceitos sobre vizinhança de pixels é possível esclarecer mais sobre a conectividade deles. Para identificar um conjunto de pixels conectados é necessário analisar se estes são vizinhos e se seus valores satisfazem um critério.

Suponha uma imagem com profundidade de 256 tons de cinza. A tarefa é identificar um objeto nesta imagem. O primeiro passo é estipular qual o conjunto de cores que este objeto possui e em seguida procurar por pixels vizinhos com estes valores.

Considere V como o conjunto de valores de níveis de cinza que este objeto pode possuir. Partindo de um pixel p(x, y) previamente escolhido, os vizinhos de p serão analisados e comparados com o conjunto V. Os vizinhos que possuírem os valores pertencentes ao conjunto V serão marcados e analisados da mesma forma que o pixel p. Este algoritmo se repete até toda imagem ser analisada ou não haver mais vizinhos que atendem o critério de conectividade (GONZALEZ E WOODS, 2000).

2.3.3.3. Distância entre pixels

A distância entre dois pixels p(x, y) e q(s, t) é representada por D(p, q) e pode ser calculada dos seguintes modos:

1. Distância Euclidiana(DE), definida por:
   a. $DE(p, q) = [(x – s)^2 + (y – t)^2]^{1/2}$
2. Distância "quarteirão" ou "city block" (D4), definida por:
   a. $D4(p, q) = | x – s | + | y – t |$
3. Distância que forma um quadrado em p(x,y) (D8), definida por:
   a. $D8(p, q) = \max(| x – s |, | y – t |)$

A Figura 4 mostra as distâncias do pixel p (com o valor 0) em relação aos outros pixels. Os números apresentados são a distância dos pixels em relação ao pixel p que está localizado no centro.

$8^{1/2}$	$5^{1/2}$	2	$5^{1/2}$	$8^{1/2}$
$5^{1/2}$	$2^{1/2}$	1	$2^{1/2}$	$5^{1/2}$
2	1	0	1	2
$5^{1/2}$	$2^{1/2}$	1	$2^{1/2}$	$5^{1/2}$
$8^{1/2}$	$5^{1/2}$	2	$5^{1/2}$	$8^{1/2}$

A

Distância Euclidiana

4	3	2	3	4
3	2	1	2	3
2	1	0	1	2
3	2	1	2	3
4	3	2	3	4

B

Distância D4

2	2	2	2	2
2	1	1	1	2
2	1	0	1	2
2	1	1	1	2
2	2	2	2	2

C

Disntância D8

Figura 4: Distância entre pixels

2.3.4. Histograma

O histograma pode ser visto como um gráfico da ocorrência de pixels em relação ao nível de cinza. Ele é uma função dada por:

$p(r_k) = n_k/n$, onde k é o nível de cinza, n_k é a quantidade de pixels neste nível de cinza e n é a quantidade total de pixels da imagem. Alguns exemplos de histogramas estão na Figura 5.

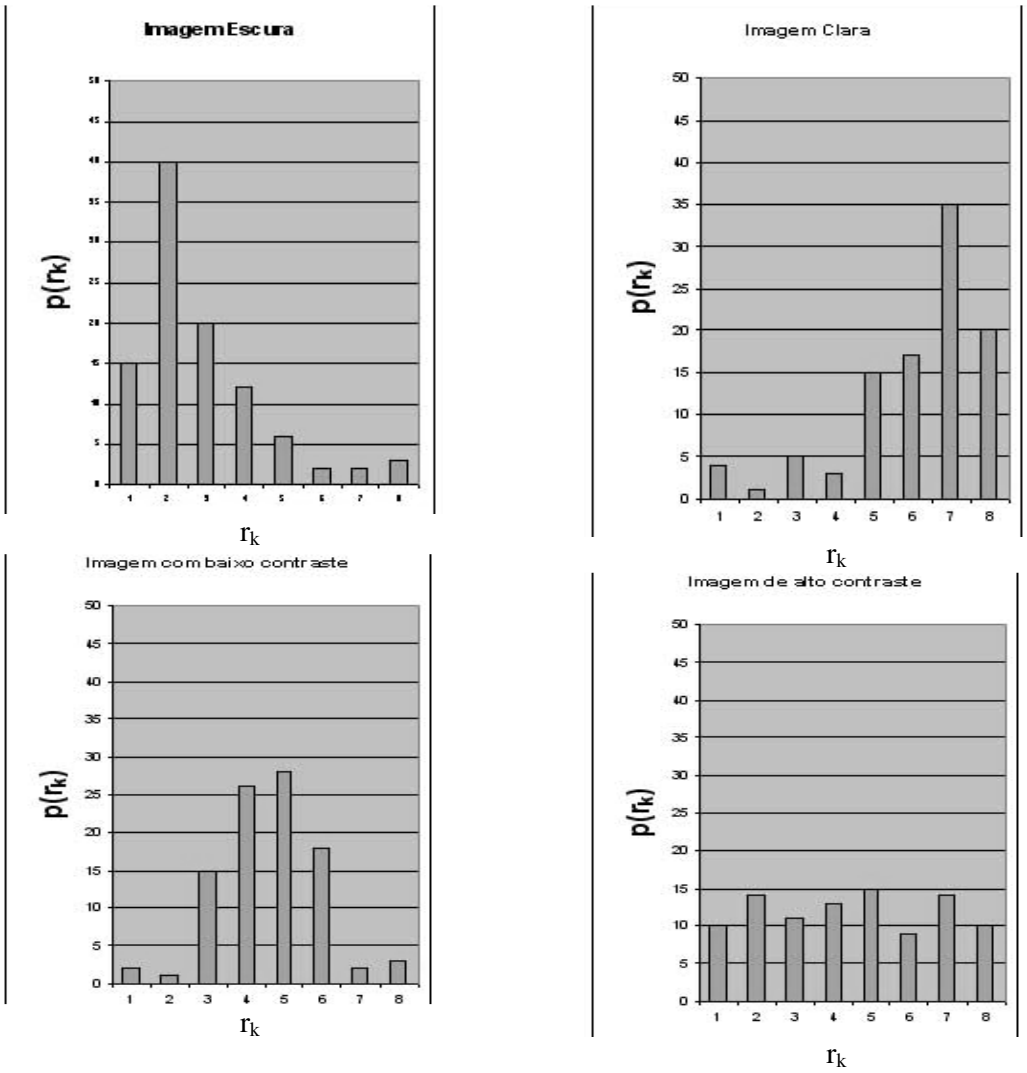


Figura 5: Histogramas

Em uma imagem com 8 bits de profundidade, o meio da profundidade desta imagem está entre o bit 4 e 5, entretanto isto não significa que se a imagem for dividida ao meio (entre os bits 4 e 5) metade das cores ficarão em uma parte da divisão e metade na outra. Para isso é necessário calcular onde está a concentração dos pixels, observando o histograma da imagem é possível notar onde está a concentração dos pixels.

2.3.5. Operações aritméticas e lógicas

2.3.5.1. Filtragem Espacial

Máscaras espaciais são matrizes com valores usados para aplicar determinada ação em imagens. Usar máscaras espaciais para operar imagens é chamado de filtragem espacial, sendo que as máscaras são chamadas de filtros espaciais.

2.3.5.2. Realce

A Figura 6 é a representação de uma porção de uma imagem, sendo z_n os valores dos níveis de cinza dos pixels desta imagem.

z1	z2	z3
z4	z5	z6
z7	z8	z9

Figura 6: Imagem em cinza

Para realçar o ponto z5 algumas operações com os valores dos pixels vizinhos são feitas e o resultado é atribuído ao pixel z5.

$[(z5 - z8)^2 + (z5 - z6)^2]^{1/2}$

Ou podem ser obtidos resultados próximos usando os valores absolutos, ao invés de operações com quadrados e raiz quadrada:

$| z5 - z8 | + | z5 - z6 |$

Outra abordagem usada é a equação de diferenças cruzadas:

$[(z5 - z9)^2 + (z6 - z8)^2]^{1/2}$

Ou com valores absolutos:

$| z5 - z9 | + | z6 - z8 |$

Qualquer uma destes cálculos podem ser feitos, entretanto ao invés de executar esta equação para cada pixel, pode ser usada a implementação por máscaras. A Figura 7 mostra duas máscaras, chamadas de máscaras de Roberts, os valores contidos nestas máscaras serão usados no cálculo do realce (GONZALEZ E WOODS, 2000).

1	0
0	1

0	1
1	0

Figura 7: Máscaras de Roberts

Normalmente a implementação com máscaras quadradas é incomoda, por este motivo a implementação é feita por máscaras 3 x 3 ou 5 x 5. Usando máscaras 3 x 3 a equação usada para a implementação pode ser modificada para:

$$| (z7 + z8 + z9) - (z1 + z2 + z3) | + | (z3 + z6 + z9) - (z1 + z4 + z7) |$$

As máscaras usadas para implementar esta equação podem ser as máscaras de Prewitt ou de Sobel que estão, respectivamente, descritas nas Figuras 8 e 9.

-1	-1	-1
0	0	0
1	1	1

-1	0	1
-1	0	1
-1	0	1

Figura 8: Máscara de Prewitt

-1	-2	-1
0	0	0
1	2	1

-1	0	1
-2	0	2
-1	0	1

Figura 9: Máscara de Sobel

O uso destas máscaras em imagens produz o realce das bordas principais. Se a necessidade for especificamente o reconhecimento de bordas as máscaras de Sobel são mais eficientes, pois elas transformam a imagem em uma imagem binária, onde as bordas ficam com a cor branca e o resto com a cor preta (GONZALEZ E WOODS, 2000).

2.3.5.3. Segmentação de imagens

A segmentação da imagem, geralmente, é o primeiro passo na análise da mesma. A segmentação subdivide a imagens em suas partes. O nível dessa subdivisão varia de acordo com a necessidade da análise.

Por exemplo, em aplicações de análise de alvos terrestres o interesse, entre outras coisas, está na identificação de veículos em uma estrada. A segmentação pode ser feita para que a imagem seja subdividida o suficiente para identificar a estrada e os veículos nela, fugindo a necessidade de continuar a subdivisão até que elementos como os passageiros, ou placas nas estradas sejam encontradas.

Os algoritmos de segmentação para imagens na escala de cinza são geralmente baseados na descontinuidade ou similaridade dos valores de níveis de cinza. A segmentação por descontinuidades é usada na identificação de pontos isolados ou linhas e bordas na imagem. Já a segmentação por similaridade está em cima da limiarização, crescimento de regiões, divisão e fusão de regiões, sendo que também pode ser aplicada em imagens que variam com o tempo (GONZALEZ E WOODS, 2000).

2.3.5.4. Detecção de Linhas

No tópico anterior foi dito como o realce das bordas principais de uma imagem pode ser feito. Neste tópico o realce destas bordas, ou a detecção das linhas, é explicado mais a fundo.

