FUNDAÇÃO DE ENSINO "EURÍPIDES SOARES DA ROCHA"
CENTRO UNIVERSITÁRIO "EURÍPIDES DE MARÍLIA" – UNIVEM
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

THIAGO CORRÊA DA FONSECA

PROCESSAMENTO DE IMAGEM PARA DETECÇÃO DE NÓDULOS EM MAMOGRAMAS DIGITALIZADOS

MARÍLIA
2007

THIAGO CORRÊA DA FONSECA

PROCESSAMENTO DE IMAGEM PARA DETECÇÃO DE NÓDULOS EM MAMOGRAMAS DIGITALIZADOS

Monografia apresentada ao Curso de Bacharelado em Ciência da Computação, do Centro Universitário Eurípides de Marília, mantido pela Fundação de Ensino Eurípides Soares da Rocha, como requisito para obtenção do grau de Bacharel em Ciência da Computação.

Orientadora:
Profª. Drª. Fátima L.S. Nunes Marques

MARÍLIA
2007

FONSECA,Thiago Correa

Processamento de Imagens para Detecção de Nódulos em Mamogramas Digitalizados / Thiago Corrêa da Fonseca; orientador: Profª. Drª. Fátima L.S. Nunes Marques. Marília, SP: [s.n], 2007.

91 f.

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

CDD:006

THIAGO CORRÊA DA FONSECA

"PROCESSAMENTO DE IMAGENS PARA DETECÇÃO DE NÓDULOS EM MAMOGRAMAS DIGITALIZADOS"

Banca Examinadora da monografia apresentada ao Curso de Bacharelado em Ciência da Computação do UNIVEM, para obtenção do Título de Bacharel em Ciência da Computação.

Nota:_____

Orientador: Fátima L.S. Nunes Marques _____

1° Examinador: Márcio Eduardo Delamaro _____

2° Examinador: Kalinka Regina L.J.C. Branco _____

Marília, ____de_____de 2007

A Deus e meus familiares

AGRADECIMENTOS

Aos meus pais por todo incentivo que foi dado a mim durante esse período e pelo imensurável amor e compreensão contribuindo para a minha formação pessoal e profissional.

À minha orientadora Fátima, por sempre me estimular, não me deixando desanimar, e pela sua ajuda no desenvolvimento deste trabalho.

A todos os professores e colegas de turma com quem eu convivi neste período.

Aos meus amigos da faculdade Ricardo, Wellington, Rogério e do laboratório LAPIS, pela ajuda e incentivo.

FONSECA, Thiago Corrêa da Fonseca. Processamento de Imagens para Detecção de Nódulos em Mamogramas Digitalizados. 2007. 91 f. Monografia (Bacharelado em Ciência da Computação) – Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2007.

RESUMO

O presente trabalho traz um estudo de conceitos relacionados a sistemas de auxílio ao diagnóstico (CAD – *Computer Aided Diagnosis*) para detecção de nódulos em mamografias e a implementação de um sistema que faz a análise da imagem e demarca a região da mama que contém um nódulo. O foco de tais sistemas é auxiliar o usuário na avaliação das imagens oferecendo resultados que diminuam os equívocos ocasionados por uma única avaliação. Este trabalho foi desenvolvido utilizando a linguagem de programação Java juntamente com a API JAI (*Java Advanced Imaging*).

Palavras-Chave: Mamografia, CAD, Processamento de Imagens, Imagens Médicas.

FONSECA, Thiago Corrêa da Fonseca. Processamento de Imagens para Detecção de Nódulos em Mamogramas Digitalizados. 2007. 91 f. Monografia (Bacharelado em Ciência da Computação) – Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2007.

ABSTRACT

This work presents a study of related concepts of Computer Aided Diagnostic (CAD) for detection of nodules in mammographic images and the implementation of a system for that analyzing the image and demarcates the region of the breast that contains a masse. The focus of such systems is helping the user in the evaluation of the images offering that they diminish the mistakes caused for an evaluation. This work was developed by using the programming language Java together with API JAI (Java Advanced Imaging).

Keywords: Mammography, CAD, Image Processing, Medical Images

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
BMP	Windows Bitmap
CAD	Computer Aided Diagnosis
GIF	Graphics InterChange Format
INCA	Instituto Nacional de Combate ao Câncer
JAI	Java Advanced Imaging
PNG	Portable Network Graphics
ROC	Receiver Operation Characteristic
ROI	Region of Interest
TIFF	Tagged Image File Format

SUMÁRIO

INTRODUÇÃO

Apesar das diversas campanhas para um maior esclarecimento e prevenção, como o Programa Nacional de Controle do Câncer do Colo do Útero e de Mama do Instituto Nacional de Combate ao Câncer, segundo o INCA (2006), o câncer de mama, que é o câncer mais letal entre as mulheres, teve uma estimativa de 48.930 casos no ano passado, e a cada ano o número de casos tem-se tornado maior.

Uma grande porcentagem dos casos ocorre em mulheres com mais de 35 anos de idade e, na maioria dos casos, o nódulo, que é o crescimento desordenado de células no organismo, Sampat *et al.,* (2005a) é constatado quando a própria mulher toca a mama. Nestes casos geralmente o nódulo já está em um estágio avançado e o tratamento tem a sua dificuldade elevada.

Segundo Sampat *et al.,* (2005a), os programas de *screening,* que são exames periódicos realizados em mulheres que se enquadram em grupos de risco, constituem o meio mais eficaz de detecção do câncer no seu estado inicial de desenvolvimento, pois a detecção precoce de um nódulo possuindo um tamanho pequeno (1cm até 3cm) representa uma oportunidade de cura muito elevada.

De acordo com Santos (2002), o processamento de imagens médicas tem se tornado um meio de importância relevante para o auxílio ao diagnóstico devido ao tempo utilizado para a manipulação das imagens, a quantidade de atributos que podem ser extraídos e as técnicas que são desenvolvidas para o uso.