Com a equação $|(z7 + z8 + z9) - (z1 + z2 + z3)| + |(z3 + z6 + z9) - (z1 + z4 + z7)|$, que foi apresentada no tópico 2.3.5.2 é possível implementar o uso das máscaras. Usando a máscara da Figura 10 é possível identificar pontos isolados em uma imagem.

-1	-1	-1
-1	8	-1
-1	-1	-1

Figura 10: Máscara para a identificação de pontos

Para identificar linhas há outras máscaras, porém para cada sentido da linha que a imagem possui é necessário uma máscara diferente, as máscaras estão na Figura 11:

-1	-1	-1
2	2	2
-1	-1	-1

A

-1	-1	2
-1	2	-1
2	-1	-1

B

-1	2	-1
-1	2	-1
-1	2	-1

C

2	-1	-1
-1	2	-1
-1	-1	2

D

Figura 11: Máscaras para a identificação de linhas

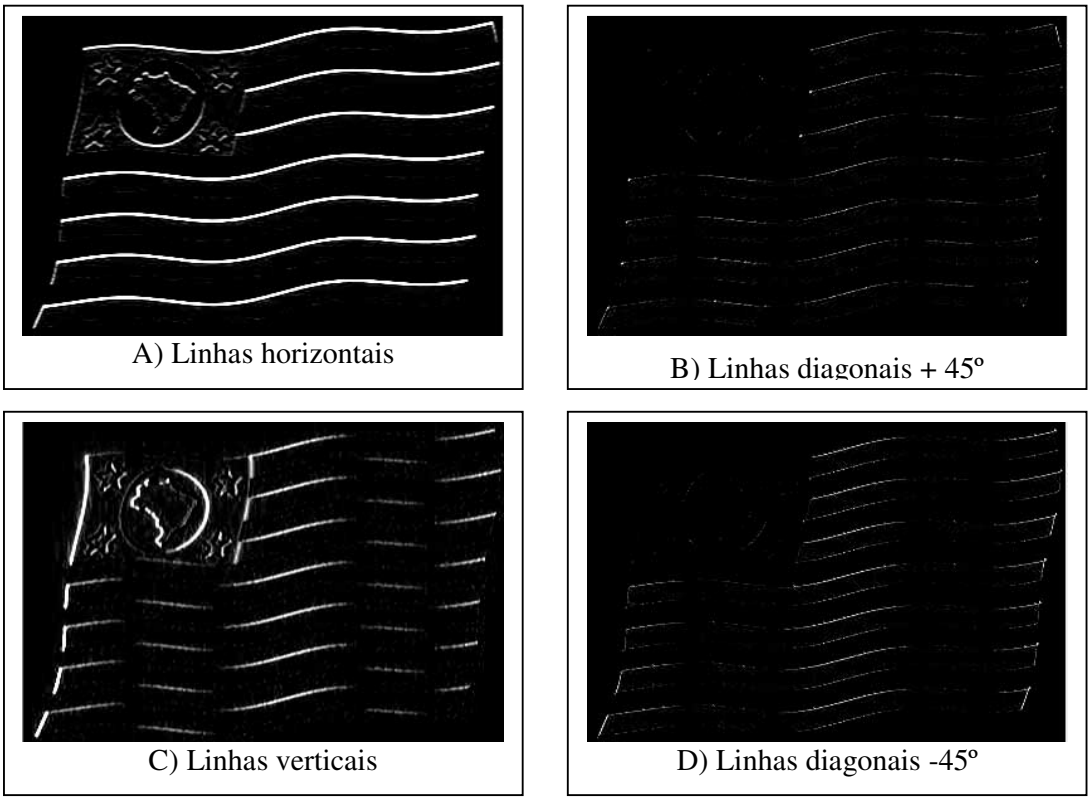


Figura 12: Exemplo de identificação de linhas

Para identificar linhas horizontais deve ser usada a primeira matriz da Figura 11 A, já com linhas verticais a matriz da Figura 11 C é empregada e com as diagonais as matrizes das Figuras 11 B e 11 D são usadas, respectivamente para diagnonais a +45º e -45º. Diagonais de baixo para cima, da esquerda para a direita são chamadas de +45º e as outras, de baixo para cima, da direita para a esquerda são as -45º (GONZALEZ E WOODS, 2000).

Na Figura 12 estão algumas imagens de exemplo na identificação de linhas.

2.3.5.5. Binarização

A binarização é um tipo de *threshold*, ou seja, é um tipo de segmentação da imagem. Binarizar significa reduzir a profundidade de cores da imagem a 1 bit, desde modo ela fica apenas com duas cores: branca e preta. Para realizar a binarização a imagem precisa estar em tons de cinza e é necessário avaliar quais pixels receberão o valor mínimo, 0, e quais o valor máximo, 1. Um valor de corte deve ser escolhido, este valor será usado como critério para definir os novos valores dos pixels da imagem (GONZALEZ E WOODS, 2000).

Por exemplo, uma imagem de profundidade 8 bits de resolução 512 x 256. Cada pixel desta imagem pode assumir o valor de 2^8, o que equivale a 256 tons de cinza. Nesta imagem o tom 137 foi escolhido para ser o valor de corte, então todos os pixels desta imagem com valores menores que 137 terão seus valores atribuídos para 0 e os demais para 1, sendo que 0 indica a cor preta e 1 a cor branca.

A equação aplicada a cada pixel é a seguinte:

f(x,y) = 0, se f(x,y) < 137

1, se f(x,y) >= 137

A binarização de imagens é utilizada para isolar e identificar elementos.

2.3.5.6. Dilatação, erosão, abertura e fechamento

Estas quatro operações são usadas para modificar a forma dos elementos de uma imagem. A dilatação e fechamento são usados para tapar “buracos”, arrumar descontinuidades ou falhas nos elementos. A erosão e abertura são usadas para aumentar “buracos”, descontinuidades ou falhas nos elementos da imagem.

A Figura 13 A e 13 B são imagens que serão usadas para exemplificar a dilatação e a erosão de imagens. A Figura 13 A sofrerá alterações pela Figura 13 B.

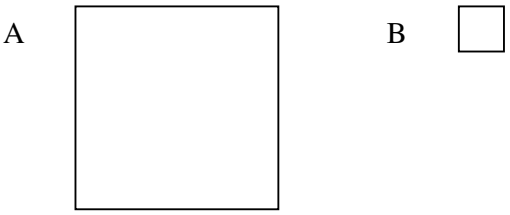


Figura 13: Imagens antes das operações

A dilatação de A por B é denotada por $A \oplus B$ e resulta na Figura 14:

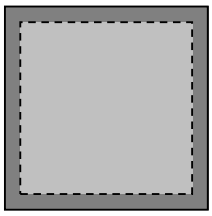


Figura 14: Imagem A dilatada por B

Em que a área interna clara é a imagem A e porção escura é o resultado da dilatação pela imagem B.

A erosão de A por B é denotada por $A \ominus B$ e resulta na Figura 15:

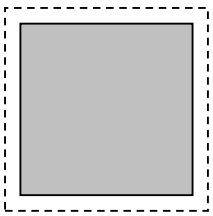


Figura 15: Imagem A erodida por B

Onde a imagem clara é a imagem resultante, sendo que a parte tracejada é a porção da imagem A que foi “apagada” devido a erosão por B.

A abertura de uma imagem consiste na erosão seguida de uma dilatação e é representada pela imagem B.

$$A \circ B = (A \ominus B) \oplus B$$

O fechamento de uma imagem é a dilatação seguida da erosão e é representada por:

$$A \bullet B = (A \oplus B) \ominus B$$

O fechamento tem a função de fechar pequenas falhas e alterar minimamente grandes falhas.

2.4. Considerações finais

O processamento de imagens é o foco teórico deste trabalho, as operações com imagens apresentadas neste capítulo foram usadas na implementação do software que é o principal fruto deste trabalho.

O processo de avaliação da porcentagem necrosada do tecido animal se resume basicamente em aplicar a operação de binarização, entretanto é necessário aplicar várias outras operações para realizar a remoção do fundo da imagem.

O fundo da imagem, ou seja, o cenário que compõe a imagem deve ser removido para haver um diagnóstico mais preciso. Esta remoção torna-se difícil e trabalhosa de implementar em algumas imagens, que serão exemplificadas mais adiante, entretanto para a maioria das imagens usadas neste trabalho a remoção por meio das operações apresentadas neste capítulo foi eficiente.

3. JAVA

A linguagem de programação Java surgiu de uma pesquisa financiada pela Sun Microsystem. Esta pesquisa tinha o codinome Green em 1991 e resultou no desenvolvimento do Java que é uma lingaguem baseada em C e C++.

Java inicialmente era destinado ao provável futuro mercado dos dispositivos eletrônicos inteligentes destinados aos usuários finais, entretanto o desenvolvimento destes dispositivos naquela época não estava crescendo com a rapidez do Java e o projeto sofreu risco de cancelamento. Por sorte em 1993 a popularidade da World Wide Web explodiu e a equipe da Sun viu o potencial de utilizar Java para criação de páginas Web com conteúdo dinâmico.

Java foi anunciado formalmente em 1995 em uma conferência importante e causou imediato interesse da comunidade comercial devido ao poder de aplicar Java em páginas Web. Atualmente a linguagem também é empregada em aparelhos como celulares, pagers, entre outros (DEITEL, 2003).

Os programas Java são constituídos por classes que possuem métodos que realizam operações e podem retornar um resultado. Java possui uma API (Applications Programming Interfaces) que é um conjunto de classes disponibilizado para toda a comunidade. O programador pode usar, para realizar determinada tarefa, alguma classe existente na API Java, evitando assim a necessidade de ter que fazer esta programação. Isto torna o desenvolvimento de uma aplicação mais rápido e eficiente (DEITEL, 2003).

A API Java, que também pode ser chamada de biblioteca Java, é mantida e fornecida por desenvolvedores de compiladores e também por fornecedores independentes.

Por exemplo, se você quer converter a string “1123” para um número inteiro, você não precisa fazer uma classe com um método que faça esta operação, basta você usar o

método `parseInt` da classe `Integer` que faz parte da API Java. Deste modo pequenos detalhes que consomem um tempo considerável e precioso no desenvolvimento de aplicações é poupado.

3.1. JAI

A manipulação de imagens em Java pode ser feita através da JAI (*Java Advanced Imaging*) que é uma API com classes e métodos que implementam as operações lógicas e aritméticas que envolvem a manipulação de imagens.

A API JAI poupa muito esforço por parte do desenvolvedor. Por exemplo, em uma aplicação é necessário realizar o realce de imagem. Para isto o programador tem a necessidade de implementar algum método dentro da sua aplicação que realize o realce, entretanto isto lhe custaria algum esforço extra. Se o programador usar a API JAI e seus métodos neste software, será possível realizar o realce rapidamente.

No site da Sun Microsystem é possível fazer o download da API JAI. Existe uma versão para a plataforma Microsoft Windows e outra para sistemas GNU/LINUX.

Há a possibilidade de acessar apenas a porção de pixel necessária para as operações, reduzindo assim o custo computacional das operações. Atualmente a JAI conta com mais de 100 operações de processamento de imagem nos formatos: Byte, UShort, Short, Inteiro de 32 bits e Floats/Double.

Como a JAI é construída na plataforma Java torna-se mais fácil a síntese de aplicações colaborativas para processar imagens e construir visualizadores de alto nível para um computador local, uma rede local ou para a Internet.

A JAI consegue ler e escrever os padrões BMP, GIF, FPX (FlashPix), JPEG, PNG, PNM, TIFF e WBMP. Essas informações de entrada e saída (E/S) atualmente estão bem desenvolvidas. Também há suporte para o método remoto de invocação (RMI – *Remote*

Method Invocation) e a *Internet Imaging Protocol*(IIP). E é possível a recuperação da imagem usando InputStream ou URL.

A habilidade de misturar imagens e gráficos é outro ponto forte da JAI. Com ela é possível interagir com a API Java 2D para misturar gráficos de duas dimensões com imagens comuns. A integração da API JAI com a API Java 2D é fácil de ser realizada. Esta facilidade faz a JAI se tornar ainda mais poderosa, já que se nela não houver alguma funcionalidade dá para suprir essa falta com a API Java 2D (WEB SITE JAI, 2005).

3.1.1. Armazenamento e representação

Na JAI a classe básica para representar uma imagem é a classe `PlanarImage`, esta classe possui mais flexibilidade do que a classe Java `BufferedImage`.

A classe `PlanarImage` armazena os elementos da imagem, pixels, em outra classe chamada `Raster`. A Raster contém uma instância de uma outra classe, a `DataBuffer`. Por sua vez a `DataBuffer` é moldada segundo as regras da classe `SampleModel` que define, entre outras coisas, se a imagem será colorida ou cinza. `ColorModel`, que contém uma instância da classe `ColorSpace`, é outra classe que se associa à `PlanarImage`. Estas classes são usadas para a JAI representar uma imagem na memória.

A `PlanarImage` não possui métodos capazes de modificar os valores dos pixels que ela contém, ou seja, ela é uma classe somente leitura. Para a alteração dos pixels deve ser usada uma subclasse da `PlanarImage`, a `TiledImage`.

A implementação na JAI de "*Tiled Images*" diz que uma imagem pode ser acessada como um conjunto de imagens menores, por exemplo, uma imagem de 1024 x 512 pixels pode ser acessada como 4 imagens de 256 x 128 pixels. Isto é uma enorme vantagem em relação a imagens grandes, ou quando não há muitos recursos computacionais disponíveis ou

mesmo quando há a necessidade de manipular apenas uma região específica da imagem (SANTOS, 2004).

Com a API JAI é possível criar imagens a partir de matrizes de valores, entretanto é mais comum ler imagens armazenadas em arquivos em um meio físico (HD, CD-ROM, etc).

3.1.2. Operações com imagens

A Figura 16 mostra o método `create`, que é o método principal da classe JAI.

```
saida = JAI.create(parâmetros, ...);
```

Figura 16: Uso da API JAI

Este método tem como primeiro parâmetro a instrução de qual será seu comportamento e o resto dos parâmetros variam conforme o comportamento do método. A Figura 17 mostra um exemplo de como ler uma imagem de um arquivo e inverter os valores (tons) dos pixels:

```
PlanarImage entrada = JAI.create("fileload", caminho);
PlanarImage saida = JAI.create("invert", entrada);
```

Figura 17: Abertura e inversão de uma imagem com a API JAI

Alguns operadores precisam de outros valores de entrada, além da imagem que será processada como no da inversão de tons, mostrado na Figura 17. Esses valores são passados por uma classe chamada `ParameterBlock` (WEB SITE DOCUMENTAÇÃO JAI, 2005).

A Figura 18 tem um exemplo do uso do `ParameterBlock`, onde uma imagem tem seu tamanho dobrado na largura e triplicado na altura.

```
ParameterBlock pb = new ParameterBlock();                    //Linha 1
pb.addSource(imagenEntrada);                                 //Linha 2
pb.add(2.0f);                                                //Linha 3
pb.add(3.0f);                                                //Linha 4
pb.add(0.0f);                                                //Linha 5
pb.add(0.0f);                                                //Linha 6
pb.add(new InterpolationNearest());                          //Linha 7
PlanarImage imagemSaida = JAI.create("scale", pb);           //Linha 8
```

Figura 18: Escala de uma imagem com a API JAI

Ao todo o `ParameterBlock pb` recebe 6 argumentos. O primeiro parâmetro adicionado é a imagem que será tratada (linha 2), os próximos dois são os valores da escala (linhas 3 e 4), os dois seguintes (linhas 5 e 6) são os valores da translação e o último (linha 7) é uma instância da classe `InterpolationNearest` que fará a interpolação dos pixels durante o processo de escalar a imagem.

3.1.3. Visualização

A API JAI possui uma classe, `DisplayJAI`, que serve para visualizar as imagens contidas em instâncias das classes `PlanarImage` e `TiledImage`. A classe `DisplayJAI` é simples, mas pode ser estendida e usada para implementar alguma outra classe mais complexa e melhor para visualizar imagens (SANTOS, 2004).

Para a visualização de uma imagem use o código da Figura 19, onde "`imagem`" é uma instância de `PlanarImage` ou de `TiledImage`

```
DisplayJAI mostraIMG = new DisplayJAI(imagem);
```

Figura 19: Visualização de uma imagem

3.1.4. Formato

Uma imagem em JAI (`PlanarImage` ou `TiledImage`) pode ser dos seguintes tipos: BYTE, SHORT, USHORT, INT, FLOAT e DOUBLE. Este formato molda o `DataBuffer` onde os pixels são armazenados e indica o tipo que os pixels serão armazenados (WEB SITE TIPO DE DADOS JAI, 2005).

No momento da criação da classe `SampleModel` este formato deve ser especificado. Entretanto, se a imagem é aberta de um arquivo não é possível especificar e nem fará diferença.

No caso de passar uma imagem (`PlanarImage` ou `TiledImage`) para uma matriz, no momento da passagem é possível especificar o formato de "saída" dos pixels. E quando uma matriz contendo os pixels for passada para uma classe `PlanarImage` ou `TiledImage` os formatos delas devem estar de acordo.

3.1.5. Tipos de arquivos

Atualmente a API JAI oferece suporte de leitura e escrita de arquivos em vários formatos, alguns ainda não possuem codificadores para a escrita, o que provavelmente será solucionado no futuro pela equipe desenvolvedora. Estes formatos são padronizados mundialmente e suportados na API JAI para que ela ofereça maior flexibilidade na manipulação de imagens (WEB SITE ENTRADA E SAIDA COM JAI, 2005).

Os formatos suportados pela API JAI estão listados abaixo:

- BMP: a versão do Windows 95 do formato BMP é suportada.
- FlashPix: o decodificador deste formato está parcialmente implementado e o codificador não está disponível. Em outras palavras este formato pode ser lido de um arquivo, mas nenhuma imagem neste formato pode ser salva.

- GIF: o decodificador suporta GIF animados e arquivos GIF com fundo transparente. Apenas a primeira imagem do arquivo GIF animado será carregada pela JAI, o resto das imagens será ignorado e deverá ser lido por decodificadores auxiliares se for necessário.
- JPEG: este formato é suportado inteiramente pela JAI.
- PNG: o codificador deste formato determina automaticamente o tipo da imagem a ser codificada (RBG, Escala de cinza, ou Cores personalizadas). Isto tudo é obtido a partir da imagem fornecida (PlanarImage ou TiledImage).
- PNM: este tipo de arquivo pode ter dados em ASCII ou em RAW (Binário). O codificador automaticamente determina qual o formato correto.
- TIFF: este é formato possui várias opções, contudo mais opções serão adicionadas no futuro. Com ele é possível ler imagens de qualquer tipo (FLOAT, BYTE, etc). Imagens grandes como as de ponto flutuante de 32-bits são suportadas. O decodificador pode descompactar imagens no formato de compactação LZW.
- WBMP: o codificador e decodificador deste formato le e escreve imagens no formato Wireless Bitmap que está descrito no capítulo 6 e apendice A do "Wireless Application Protocol (WAP)" em 29 de Março de 2000. O tipo de WBMP suportado é o WBMP 0 com Bitmap descomprensados. Não há limitações nas dimensões da imagem.

3.2. Considerações finais

Java é uma linguagem poderosa e excelente para implementar aplicações que necessitam de uma boa interação com o usuário. Uma vez que Java possui uma biblioteca de classes pronta não é necessário gastar muito tempo com o design da aplicação. Como por

exemplo, programar botões, menus ou mesmo as funções básicas de uma janela no sistema operacional (minimizar, maximizar e fechar).

Outro ponto forte de Java e também da sua biblioteca é outros métodos, além dos que dizem respeito à interface gráfica, que permitem o desenvolvimento mais acelerado da aplicação, uma vez que não é necessário realizar uma programação de classes e métodos que já estão prontos.

Os motivos principais da escolha de Java para a implementação do software deste trabalho são citados a seguir:

- É uma ótima e poderosa linguagem orientada a objeto.
- Possui documentação abundante.
- É uma linguagem utilizada atualmente.
- Não precisa de investimento financeiro na aquisição de licenças.
- É livre de plataforma operacional.

A JAI, por ser também uma biblioteca Java, dá ao programador a vantagem de não precisar implementar várias páginas de teoria para realizar operações com imagens. Atualmente a JAI é usada na análise e processamento de fotografias digitais em pesquisas biomédicas, processamento de dados geo-espaciais, defesa e inteligência. Futuramente ela poderá ser aplicada em operações com imagens nos Servicos da Web (*Web Services*).

Neste trabalho a implementação das operações com imagens foi feita com o auxílio da API JAI, sendo que apenas uma pequena parte foi implementada sem a ajuda da JAI.

Java e JAI foram escolhidas para a implementação do software neste trabalho, principalmente pelo motivo de Java possuir uma biblioteca rica em classes e a JAI ser uma biblioteca Java específica para a manipulação de imagens

A API Java, junto com a API JAI formam um conjunto poderoso e perfeito para o desenvolvimento e implementação da aplicação proposta. No caso a implementação foi feita

com a ajuda do ambiente de desenvolvimento Eclipse SDK 3.0.1 da Eclipse Foundation (www.eclipse.org), mas não necessita deste ambiente para ser compilada ou executada, basta apenas o J2SE 1.5 e a API JAI 1.2 para compilar ou o JRE 1.5 e a API JAI 1.2 para apenas executar as classes já compiladas.

4. IMPLEMENTAÇÃO

4.1. Introdução

O software implementado neste trabalho é uma aplicação Java independente de plataforma e relativamente de baixo custo computacional. A implementação foi feita com base nos estudos abordados neste trabalho e com informações vindas da documentação da API JAI encontrada no site da Sun Microsystem.