Na pesquisa de métodos que auxiliem a detecção precoce, os sistemas de auxílio ao diagnóstico, ou CAD ("*Computer-Aided Diagnosis*") têm sido amplamente desenvolvidos. Giger(2000) define o diagnóstico auxiliado por computador como aquele no qual o radiologista usa os resultados de uma análise computadorizada de

imagens médicas como uma “segunda opinião” na detecção de lesões e na elaboração do diagnóstico.

Objetivo

Este trabalho tem como objetivo a implementação de um sistema de processamento de imagens fazendo uso linguagem de programação Java e da API JAI (*Java Advanced Imaging)* para a detecção de massas ou nódulos em imagens digitalizadas de mamografias

Organização dos Capítulos

Esta monografia está organizada da seguinte forma:

CAPÍTULO 1 - Princípios básicos de processamento de imagens, desde a definição matemática de uma imagem digitalizada até técnicas de manipulação de imagens como realce, suavização e reconhecimento de bordas, entre outras.

CAPÍTULO 2 - Principais aspectos sobre as pesquisas e as técnicas utilizadas no desenvolvimento de sistemas de auxílio à detecção de nódulos em mamografias digitalizadas.

CAPÍTULO 3 - Tecnologias utilizadas no desenvolvimento do sistema e suas principais características.

CAPÍTULO 4 - Algoritmos executados para a segmentação das imagens.

CAPÍTULO 5 - Relata os testes executados para a detecção dos nódulos nas imagens no sistema assim como a discussão dos resultados apresentados pelo sistema durante a execução dos testes.

CAPÍTULO 6 - Conclusões e também os trabalhos futuros.

CAPÍTULO 1. PROCESSAMENTO DE IMAGENS DIGITAIS

1.1. Formação da Imagem

De acordo com Ballard e Brown (1982), a formação de uma imagem ocorre quando um sensor detecta que uma radiação emitida por um aparelho interagiu com um objeto físico. Portanto, pode-se concluir que uma imagem é a representação de um objeto físico, possibilitando o armazenamento, a manipulação e a interpretação para o alcance de determinado resultado.

Matematicamente, Gonzales e Woods (2002), definem que uma imagem pode ser descrita como uma função luminosa bidimensional *f(x,y)*, onde *x,y* indicam as coordenadas espaciais e o valor de *f* em qualquer combinação de *x* e *y* corresponde ao brilho ou nível de cinza da imagem naquele ponto.

Sendo a luz uma forma de energia, a função *f*(x,y) deve ser positiva, conforme mostrado na Equação 1

$$0 < f(x,y) < 0 \tag{1}$$

Uma imagem basicamente pode ser caracterizada a partir de dois componentes:

- Quantidade de luz que incide nos objetos que compõem a cena e
- quantidade de luz que é refletida pelos objetos presentes na cena

Estes dois componentes podem ser definidos como iluminação e reflectância, e são representados por *i(x,y)* e *r(x,y)*, conforme as equações 3 e 4. O produto destas duas funções resulta em *f(x,y)* que é demonstrado na equação 2.

$$f(x,y) = i(x,y), r(x,y) \tag{2}$$

onde $$0 < r(x, y) < 1 \quad (3)$$

e $$0 < i(x, y) < \infty \quad (4)$$

1.2. Imagem Digital

Um caso especial de imagem é constituído pelas imagens digitais, onde a representação consiste em um vetor de valores discretos. Geralmente este vetor é unidimensional e o domínio e imagem de *f(x,y)* são também discretos. O domínio é finito (geralmente uma matriz retangular) e o conjunto imagem é formado por valores no intervalo [0,M]. Para aplicações práticas, a imagem é uma função contínua que é representada por medidas em intervalos regularmente espaçados. Os valores assumidos em cada ponto medido são quantificados em um número pertencente a uma escala de diferentes níveis de cinza. Assim, é estabelecido que o valor 0 (zero) é atribuído à cor mais escura (preto) e o valor máximo M relaciona-se à cor mais clara da escala (branco).

Podemos então representar uma imagem como uma matriz onde cada ponto é um valor discreto, como mostrado na equação 5.

$$f(x,y) \cong \begin{bmatrix} f(0,0) & f(0,1) & \dots & f(0,n-1) \\ f(1,0) & f(1,1) & \dots & f(1,n-1) \\ . & & & \\ . & & & \\ . & & & \\ f(m-1,0) & f(m-1,1) & \dots & f(m-1,n-1) \end{bmatrix} \quad (5)$$

Ballard e Brown (1982) e Gonzales e Woods (2002) enfatizam que o objetivo de definir matematicamente uma imagem é a possibilidade de manipular o seu conteúdo a fim de transformá-la ou retirar dela informações importantes. Ao vasto conjunto de operações que podem ser aplicados em uma matriz que representa uma imagem denominamos processamento de imagem.

Segundo Gonzales e Woods (2002), cada ponto ou elemento constituinte da matriz-imagem é chamado de "pixel" que é uma abreviação do termo em inglês *picture element*. A medida de um pixel depende da resolução com a qual a imagem foi adquirida.

Como o pixel é, então, a menor unidade sobre a qual pode-se realizar operações, para isso são definidas algumas relações fundamentais, as quais são apresentadas nas seções que seguem.

1.2.1. Relações Fundamentais entre os pixels

Enquanto estiverem sendo descritas as relações entre os pixels, será usada para representar $f(x,y)$ uma imagem digital, a representação de um único pixel será feita por letra minúscula, como *m* e *n.* Um conjunto de pixels será denotado por uma letra maiúscula, como T.