O resultado esperado da implementação era a síntese de uma aplicação capaz de processar imagens de tecido animal e avaliar qual a porcentagem necrosada do tecido. Entretanto o tecido deve passar previamente por um tratamento, este tratamento será explicado no tópico 5.1 deste trabalho.

Esta aplicação foi testada nas plataformas Microsoft Windows XP e GNU/LINUX. Na verdade ela deve funcionar em qualquer plataforma, uma vez que foi desenvolvida em Java que é uma linguagem independente de plataforma, desde que nesta plataforma exista a instalação do JRE 1.5 e da JAI 1.2, mais informações sobre estas instalações podem ser obtidas no site da Sun Microsystem (http://java.sun.com).

4.2. Metodologia

Para a implementação da aplicação e obtenção do resultado esperado foram empregados os seguintes passos:

- Escolha do arquivo que contém a imagem que será analisada.
- O cálculo da largura e altura da imagem.
- Transformação para a escala de cinza.
- Remoção do fundo (cenário) que compõe a imagem, por meio dos passos:

- Identificação das bordas na imagem.
- Binarização da imagem.
- Dilatação da imagem.
- Identificação da borda principal da imagem.
- Limpeza dos pixels correspondentes ao cenário na imagem.

- Contagem da Necrose
  - Restauração dos pixels da imagem original correspondente ao tecido animal.
  - Binarização da imagem.
  - Contagem da Necrose no tecido animal.

Os tópicos a seguir explicam com mais detalhes as operações citadas acima.

4.3. Escolha do arquivo

O arquivo pode estar armazenado em qualquer tipo de mídia, porém deve estar nos formatos suportados pela JAI, uma vez que a abertura da imagem é feita por ela. A Figura 20 mostra como um arquivo é aberto usando a JAI.

```
PlanarImage imagem = JAI.create("fileload", caminho da imagem);
```

Figura 20: Abertura da imagem

4.4. O cálculo da largura e altura da imagem

Este cálculo é feito para evitar a manipulação de imagens grandes. Logo após a imagem ser aberta é verificado se sua largura ou altura ultrapassa um valor previamente determinado (o padrão é 800 pixels, mas pode ser alterado). Se ultrapassar, a imagem tem seu tamanho reduzido de forma proporcional, ou seja, se a largura for reduzida em 20% a altura também é. Isto evita distorções.

Com este procedimento a análise da imagem se torna mais rápida para imagens grandes, já que elas terão seu tamanho reduzido. A dúvida seria se esta redução não atrapalha na identificação da necrose no tecido animal. É certo que quanto maior a imagem, mais preciso será o resultado, entretanto 800 pixels é um tamanho razoável e se for do desejo do usuário este valor pode ser alterado. E se a imagem for menor do que este valor ela não terá o seu tamanho aumentado.

4.5. Transformação para a escala de cinza

A imagem é transformada em escala de cinza por dois motivos principais:

1. Necessidade para algumas das operações posteriores.

2. Redução do espaço utilizado na memória e conseqüente diminuição de custo computacional.

As imagens abertas são coloridas, caso não sejam este passo deve ser pulado. A API JAI possui um método que faz a combinação das bandas RGB (Vermelho, Verde e Azul) para obter imagens em tons de cinza. O algoritmo desta combinação está na Figura 21.

```
double[][] matriz = { {
        (double)b,
        (double)g,
        (double)r,
        0.0D} };
ParameterBlock pb = new ParameterBlock();
pb.addSource(imagemColorida);
pb.add(matriz);
imagemCinza = JAI.create("bandcombine", pb, null);
```

Figura 21: Combinação das bandas RGB

São instâncias da classe `PlanarImage` os objetos `imagemColorida` e `imagemCinza`.

A matriz contém os pesos das cores do modelo RGB, dependendo dos valores atribuídos na matriz uma cor pode influenciar mais que outra na conversão para a escala cinza.

Por exemplo, considere uma imagem do pôr do sol que possui vários tons dependentes do plano R (vermelho), se esta imagem for passada para tons de cinza e na matriz o valor de `r` for maior do que o de `g` e `b` esta imagem ficará clara. Entretanto se o valor de `r` for menor a imagem ficará escura.

4.6. Remoção do fundo (cenário) que compõe a imagem

A parte mais trabalhosa da implementação foi a remoção do cenário que está presente da imagem, uma vez que a fotografia do tecido animal possui um fundo, os pixels que compõe este fundo devem ser identificados para que sejam desconsiderados na análise da necrose.

4.6.1. Identificação das bordas na imagem

O primeiro passo para a remoção do fundo é a identificação dos elementos que compõe a imagem. Para isto é feito o realce das linhas (bordas) da imagem. A API JAI oferece um método capaz de identificar as bordas principais da imagem, na verdade o método aplica máscaras na imagem e dependendo da máscara usada o realce da borda é obtido, a demonstração está na Figura 22.

Na aplicação para realçar as bordas foi usada a máscara de Sobel que foi explicada no tópico 2.3.5.2 que trata sobre realce da imagem.

```
float[] kernelMatrix = {-1, 0, 1, -2, 0, 2, -1, 0, 1};
KernelJAI kernel = new KernelJAI(3, 3, kernelMatrix);
imagemSaida = JAI.create("convolve", imagemEntrada, kernel);
```

Figura 22: Identificação das bordas

A `imagemEntrada` e `imagemSaida` são instâncias da classe `PlanarImage` e `kernelMatrix` é a máscara de Sobel. Os valores da máscara podem ser aumentados proporcionalmente para que a identificação das bordas seja mais "sensível".

4.6.2. Binarização da imagem

Binarizar uma imagem significa deixá-la com dois valores apenas: Branco e Preto. A teoria sobre binarização é explicada no tópico 2.3.5.4.

Após a identificação das bordas a imagem está dividida em dois tipos de elementos: os com tons claros (as bordas) e os com tons escuros.

Para haver um melhor reconhecimento a binarização é aplicada e então as bordas são transformadas em Branco (1) e o resto da imagem em Preto (0).

```
double[] threshold = histogram.getPTileThreshold(valor);
imagemSaida = JAI.create("binarize", imagem, new Double(threshold[0]));
```

Figura 23: Binarização

A variável `valor` varia de 0 a 100 e através dela é obtido um valor no histograma da imagem (previamente calculado), este valor é o limiar (o valor de corte).

O método `JAI.create` recebe como parâmetro o comando para binarizar (`binarize`), a imagem de entrada e o valor de corte. Então a JAI se encarrega em transformar em preto os pixels com os tons abaixo do valor de corte e em branco os demais. E como saída há a imagem binarizada.

4.6.3. Dilatação da imagem

No tópico 2.3.5.5 foi apresentado a dilatação, erosão, abertura e fechamento dos elementos de uma imagem. Na implementação foi testado o uso destes quatro itens e foi constatado que o fechamento tinha um resultado melhor para fazer a conexão de descontinuidades na borda principal (a borda que separa o tecido animal do cenário).

O processo de fechamento serve para conectar pequenas falhas nas linhas (bordas) identificadas na imagem. Esta correção não cobre grandes descontinuidades, mas é suficiente para as pequenas que geralmente o usuário deixa passar. Para realizar a dilatação pode ser usado o código da Figura 24 e para realizar a erosão o código da Figura 25 é usado.

```
float[] kernelMatrix =  { 0, 0, 0, 0, 0,
            0, escala, escala, escala, 0,
            0, escala, escala, escala, 0,
            0, escala, escala, escala, 0,
            0, 0, 0, 0, 0};
            KernelJAI kernel = new KernelJAI(5,5,kernelMatrix);
pb.addSource(imagem);
pb.add(kernel);
imagem = JAI.create("dilate", pb, null);
```

Figura 24: Dilatação

```
pb.addSource(imagem);
pb.add(kernel);
imagem = JAI.create("erode", pb, null);
```

Figura 25: Erosão

No código apresentado `kernelMatrix` é a máscara usada para realizar a dilatação e a erosão. Na verdade `kernelMatrix` é considerada uma outra figura e é por esta figura que são feitas as operações de dilatação e erosão. Neste tópico o que é chamado de `kernelMatrix` é chamado de “imagem B” no tópico 2.3.5.5.

4.6.4. Identificação da borda principal da imagem.

A borda principal é a borda que separa o tecido do animal do cenário. Na imagem, provavelmente várias bordas serão identificadas, entretanto a única que interessa é a principal. Para o software identificar esta borda o usuário deve clicar nela.

Quando o clique acontece um método é chamado, este método analisa o pixel clicado e seus vizinhos-de-8, procurando por conectividade entre eles. O critério de conectividade

usado é a cor do vizinho-de-8 ser branca. A conectividade é explicada em detalhes no tópico 2.3.3.2 e a vizinhança dos pixels está detalhada no tópico 2.3.3.1

Aqui está a grande vantagem da imagem estar binarizada, se ela não estivesse a procura deveria ser por um conjunto de cores claras. Procurar por um conjunto de cores é certamente mais custoso do que procurar por apenas uma única cor (a branca), que é a cor da borda.

A imagem é convertida para o padrão RGB novamente, mas as cores preta e branca são mantidas. Essa mudança acontece para destacar os pixels conectados que foram encontrados. Estes pixels são mudados para a cor azul.

4.6.5. Limpeza dos pixels correspondentes ao cenário na imagem.

Neste momento a imagem está dividida na porção interna e externa à borda principal. Ao menos o objetivo principal é chegar neste ponto com a borda principal totalmente conectada. Caso isto não ocorra, o usuário deve começar o processo novamente ajustando os valores na conversão para cinza, identificação das bordas, binarização e dilatação para que a borda principal esteja toda conectada.

Para finalizar a remoção do cenário da imagem o usuário deve clicar na porção externa à borda principal, teoricamente o conteúdo interno à borda é o tecido animal.

Com este clique outro método é chamado. Este método procura por vizinhos-de-8 que satisfaçam o critério de estar com a cor branca ou preta. Desta maneira todos os pixels externos à borda serão reconhecidos como conectados e em seguida eles serão mudados para a cor vermelha para serem diferenciados mais facilmente.

Ao fim destes procedimentos a imagem possuirá três partes: o cenário com os pixels vermelhos, a borda principal com os pixels azuis e a parte interna da imagem com pixels pretos e brancos.

4.7. Contagem da Necrose

4.7.1. Restauração dos pixels da imagem original correspondente ao tecido animal.

Com o cenário restaurado a única tarefa antes de iniciar o processo de contagem da necrose é restaurar os pixels correspondentes à porção do tecido da imagem original para que o usuário os visualize na aplicação.

Isto é feito com um método simples que copia para a imagem principal os pixels da imagem original quando o pixel da imagem principal for diferente da cor vermelha.

4.7.2. Binarização da imagem.

Após a restauração dos pixels do tecido, uma outra binarização é aplicada nestes pixels restaurados. Esta operação visa transformar em preto os pixels escuros e em branco os claros. Sendo deste modo feito a diferenciação dos pixels necrosados dos demais.