1.2.1.1. Vizinhança

Gonzales e Woods (2002) definem vizinhança como um conjunto de pixels que está a uma unidade de distância do pixel *m*. Os pixels "vizinhos" podem ser classificados em *horizontais* e *verticais*, *vizinhança-de-4* ou $N_4(P)$. Na Figura 1, os pixels que pertencem à vizinhança estão representados pelos quadrados cinza. Em relação ao pixel central, a posição dos pixels vizinhos é definida como:

$$(x+1, y), (x-1, y), (x, y+1), (x, y-1)$$

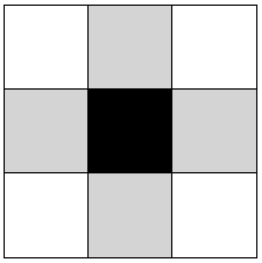


Figura 1 - Exemplo de Vizinhança N4(P)

Os pixels vizinhos diagonais, ou $N_D(P)$, têm coordenadas definidas como:

$$(x+1, y+1), (x+1, y-1), (x-1, y+1), (x-1, y-1)$$

E podem ser ilustrados da maneira apresentada na Figura 2:

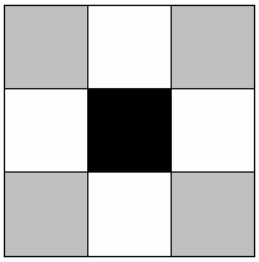


Figura 2 - Exemplo de Vizinhança ND(P)

A união destes dois conjuntos de elementos forma a "vizinhança-de-8", ou $N_8(P)$, que é representada na Figura 3:

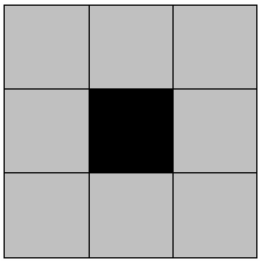


Figura 3 - Exemplo de Vizinhança N8(P)

1.2.1.2. Conectividade

É um importante conceito usado para estabelecer bordas de objetos e componentes de regiões em uma imagem. Gonzales e Woods (2002) definem que dois pixels são conectados se:

a) são adjacentes e;

b) obedecem a um critério de similaridade, isto é, seus valores estão dentro de um conjunto pré-estabelecido de valores.

Seja V={G1, G2, ..., Gk} o conjunto de "k" valores de níveis de cinza usado para definir a conectividade. São definidos três tipos de conectividade:

- Conectividade-4: dois pixels p e q com valores em V e $q \supset N_4(p)$;
- Conectividade-8: dois pixels p e q com valores em V e $q \supset N_8(p)$;
- Conectividade-m: dois pixels p e q com valores em V e:

i) $q \supset N_4(p)$ ou

ii) $q \supset N_D(p)$ e $N_4(p) \cap N_4(q) = \varnothing$.

1.2.1.3. Distância entre pixels

A distância entre pixels é uma importante definição para grande parte dos algoritmos que manipulam a imagem. Conforme citado em Gonzalez e Woods (2002) , a distância d(x,y) geralmente é um valor mensurável e:

- d(x,y) = 0, se e somente se x = y;
- d(x,y) = d(y,x);
- $d(x,y) + d(y,z) \geq d(x,z)$.

Há diversas fórmulas empregadas para a definição de distância e atualmente ainda são definidas e adaptadas para aplicações específicas. Algumas das métricas mais conhecidas, aplicadas para dois pixels $p=(x_1,y_1)$ e $q=(x_2,y_2)$ são mostradas nas equações 6,7 e 8

Distância Euclidiana: $$d(p,q) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \quad (6)$$

Distância "City Block": $$d(p,q) = \max\{|x_1 - x_2|, |y_1 - y_2|\} \quad (7)$$

Distância “Chessboard”: $$d(p,q) = |x_1 - x_2| + |y_1 - y_2| \quad (8)$$

1.2.1.4. Histograma

Um conceito que é muito utilizado no processamento digital de imagens é o de histograma. Segundo Gonzales e Woods (2002), com o histograma é possível analisar estatisticamente uma imagem digital. O histograma é uma ferramenta básica de processamento digital de imagens, utilizada como ponto de partida para algumas técnicas de realce e análise, capaz de fornecer a distribuição de freqüência dos níveis de cinza de uma imagem. O modelo matemático do histograma é dado pela razão entre o número de pixels com “radiometria” k, (valor que determina o brilho de uma superfície na imagem) e o número total de pixels, onde k pertence ao intervalo dado pelo número máximo de níveis de cinza da imagem.

Segundo Gonzales e Woods (2002), a forma com que a curva do histograma é dada caracteriza a distribuição estatística das tonalidades na imagem. Uma simples manipulação na forma da curva do histograma pode alterar diretamente parâmetros como brilho e contraste. Os histogramas tendem, em geral, a apresentar uma forma que se aproxime de uma distribuição gaussiana, com maior alongamento da curva em direção aos níveis de cinza de radiância, que é definido como a potência emitida por unidade de área, ou seja, um pixel com intensidade do nível de cinza mais elevado. Para imagens de baixo contraste, o histograma apresenta uma forma mais estreita (com menor variância dos níveis de cinza), enquanto que as imagens de alto contraste apresentam histogramas mais abrangentes (conseqüentemente com maior variância dos níveis de cinza).

Dessa forma, o valor médio de tonalidade, encontrado no histograma, indica o valor do brilho médio da imagem digital correspondente, enquanto a variância do histograma indica o grau de contraste.

Embora o histograma forneça informações de brilho e contraste, nenhuma informação espacial é fornecida diretamente pelo histograma.

Pode-se representar um histograma de uma imagem de acordo com a Figura 4.

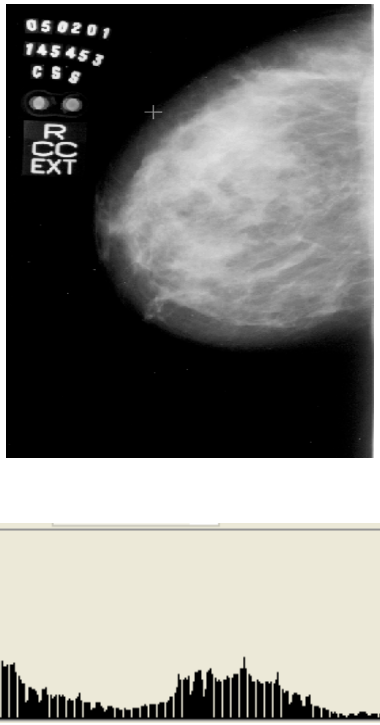


Figura 4 - Imagem em nível de cinza e seu histograma

Introduzidos os conceitos anteriores, pod-se aplicá-los ao processamento de imagens. As técnicas de processamento de imagem podem ser divididas em três categorias:

- Processamento de baixo nível: responsável pelo tratamento de funções que podem ser desde a aquisição da imagem, como a redução de ruídos e o realce de características importantes da imagem, como o contraste.

- Processamento de nível médio: trata da extração e caracterização de partes de uma imagem, como por exemplo, regiões e utiliza os processos de segmentação, representação e descrição.

- Processamento de nível alto: trabalha com a interação da imagem com um banco de conhecimento, executando tarefas como o reconhecimento e a interpretação das informações extraídas da imagem.

Gonzalez e Woods (2002) ilustram os processos e suas relações de acordo com a Figura 5.

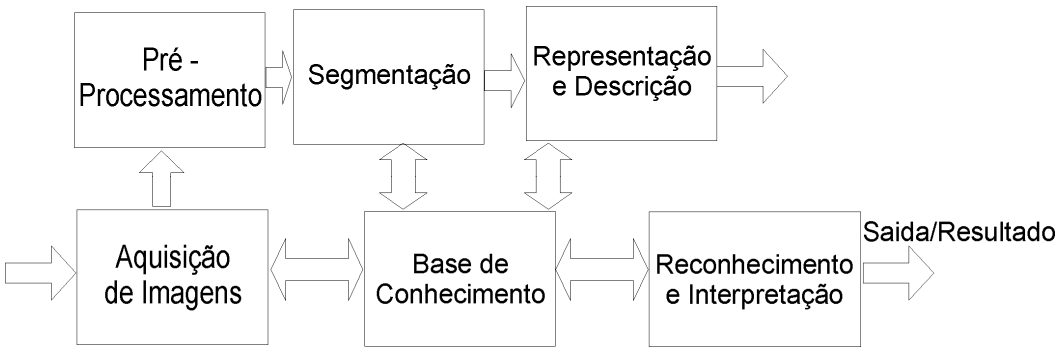


Figura 5 - Esquema das etapas do processamento de imagem

1.3. Processamento de Baixo Nível

De acordo com Gonzalez e Woods (2002), o processamento de baixo nível trata da melhoria das características das imagens através do realce e da filtragem de dados decorrentes da aquisição, a fim de que os resultados das técnicas aplicadas sejam mais apropriados para a obtenção de resultados do que a imagem original.

Gonzales e Woods (2002) listam as principais técnicas de baixo nível, que são: filtragem, operadores de borda, transformações de escala, orientação de superfície e fluxo ótico.

Russ (1995) ressalta que as técnicas que fazem parte do processamento de baixo nível podem ser aplicadas na imagem logo após a sua digitalização e o seu armazenamento, pois uma digitalização de altíssima qualidade é impraticável devido aos custos inerentes.

As técnicas e os algoritmos trabalham sobre dois domínios da imagem, que são o domínio espacial e o domínio de freqüência da imagem.

Gonzales e Woods (2002) definem o domínio espacial como o agregado ou o conjunto de pixels que compõem uma imagem e, conseqüentemente, as técnicas pertencentes a este domínio, atuam diretamente sobre os pixels. Matematicamente, essas técnicas podem ser definidas de acordo com a equação 9.

$$g(x,y) = T[f(x,y)] \tag{9}$$

As técnicas podem ser aplicadas pixel a pixel, transformando *T* em uma função de mapeamento dos níveis de cinza de acordo com a equação 10, onde *s* e *r* são os valores de *f(x,y)* e *g(x,y)*, ou também podem ser baseadas na vizinhança do pixel, que é uma subimagem cujo centro é o pixel em questão.

$$s = T(r) \tag{10}$$

O domínio da freqüência é representado pelas operações baseadas na Transformada de Fourier. De acordo com Dias (2004), no domínio da freqüência a principal metodologia a ser usada é a Transformada Discreta de Fourier (TDF), pois é possível trabalhar em um domínio que proporcionalmente tem menos valores, mas não se perde a quantidade de informação relevante.

A Transformada de Fourier para imagens bidimensional pode ser apresentada conforme a equação 11.

$$\tau\{f(x)\} = F(u) = \int_{-\infty}^{\infty} f(x)\exp[-j2\pi ux]dx \tag{11}$$

A equação descrita detecta as freqüências no sinal *f*. Logo $\exp[-j\pi ux]$ é um sinal periódico de freqüência em *x*. A integral pode ser definida como um medida de densidade da freqüência *x* no sinal *f*.

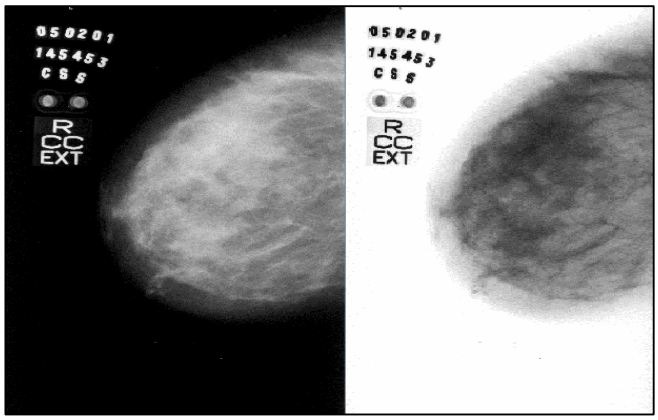
1.3.1. Filtro Negativo

Gonzales e Woods (2002) definem o filtro negativo como uma transformação que reverte a ordem do preto para o branco, de maneira que a intensidade da imagem de saída diminua à medida que a intensidade de entrada aumente.