Os pixels necrosados ficarão com a cor preta e neste ponto o usuário deve interferir para ajustar o valor de corte para que seja identificada a quantidade correta de pixels necrosados. Para uma eficiência maior o usuário deve comparar a imagem binarizada com a imagem original, fazendo assim os ajustes necessários no valor de corte.

4.7.3. Contagem da Necrose no tecido animal.

Após o ajuste do valor de corte são feitas as contagens dos pixels do tecido animal e dos pixels necrosados (pretos), restando apenas uma divisão para estipular quanto é a porcentagem necrosada, repare na Figura 26.

Porcentagem necrosada = (Pixels Necrosados / Total de Pixels) * 100

Figura 26: Cálculo da necrose

4.8. Considerações finais

A implementação foi bem sucedida principalmente pelas facilidades providas pelas bibliotecas Java, entre elas especialmente a JAI, que com suas classes e métodos ofereceram suporte à maioria dos procedimentos que foram usados.

Sem a JAI toda teoria vista no capítulo 2 teria que ser implementada na integra. Claro que todas essas operações foram usadas neste trabalho, entretanto a implementação destas operações ficou, na maior parte, por conta da biblioteca JAI.

A implementação visou o desenvolvimento de um aplicativo orientado à objeto de fácil manutenção e de acompanhamento por parte do usuário do que está sendo feito. Isto implicou em um software de difícil manuseio à primeira vista, contudo isto é facilmente sanado com o próximo capítulo, onde é dado um exemplo do seu uso.

O usuário também conta com a possibilidade de usar a opção `Automatizar` do menu `Automático`, onde todas as operações serão feitas automaticamente, com o mínimo de interação com o usuário. Os parâmetros usados nesta opção podem ser configurados também pelo menu `Automático`, no item `Configurar`.

5. ESTUDO DE CASO

Neste capítulo há ilustrações passo a passo sobre o uso do software, desde a abertura da imagem até o calculo da porcentagem necrosada.

5.1. OBTENÇÃO DAS IMAGENS

As imagens usadas neste trabalho são fotografias digitais, porém antes dos tecidos animais serem fotografadas por câmeras digitais é necessário prepará-los junto com o cenário.

Nos tecidos é usado um produto químico que escurece as regiões necrosadas, tornando possível a identificação das mesmas pelos tons de cores, o que facilita muito a implementação do software.

A aplicação deste produto químico é feita por pessoas de outras áreas, no caso das imagens usadas neste trabalho a aplicação foi feita por uma equipe da Faculdade de Medicina de Marília.

A preparação do cenário para tirar a fotografia também é muito importante. Os elementos que compõe o cenário podem ser identificados nas etapas seguintes como parte do tecido, interferindo no resultado final. Portanto o local onde o tecido animal será colocado, provavelmente uma mesa, deve ser de cor sólida, ou seja, não deve possuir riscos, frestas ou gravuras. A cor usada no fundo pode ser qualquer uma, porém é melhor se for um tom diferente do tecido animal, por exemplo, de cor azul, amarela ou branca.

Talvez seja necessário usar algum tipo de grampo para prender o tecido animal no fundo, entretanto este procedimento é desaconselhável, porque estes elementos podem ser identificados como parte do tecido animal e a remoção deles é difícil no processo de análise da imagem

5.2. Exemplo passo a passo

5.2.1. Abertura da aplicação

A aplicação possui três menus: `Automático`, `Comandos` e `Janela`.

O menu `automático` implementa uma forma mais automática do tratamento da imagem, onde o usuário tem o mínimo de interação com o processo, mas também tem um grau de controle menor do resultado.

O menu `Comandos`, como pode ser visto na Figura 27, possui um item para cada passo usado no tratamento da imagem, deste modo é possível o usuário acompanhar e ajustar os parâmetros do tratamento.

O menu `Janelas` provê um sistema por onde as imagens abertas podem ser acessadas em ordem cronológica de abertura.

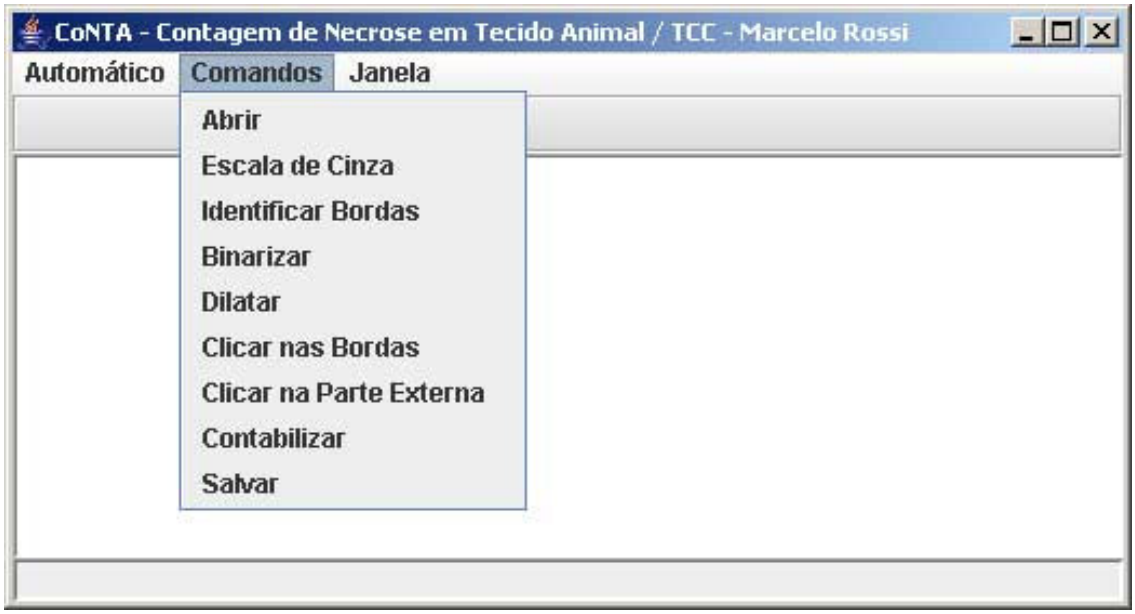


Figura 27: Menu Comandos da aplicação

5.2.2. Escolha do arquivo que contém a imagem que será analisada.

A escolha é simples e não há muito que comentar. Basta acessar o menu `Comandos`, a opção `Abrir` e então escolher o arquivo em qualquer unidade do sistema como mostrado na Figura 28.

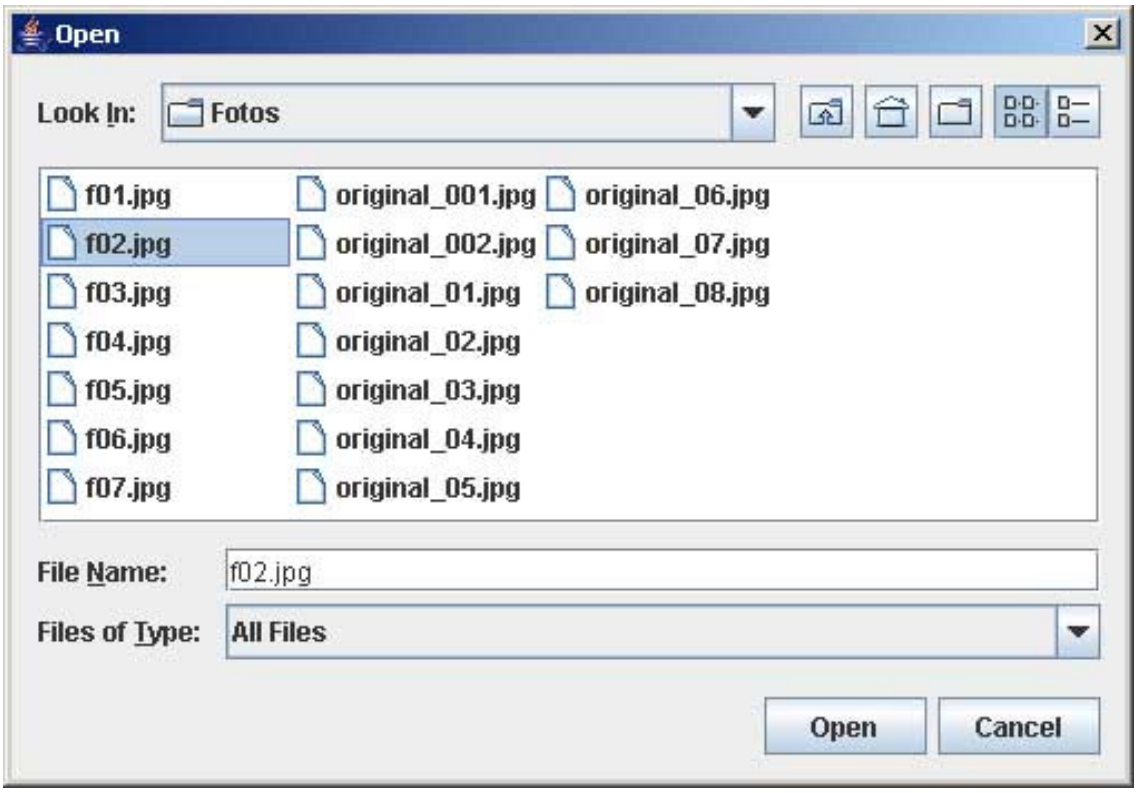


Figura 28: Abertura de um arquivo

A recalculagem da largura e altura da imagem acontece logo após a escolha da imagem e é feita de forma automática, não necessitando da intervenção do usuário e de nenhum comando extra.

5.2.3. Transformação para a escala de cinza.

A transformação em cinza é acionada no menu `Comandos`, opção `Escala de Cinza`. O painel de controle para esta opção aparece na parte superior da janela e nele o usuário pode regular os pesos das três cores primárias, RGB.

O usuário deve clicar no botão de `OK` do painel de controle para confirmar a mudança, se não fizer isto ele não poderá avançar para os próximos procedimentos. E a qualquer momento o usuário pode cancelar este procedimento com o botão `Cancelar` do painel de controle. Isto vale para os próximos passos, neles o usuário também deve clicar no botão `OK` localizado no painel de controle para que o tratamento possa continuar, observe a operação na Figura 29.