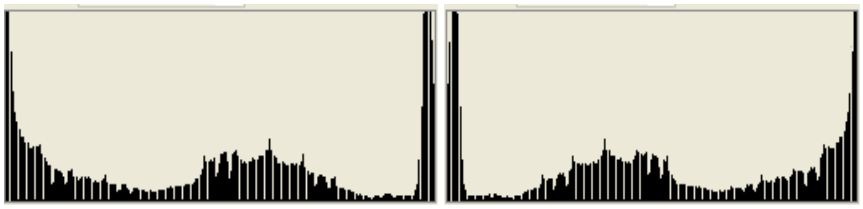
Neto *et al*. (2004) representam matematicamente esta técnica conforme a equação 12, onde f(x,y) representa a imagem original, g(x,y) representa a imagem resultante e W representa o maior valor da escala de níveis de cinza da imagens.:

$$T[f(x,y)] = g(x,y) = W - f(x-y) \tag{12}$$

Um exemplo de aplicação desta técnica pode ser vista na Figura 6.



(a)



(b)

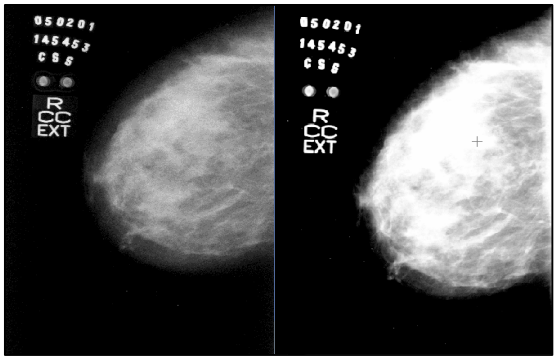
Figura 6 – Aplicação do filtro negativo: (a) imagem original e processada e (b) respectivos histogramas

1.3.2. Alargamento de Contraste (*Stretch*)

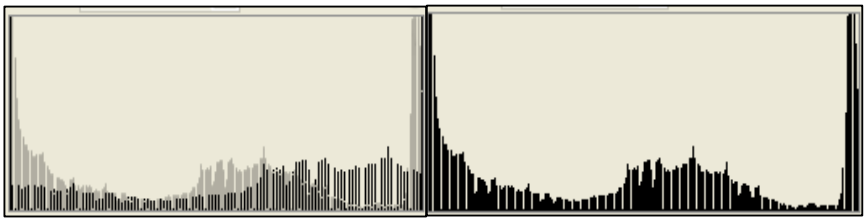
Esta técnica promove o aumento da escala dinâmica dos níveis de cinza em uma imagem com baixo contraste, que pode ser resultado de diversos fatores, como iluminação insuficiente ou mesmo um procedimento errôneo na aquisição da imagem.

A função utiliza dois pontos p(x1,y1) e q(x2,y2). Se x1 = y1 e x2 = y2, a transformação é linear, não ocorrendo mudanças nos níveis de cinza. Se x1 = y1 e x2 = 0 e y2 = N – 1, ela se torna uma função de limiarização criando uma imagem binária.

O uso desta transformação preserva a ordem dos níveis de cinza. Um exemplo de aplicação desta técnica pode ser visto na Figura 7.



(a)



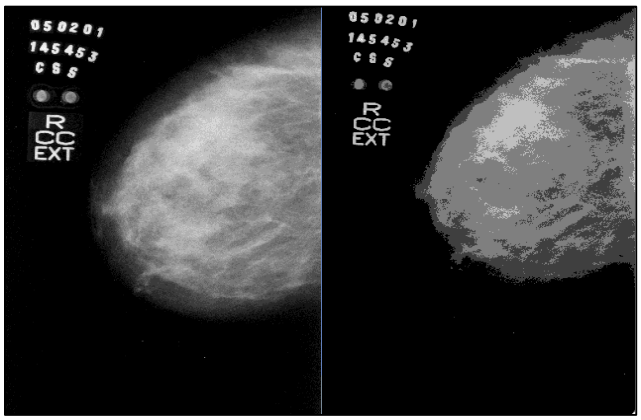
(b)

Figura 7 – Aplicação do filtro *stretch:* (a) Imagem Original Processada e (b) Respectivos histogramas

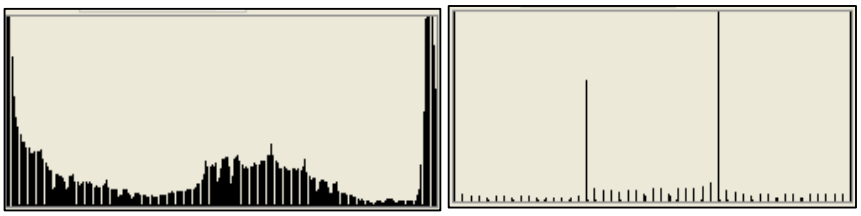
1.3.3. Quantização

De acordo com Nunes (2006), a técnica da quantização, também conhecida como o agrupamento dos níveis de cinza pode ser definida como uma função que atua sobre o histograma da imagem e realiza uma redução dos níveis de cinza na imagem.

Um exemplo da aplicação desta técnica pode ser visto na Figura 8.



(a)



(b)

Figura 8 – Aplicação do filtro quantização (a) Imagem original processada e (b) Respectivos histogramas

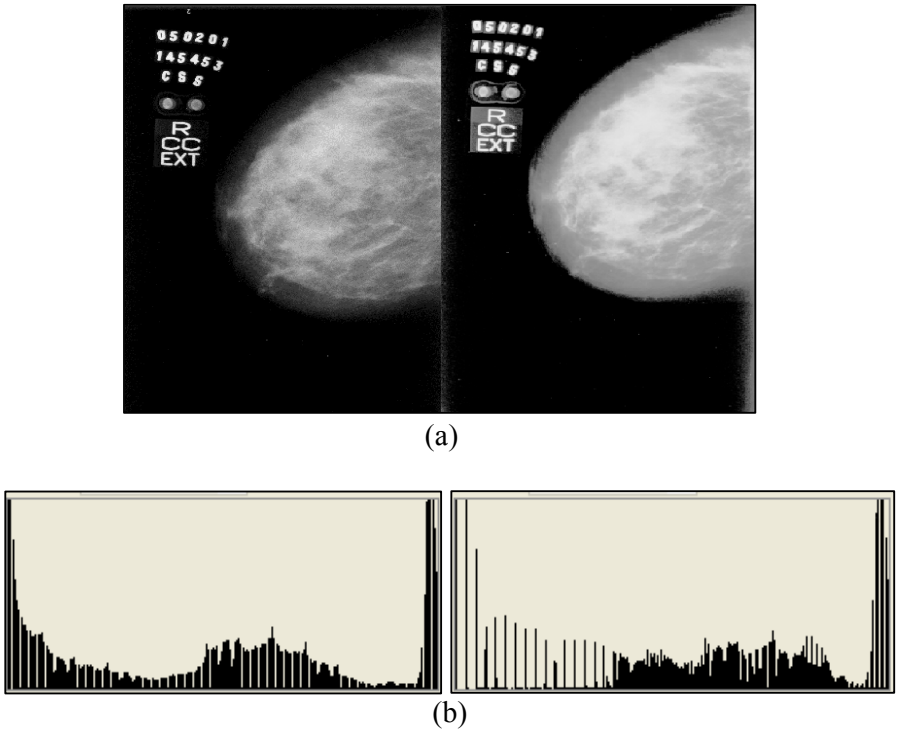
1.3.4. Equalização do Histograma

Nunes (2006) define a equalização como o mapeamento que tem por finalidade produzir uma imagem cujo histograma seja uniforme, ou seja, todos os níveis de cinza devem aparecer na imagem com freqüência semelhante.

De acordo com Russ(1995), um dos processos para a equalização do histograma pode ser definido da maneira descrita a seguir. Para cada nível de intensidade de cinza na imagem original e em seu respectivo histograma, que será representado por J, o novo valor calculado será calculado pela equação 13.

$$k = \sum_{j=0}^{j} \left(\frac{N_i}{T} \right) \tag{13}$$

onde o somatório representa o número total de pixels na imagem com nível de intensidade de cinza igual ou menor do que J e T representa o número total de pixels na imagem. A demonstração da técnica está representada na Figura 9.



(a)

(b)

Figura 9 – Aplicação da equalização de Histograma: (a) Imagem original processada e (b) Respectivos histogramas

1.3.5. Técnicas baseadas na vizinhança (máscaras)

De acordo com Gonzales e Woods (2002) técnicas baseadas na vizinhança usam um *template*, ou seja, uma máscara que tem como objetivo modificar o valor do nível de cinza de um pixel em função do seu próprio valor e dos valores de seus vizinhos. Em geral, o pixel cujo valor será modificado está no centro da máscara.

A idéia básica que se pode conceber ao manipular uma máscara é a soma do produto entre coeficientes dos níveis de cinza dos pixels sobre a máscara e os coeficientes da máscara.

Em uma máscara 3x3, por exemplo, que é demonstrada na Figura 10, os níveis de cinza serão representados por w1,w2,...,w9, a resultante é apresentada na equação 14

$$R = w1x1 + w2x2 + ... + w9x9 \tag{14}$$

e o valor resultante é atribuído ao pixel localizado no centro da máscara.

w1	w2	w3
w4	w5	w6
w7	w8	w9

Figura 10 – Exemplo de uma máscara 3x3 genérica

1.3.5.1. Filtro de Média

De acordo com Gonzales e Woods (2002) , no domínio do espaço, a filtragem pela média da vizinhança é um exemplo de filtro passa-baixa que tem por objetivo a redução de ruído atribuindo ao pixel central a média dos valores de todos os pixels da vizinhança, conforme a equação 15, onde *g(x,y)* é a imagem gerada, *f(p,q)* é a imagem original, V é o total de elementos da vizinhança e S é o conjunto de coordenadas da vizinhança. No entanto, se obtém uma imagem resultante "borrada", com perda de definição nas bordas. Quanto maior o tamanho da máscara maior será o borramento da imagem.

Um exemplo da aplicação desta técnica é apresentado na Figura 11.

$$g(x,y) = \frac{1}{V} \sum_{(p,q \in S)} f(p,q) \tag{15}$$

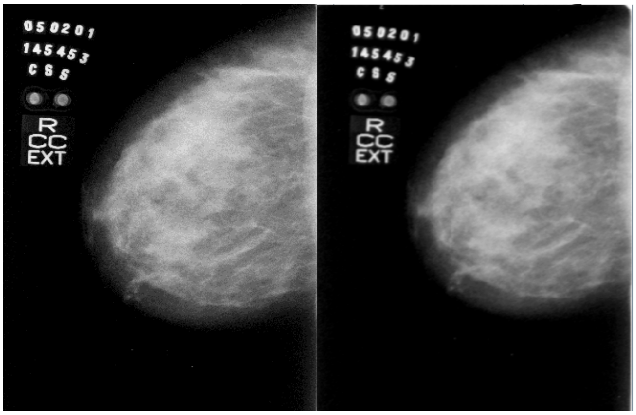


Figura 11 – Aplicação do filtro de média

1.3.5.2. Filtro Mediana

O filtro por mediana é uma alternativa na tentativa de se obter a suavização sem borrar em demasia a imagem, onde o pixel central receberá o valor correspondente à mediana dos valores da vizinhança. Em uma máscara de tamanho 3x3, a mediana é o 5º valor obtido quando se ordena em ordem crescente os 9 pixels da vizinhança. Com essa técnica, como ilustra a Figura 12, o objetivo principal é induzir pontos com intensidades desiguais a se tornarem iguais aos seus vizinhos, suavizando pontos que sejam diferentes na imagem.