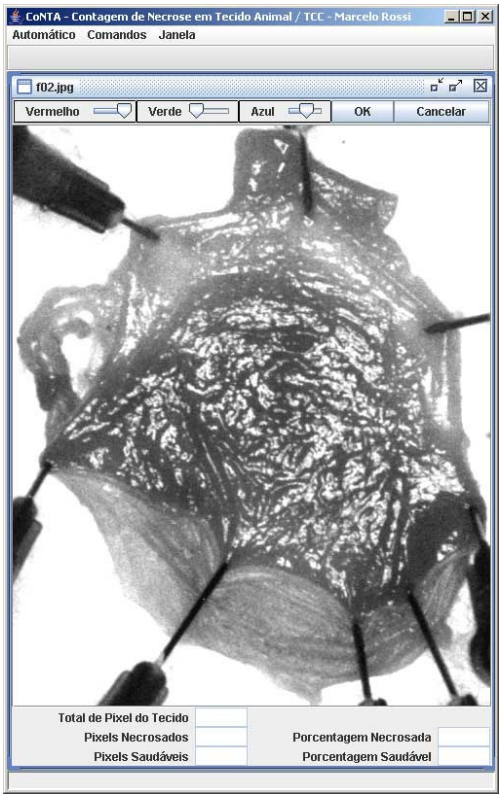


Figura 29: Conversão para escala de cinza

5.2.4. Remoção do fundo (cenário) que compõe a imagem.

5.2.4.1. Identificação das bordas na imagem.

A identificação da borda é acionada pelo menu `Comandos`, opção `Identificar Bordas`. O painel de controle da borda aparece e através dele é possível regular a sensibilidade da identificação das bordas. O ideal é deixar esta regulagem o menor possível para que apenas os objetos mais importantes sejam detectados, mas não a ponto de aparecer descontinuidades na borda principal.

A confirmação do término desta operação também deve acontecer com o clique no botão OK do painel de controle, o exemplo está na Figura 30.

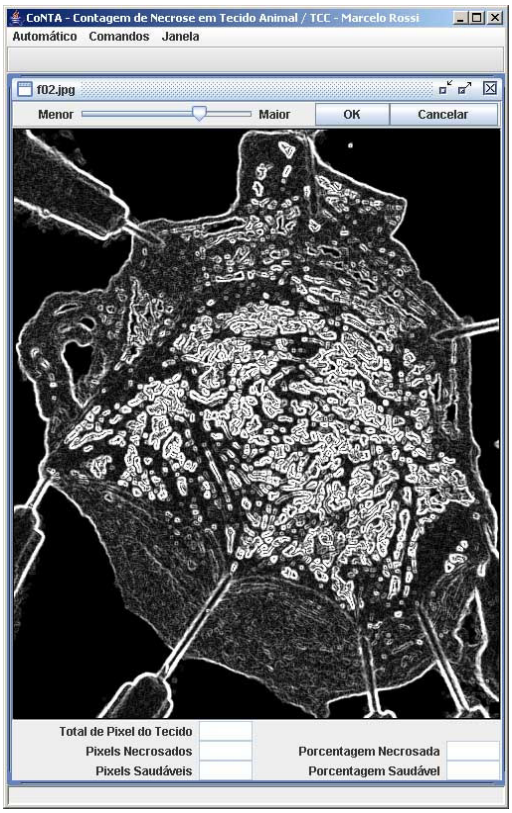


Figura 30: Identificação das bordas

5.2.4.2. Binarização da imagem.

A binarização é acionada no menu `Comandos`, opção `Binarizar` e deve ser ajustada pelo painel de controle a ponto de não deixar descontinuidades na borda principal da imagem. Há uma significante diferença entre a Figura 31 onde o usuário não ajustou corretamente a binarização e a Figura 32 onde o ajuste foi feito de maneira mais sensata e a borda principal ficou sem descontinuidades.

Figura 31: Binarização mal ajustada

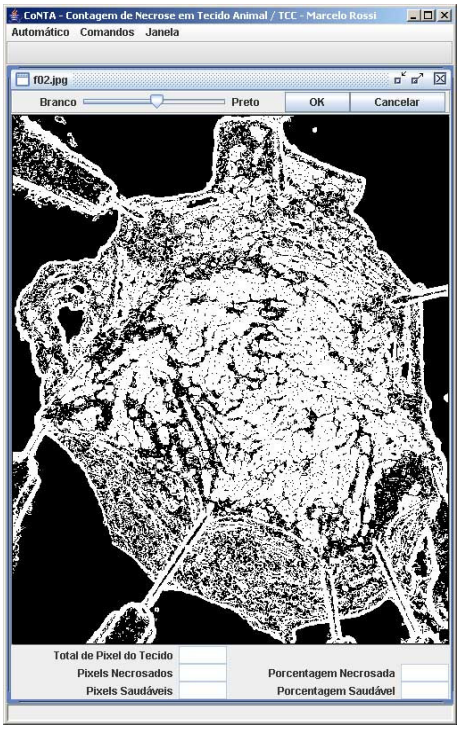


Figura 32: Binarização correta

Na Figura 31 o valor de corte é mais puxado para os tons escuros, deixando falhas na borda principal. Na Figura 32 o usuário ajustou o valor de corte para mais próximo dos tons brancos e deste modo a borda principal aparentemente está contínua.

Após o ajuste o usuário deve confirmar no painel de controle com o botão `OK`.

5.2.4.3. Dilatação da imagem.

A dilatação está exemplificada na Figura 33 e deve ser acionada pelo menu `Comandos`, opção `Dilatar`. Assim pequenas falhas nas bordas (descontinuidades) serão corrigidas.

Esta opção tem o nome de dilatação, mas o processo que ela realiza é o fechamento (como foi explicado no tópico 4.6.3 deste trabalho). No caso foi escolhido deixar no menu o nome “Dilatação” e não “Fechamento” por questão de sonoridade ou de associação com o fato de esta opção deixar as bordas maiores, ou seja, “dilatadas”.

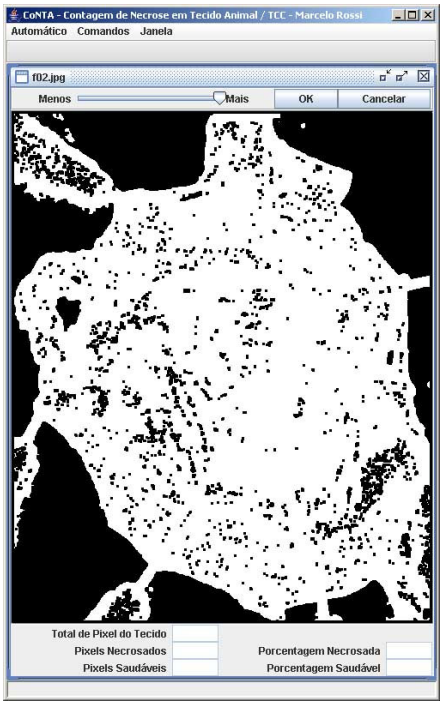


Figura 33: Dilatação das bordas

Espera-se que o usuário clique no botão `OK` no painel de controle para confirmar o fim da operação.

5.2.4.4. Identificação da borda principal da imagem.

Agora o usuário deve clicar no menu `Comandos`, opção `Clicar nas Bordas` para que ele possa clicar na borda principal e esta ser identificada.

Neste momento a porção do tecido animal se mistura com a borda (a parte branca da imagem). Isso não é ruim, pois de qualquer modo o software vai identificar a borda principal como parte do tecido. Após o clique na borda principal a imagem ficará como a Figura 34.

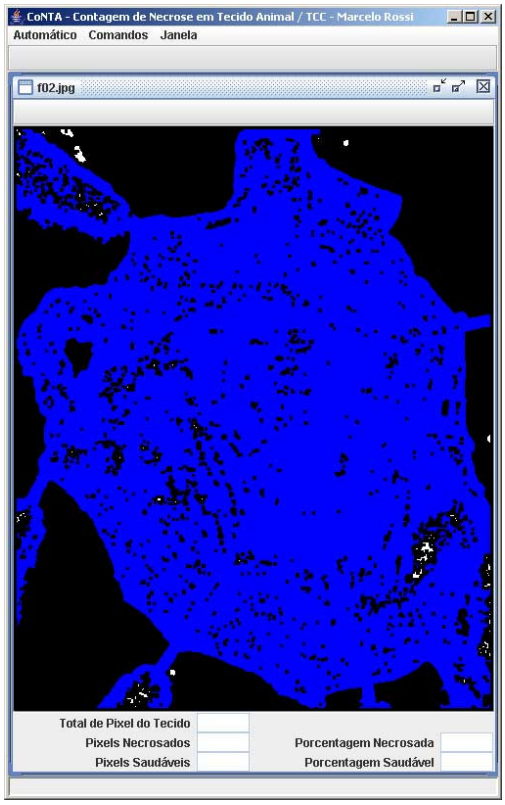


Figura 34: Identificação da borda principal

5.2.4.5. Limpeza dos pixels correspondentes ao cenário na imagem.

Agora o usuário deve clicar no cenário, ou seja na parte externa à borda principal. Porém antes deve acionar a opção `Clicar na Parte Externa`, no menu `Comandos`. O resultado pode ser visto na Figura 35 onde a cor vermelha representa o fundo da imagem (removido), a cor azul representa a borda principal identificada no procedimento anterior, a cor preta representa o tecido animal e a cor branca representa alguma borda que não está conectada à borda principal. Na verdade as cores azul, preta e branca formam o tecido animal onde a contagem da necrose será feita na próxima etapa.

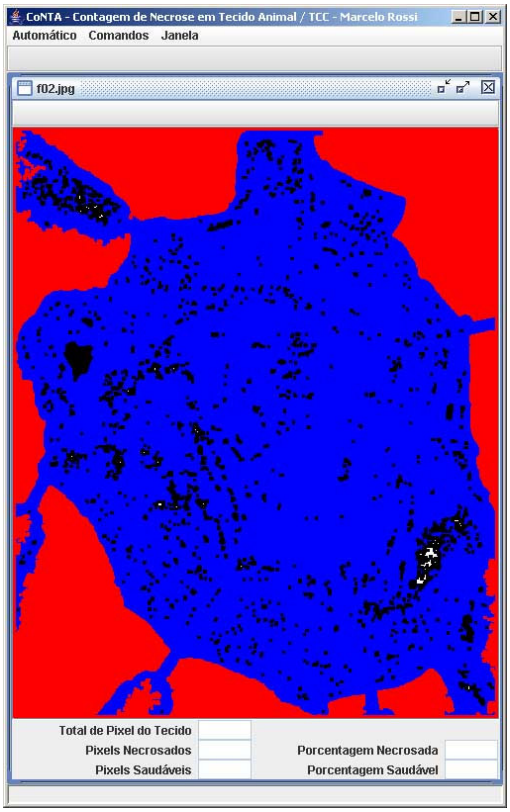


Figura 35: Limpeza da parte externa

5.2.5. Contagem da Necrose

5.2.5.1. Restauração dos pixels da imagem original correspondente ao tecido animal.

A restauração da imagem é feita ao acionar o menu `Comandos`, opção `Contabilizar` e pode ser observada na Figura 36.

Repare que foi adicionado na parte superior da aplicação o painel de controle da binarização.

Figura 36: Restauração do tecido

5.2.5.2. Binarização da imagem.

Após acionar a opção para restaurar os pixels da imagem orignal, o painel de controle da binarização volta a aparecer. Agora o usuário deverá identificar os pixels necrosados.