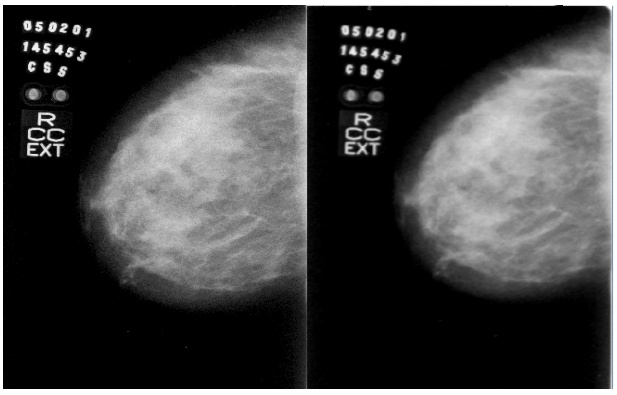


Figura 12 – Aplicação do filtro de mediana

1.3.6. Filtro Passa-Alta

Segundo Nunes (2006), os filtros passa-alta no domínio espacial têm por objetivo destacar os detalhes considerados finos ou realçar detalhes.

Gonzales e Woods (2002) definem como um modelo de filtro passa-alta no domínio espacial, uma máscara cujo valor do pixel central seja positivo e os demais valores sejam negativos, conforme a Figura 13.

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

Figura 13 – Máscara para um filtro pasa-alta

1.4. Processamento de Nível Médio

Essa fase de processamento tem a responsabilidade de identificar as formas significativas de uma imagem a fim de fornecer informações para a sua interpretação.

Este processo recebe o nome de segmentação, definido como "o processo que subdivide uma imagem em suas partes ou objetos constituintes". Para Ballard e Brown (1982), uma imagem segmentada, em visão computacional, é resultante do agrupamento de partes de uma imagem generalizada em unidades homogêneas considerando um ou mais aspectos.

O sucesso do algoritmo da segmentação determina, freqüentemente, o sucesso ou a falha do algoritmo total da análise de imagem.

Segundo Peccini e D´ornellas (2005), o processo de segmentação de imagens pode ser classificado na seguinte forma:

- Técnicas de *Threshold* (limiarização)

- Métodos baseados em detecção de bordas
- Técnicas baseadas no Crescimento de Regiões

De acordo com Schalkoff (1989) a segmentação pode ser realizada usando-se duas abordagens:

- Não contextual : a relação entre as características dos pixels ou regiões são ignoradas. As técnicas que se enquadram nesta classificação são o reconhecimento de padrão (*pattern recognition*) usando vetor baseado na intensidade dos níveis de cinza ou a limiarização (*thresholding*) que separa os pixels em dois níveis de cinza.

- Contextual: a segmentação é realizada utilizando-se as relações de vizinhança entre os pixels, pois a decisão é tomada levando-se em conta não somente o pixel analisado, mas também os pixels que fazem parte da sua vizinhança. As técnicas que se enquadram nesta classificação são a detecção de borda ou fronteira e o crescimento de região.

1.4.1. Limiarização (*Thresholding*)

Segundo Gonzales e Woods (2002), o processo de limiarização consiste na composição de áreas na qual os pixels que possuírem um nível de cinza que esteja dentro de um dos intervalos, recebam um valor uniforme de intensidade. Geralmente, os valores adotados para representar as regiões são o zero e o nível máximo da escala de cinza dependendo da resolução de contraste da imagem. Com a utilização destes valores a imagem segmentada utilizará a cor branca e a cor preta.

Matematicamente, a função pode ser representada pela equação 16, onde se o nível de cinza de um pixel for maior do que o limiar T ele é denominado um ponto do objeto, caso contrário, é considerado um ponto pertencente ao fundo da imagem.

$$g(x,y) = \begin{cases} 1 \text{ se } f(x,y) > T \\ 0 \text{ se } f(x,y) \leq T \end{cases} \qquad (16)$$

A demonstração desta técnica está representada na Figura 14.

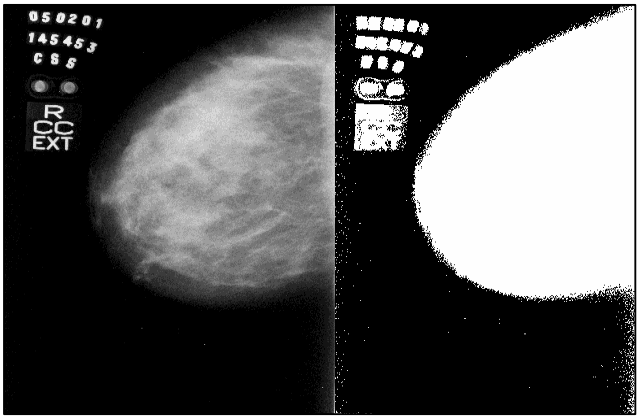


Figura 14 – Aplicação da Limiarização

1.4.2. Detecção de Bordas

Segundo Schalkoff (1989) e Santos (2002), uma borda pode ser definida como descontinuidade do nível de cinza em pixels vizinhos, que podem ter a característica de pontos isolados, segmentos ou curvas. O processo de detecção utiliza os gradientes do nível de cinza da imagem.

a) Operador Gradiente

Para uma imagem *f(x,y)*, o gradiente de *f* nas coordenadas *x* e *y* pode ser definido de acordo com a equação 17 e sua magnitude de acordo com a equação 18

$$\nabla f = \begin{bmatrix} \dfrac{\partial f}{\partial x} \\ \dfrac{\partial f}{\partial y} \end{bmatrix} \tag{17}$$

$$\nabla f = mag(\nabla f)\left[\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2\right]^{1/2} \tag{18}$$

Gonzales e Woods (2002) afirmam que o vetor gradiente aponta na direção de mudança mais rápida de *f* na posição *(x,y)*. Quando se trabalha com deteção de bordas, esse resultado é importante e comumente é chamado de gradiente e está representado na equação 19.

$$|G| = \sqrt{Gx^2 + Gy^2} \tag{19}$$

Quando a detecção utilizada for a de pontos isolados, um ponto será considerado aceito pelo algoritmo quando ocorrer uma variação drástica do valor de cinza em relação aos seus vizinhos. Quando a detecção for por linhas, será necessário achar os pixels que têm características comuns e verificar se eles são contínuos para assim formar uma linha. A detecção de pontos isolados e a detecção por linhas estão representadas na Figura 15.