Isto é feito ajustando manualmente o valor de corte da binarização no painel na parte superior da aplicação. Neste ponto o usuário deve comparar a imagem que está sendo processada com a imagem original para que ele possa ajustar o melhor valor de corte. Repare o resultado do nosso exemplo na Figura 37.

Figura 37: Identificação da necrose

5.2.5.3. Contagem da Necrose no tecido animal.

Quando o usuário terminar de ajustar o valor de corte ele deve clicar no botão `OK` no painel de controle da binarização e então automaticamente a porcentagem da necrose é calculada e exibida na parte inferior da janela. No exemplo apresentado os valores de saída são os da Figura 38.

```
Total: 207053 pixels
Necrosado: 104179 pixels | 50.315136704128896 %
Saudável: 102874 pixels | 49.684863295871104 %
```

Figura 38: Resultado final

5.2.6. Considerações

Como mostrado no exemplo acima a imagem tem seu fundo removido e então o tecido animal é avaliado. Entretanto a remoção do fundo da imagem, neste exemplo, não é feita com total sucesso. Repare na Figura 39 nos objetos azuis. Estes objetos são parte do fundo da imagem (são grampos que prendem o tecido animal ao fundo) e não foram removidos.

Repare que na Figura 37 parte destes objetos aparece como necrosada e parte não, isto interfere no diagnóstico do software e não deveria acontecer. O único modo disto não acontecer é conseguir remover com sucesso estes objetos junto com o fundo.

Esta remoção é trabalhosa e complicada, por isto é recomendado que as fotos usadas não possuam objetos com cores diferentes do fundo. Se todo cenário onde a fotografia for tirada estiver com um tom similar de cor e logicamente diferente do tom do tecido animal a remoção se torna drasticamente mais eficiente. Claro que o ideal seria não ter objetos como estes que possam causar confusão no software.

Nos próximos tópicos estão descritos outros casos com diferentes objetos na imagem e seus respectivos resultados finais.

Figura 39: Objetos que não foram removidos

5.3. Outros resultados

Outros três exemplos de aplicação do software desenvolvido neste trabalho estão apresentados abaixo.

No primeiro é usado uma imagem com o fundo com vários elementos, ou seja, um fundo que não tem uma cor sólida. O segundo e o terceiro são feitos com o mesmo tecido animal, entretanto o segundo possui uma fotografia mais “aberta”, onde é possível ver outros elementos, enquanto que o terceiro exemplo é sobre uma fotografia mais “fechada”, onde aparece apenas o tecido animal e nenhum outro elemento.

A Figura 40 é a imagem original, em escala de cinza. Observe como o fundo está repleto de marcas, este não é um fundo recomendado, pois como pode ser visto na Figura 41 ele interfere totalmente na análise da imagem.

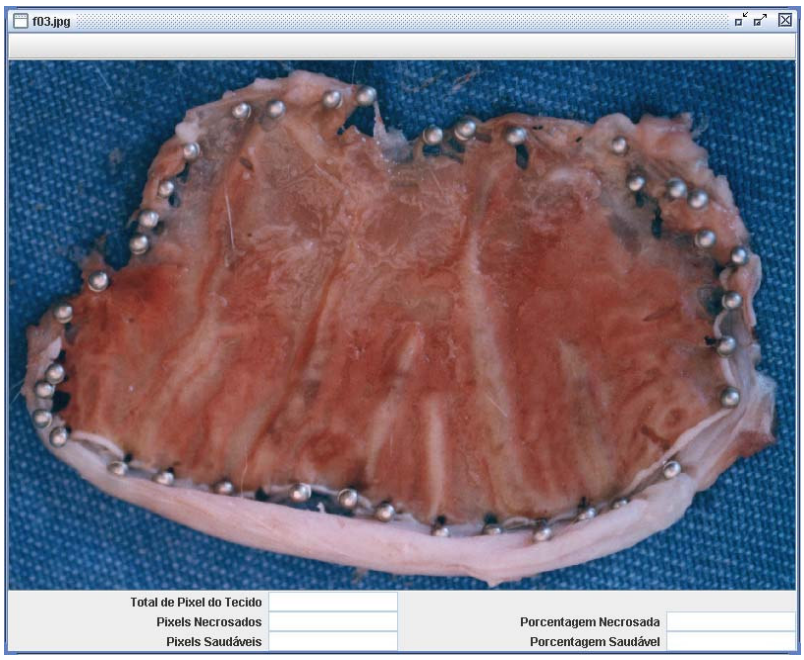


Figura 40: Exemplo de fundo ruim

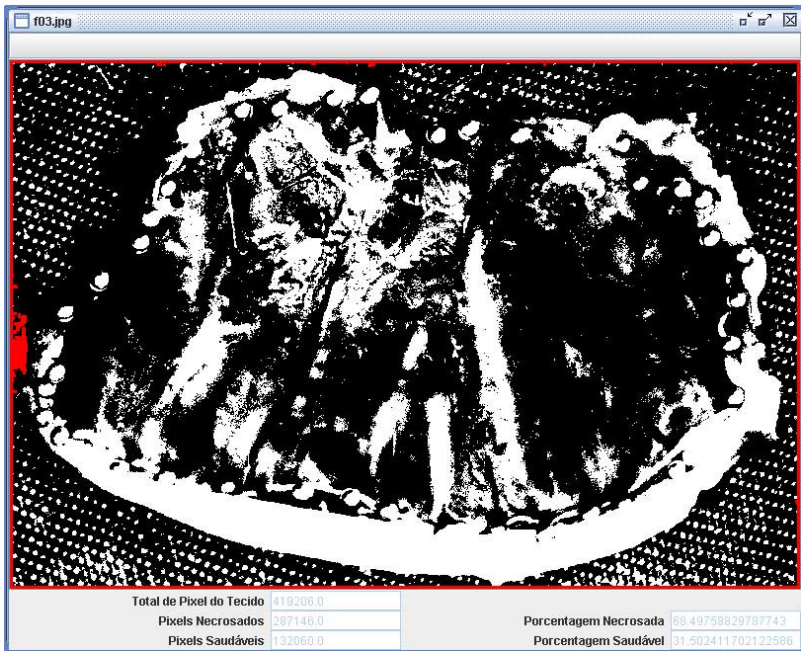


Figura 41: Resultado com fundo ruim

Apenas uma mínima porção do fundo, localizada no lado esquerdo, pode ser removida (identificada pela cor vermelha). O resultado não é nada satisfatório.

Já em outro caso com a imagem mostrada na Figura 42 o fundo é removido com perfeição, com exceção de alguns objetos usados para prender o tecido animal no fundo. O resultado pode ser visto na Figura 43.

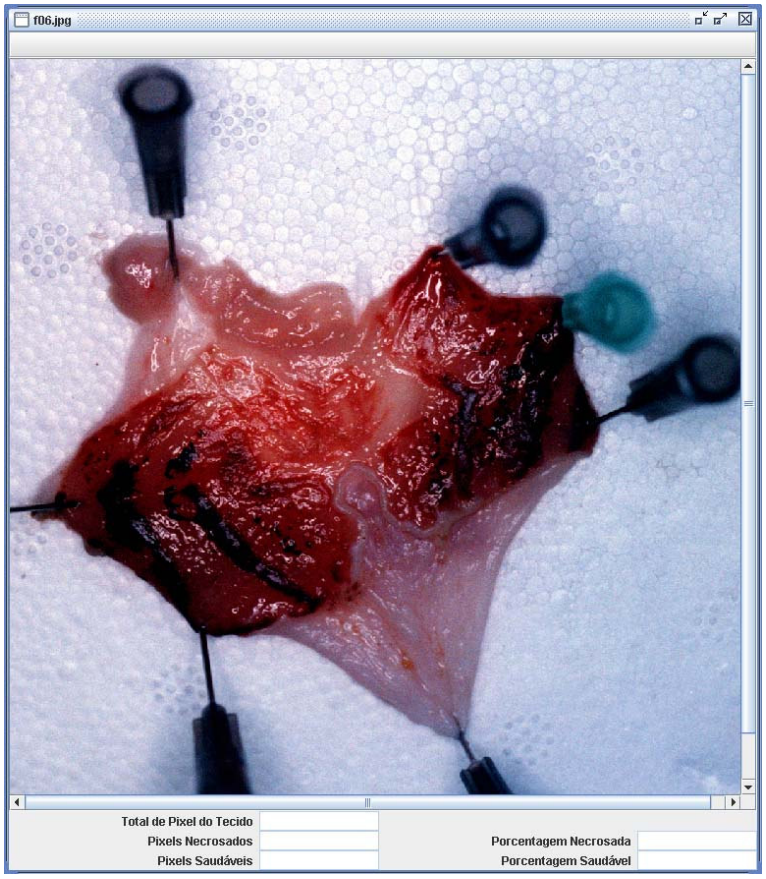


Figura 42: Exemplo com objetos ruins

Figura 43: Resultado com objetos ruins

A Figura 44 mostra o mesmo tecido, porem sem os objetos que atrapalharam na análise do exemplo anterior e a Figura 45 mostra o resultado final da análise.

Para ter uma idéia de quanto estes objetos atrapalham na avaliação a Figura 46 mostra uma comparação do resultado da avaliação da necrose das imagens mostradas nas Figuras 43 e 45.

Neste caso a variação foi de pouco mais de 1% entre as imagens, contudo pode variar bem mais.

Figura 44: Exemplo com cenário próximo do ideal

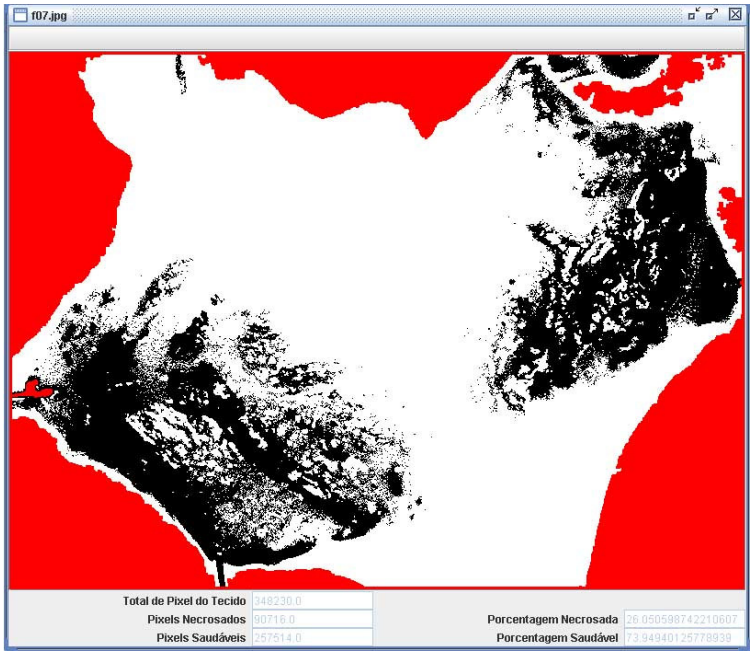


Figura 45: Resultado cenário próximo do ideal

	Figura 43	Figura 45
Porcentagem Necrosada	25,13%	26,05%
Porcentagem Sadia	74,87	73,95%

Figura 46: Comparação entre fundo ruim e cenário próximo do ideal

5.4. Automatização

A análise automática da imagem pode ser feita pelo menu `Automático`, opção `Automatizar`. Entretanto as vezes o usuário quer personalizar os parâmetros que serão usados nesta opção. A personalização pode ser feita pela opção `Configurar` do menu `Automático`, repare a Figura 47.

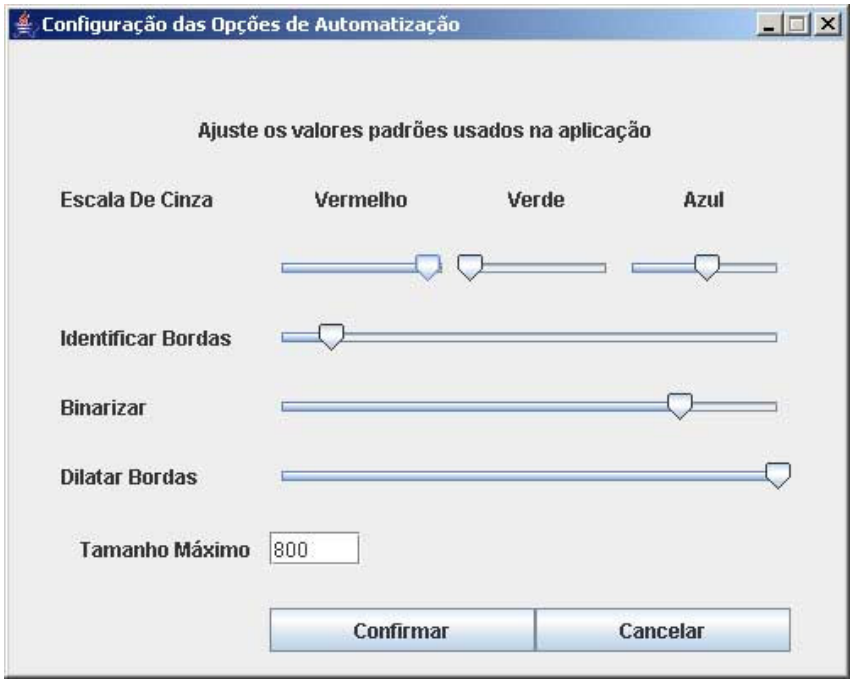


Figura 47: Configuração dos parâmetros da opção Automatizar

Os valores da conversão da imagem colorida para a escala de cinza, a precisão da detecção das bordas, o valor de corte da binarização, o valor de ajuste da dilatação e o tamanho máximo da imagem em pixels podem ser feitos nesta opção.

Imagens com padrões diferentes (textura, cor, cenário) devem ter diferentes valores ajustados nesta opção de configuração.

O resultado obtido através do menu `Automático`, opção `Automatizar` está mostrado a seguir. As Figuras 48, 49 e 50 são os resultados obtidos pela opção `Automatizar` respectivamente das Figuras 41, 43 e 45.

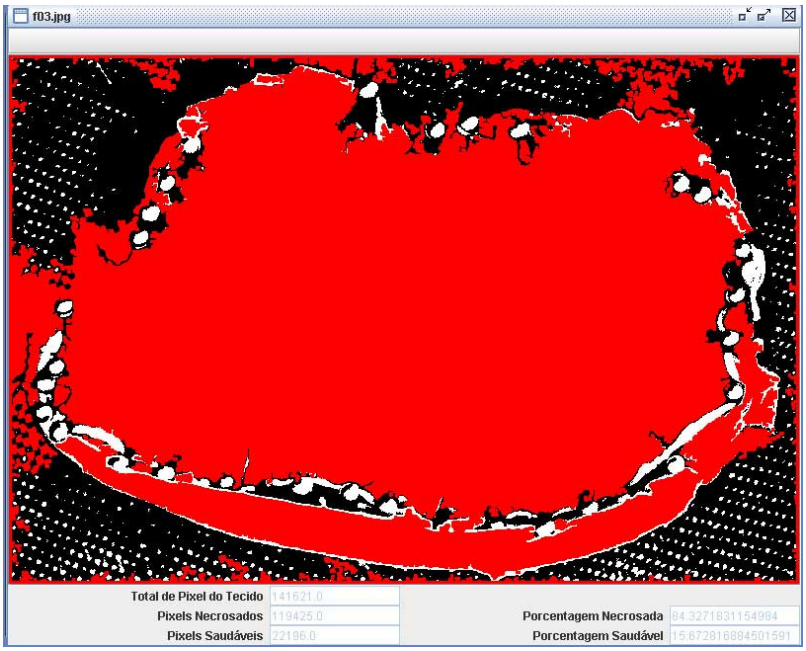


Figura 48: Exemplo automatizar com fundo ruim

Figura 49: Exemplo automatizar com objetos ruins

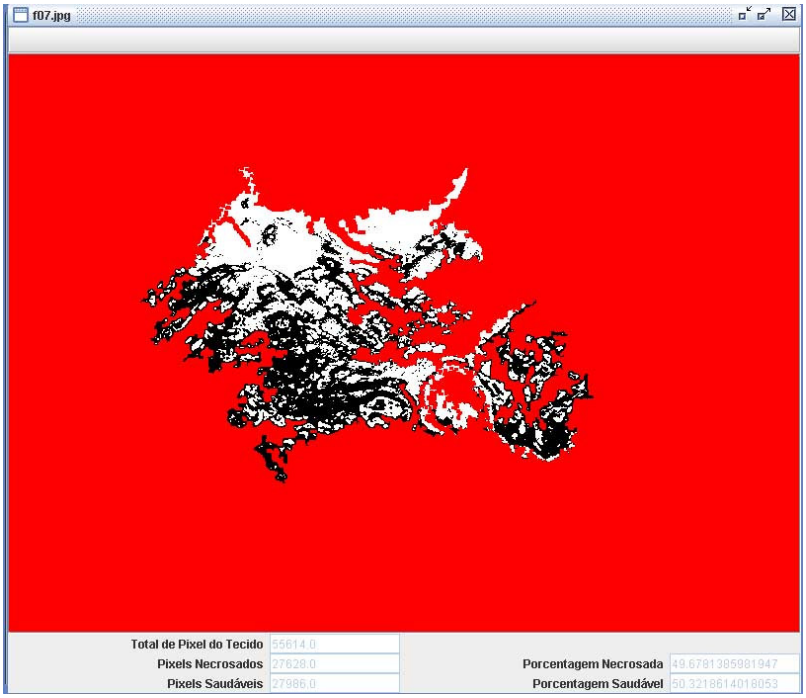


Figura 50: Exemplo automatizar com cenário próximo do ideal

5.5. Considerações finais

O resultado obtido da análise da imagem pode ser satisfatório dependendo de como o fundo da imagem está, ou seja, o cenário deve ser bem preparado antes da fotografia ser tirada, caso contrário irá atrapalhar no diagnóstico.

A opção `Automatizar` do menu `Automático` não produz bons resultados, uma vez que é difícil determinar os parâmetros que devem ser ajustados no painel acessado pelo menu `Automático`, opção `Configurar`. Cada imagem possui parâmetros diferentes e atualmente o software não possui nenhuma maneira te estipular automaticamente quais são os parâmetros que devem ser usados.

Nos exemplos dados a avaliação automática mostrada na Figura 49 pode ser considerada satisfatória, desconsiderando que objetos do cenário foram identificados, entretanto a identificação destes objetos também ocorreria no processo "manual" (passo a passo, pelo menu Comandos). Mas isto só ocorreu porque por coincidência os parâmetros estavam ajustados corretamente para os padrões desta imagem. Repare que as Figuras 43 e 49 são praticamente iguais, sendo a Figura 43 obtida pelo processo manual (pelo uso do menu `Comandos` e suas opções, passo a passo) e a Figura 49 pelo processo automático (pelo uso do menu `Automático`, opção `Automatizar`).

6. CONCLUSÃO

Para realizar este trabalho foi necessário inicialmente um estudo sobre como é possível manipular imagens e as técnicas usadas para isto. Foram vistas várias operações para a manipulação de imagens e então foi necessário estipular uma linguagem de programação capaz de implementar um software para realizar essas operações sobre imagens digitais.

A linguagem escolhida foi a Java por possuir uma vasta biblioteca, com classes e métodos que podem ser reaproveitados fazendo o desenvolvimento ser mais rápido. Para implementar as operações com imagens foi usada a API JAI, que é uma API Java com classes e métodos específicos para a manipulação de imagens.

Esta API ajudou muito na implementação de um software funcional, capaz de avaliar e diagnosticar quantos por cento do tecido animal está necrosado. A implementação visou a dar o máximo de opções ao usuário e que este soubesse exatamente o que está acontecendo com a imagem processada. Também foi implementada uma opção para automatizar o processo de análise da imagem, com o mínimo de interação com o usuário.

Este trabalho pode evoluir principalmente na questão da automatização da análise, uma vez que esta depende muito de um pré-ajuste dos parâmetros usados para avaliar a imagem e este ajuste atualmente depende do usuário. Um modo de melhorar muito este ponto seria implementar algum método de avaliar automaticamente os parâmetros que devem ser usados na imagem, assim que esta for escolhida.

Outro ponto em que este trabalho pode ser melhorado é na detecção e remoção de objetos que não pertencem ao tecido animal a ser avaliado. Estes objetos, muitas vezes pertencentes ao cenário onde a fotografia foi tirada e atrapalham na análise.

REFERÊNCIAS

GONZALEZ, Rafael C.; WOODS, Richard E. **Processamento de imagens digitais**. São Paulo: Edgard Blücher, 2000.

DEITEL, H. M.; DEITEL, P. J. **JAVA: Como programar.** Porto Alegre: Bookman, 2003.

SUN MICROSYSTEM. **What is Java Advanced Imaging**. Disponível em: <http://java.sun.com/products/java-media/jai/whatis.html#develop>. Acessado em 5 de Novembro de 2005.

SUN MICROSYSTEM. **Overview (Java 2 Plataform SE 5.0)**. Disponível em: <http://java.sun.com/j2se/1.5.0/docs/api/>. Acessado em 3 de Novembro de 2005.

SUN MICROSYSTEM. **Java Advanced Imaging**. Disponível em: <http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/index.html>. Acessado em 5 de Novembro de 2005.

SUN MICROSYSTEM. **Image Acquisition and Display**. Disponível em: <http://java.sun.com/products/java-media/jai/forDevelopers/jai1_0_1guide-unc/Acquisition.doc.html>. Acessado em 1 de Novembro de 2005.

SUN MICROSYSTEM. **Java Advanced Imaging API Home Page**. Disponível em: <http://java.sun.com/products/java-media/jai/iio.html >. Acessado em 8 de Novembro de 2005.