(a)

(b)

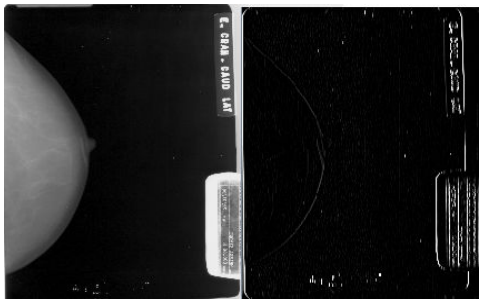


Figura 15 – (a) Exemplo da detecção de pontos isolados e (b) Deteção de linhas verticais

A detecção de bordas ocorre quando é preciso realçar o contraste entre a borda e o fundo, verificando-se os níveis de cinza de cada pixel da imagem. Este método é caracterizado pelo uso de *templates* ou máscaras como, por exemplo Sobel, Roberts, etc.

b) Operadores de Roberts

É o *template* mais antigo a ser utilizado para a detecção de bordas. Ele trabalha de forma simples, rápida para calcular o gradiente de um pixel na imagem. O cálculo do gradiente destaca as regiões que possuem um nível de freqüência espacial alto, que corresponde em grande parte dos resultados à borda de uma imagem.

Os operadores de Roberts são mostrados na Figura 16

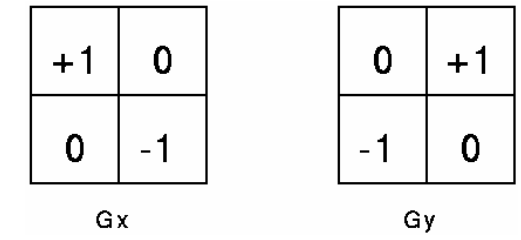


Figura 16 – Operadores de Robert para detecção de bordas

c) Operadores de Sobel

Os operadores de Sobel calculam o valor absoluto aproximado do gradiente em cada ponto da imagem analisada, deixando em maior evidência as áreas cuja freqüência espacial possui um valor alto e que correspondem às bordas da imagem. Os operadores de Sobel são apresentados na Figura 17.

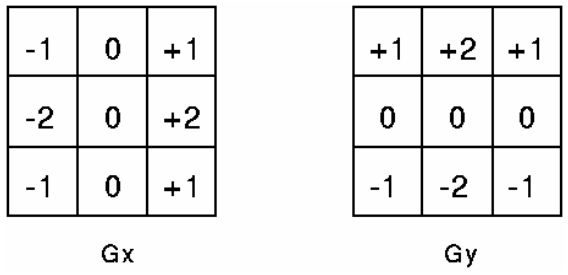


Figura 17 – Operadores de Sobel para detecção de bordas

1.5. Crescimento de Região

Segundo Gonzales e Woods (2002), este método consiste em agregar conjuntos de pixels em regiões maiores. A aproximação de processamento mais simples é a agregação de pixels, isto é: escolhe-se um pixel ou um conjunto de pixels denominados “sementes” e faz-se o crescimento da região através da agregação de pixels vizinhos às sementes que possuem propriedades similares (intensidade, cor, textura etc). O processo continua até se atingir uma condição pré-estabelecida de parada, como por exemplo, um determinado nível de cinza ou uma distância específica.

A vantagem é que a imagem não precisa ser homogênea, pois as suas características são previamente analisadas e incluídas nos descritores de semelhança. As principais desvantagens são: dificuldade na seleção dos pixels sementes (a aplicação deve ser conhecida); dificuldade no estabelecimento das propriedades de semelhança (a aplicação e os tipos de dados da imagem devem ser conhecidos) e dificuldade na determinação de condições de parada (depende da análise da imagem). Na Figura 18 é mostrado um exemplo de segmentação de imagem através da técnica de crescimento de região.

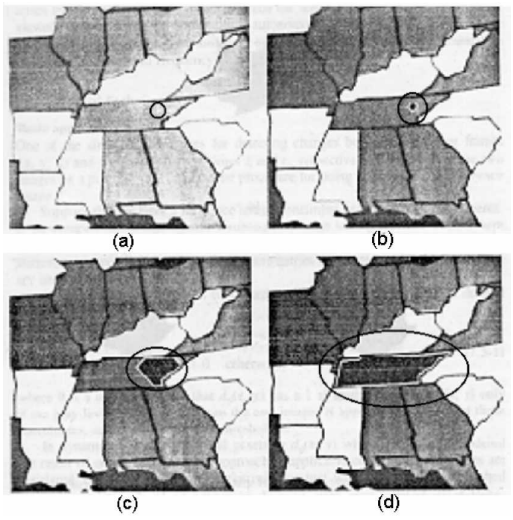


Figura 18 – Aplicação do crescimento de região, Gonzales e Woods (2002).

1.6. Considerações Finais

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

O processo de detecção da área que contém o nódulo se resume basicamente em aplicar a técnica de crescimento de região; entretanto, é necessário aplicar várias outras operações para realizar a remoção do fundo da imagem.

O próximo capítulo apresentada as principais características dos sistemas desenvolvidos para auxílio ao diagnóstico

CAPÍTULO 2. DETECÇÃO DE NÓDULOS EM MAMOGRAFIAS DIGITALIZADAS

Existem diversos estudos para a construção de sistemas de auxílio ao diagnóstico em câncer de mama, como as pesquisas desenvolvidas pelo *Kurt Rossman Laboratories for Radiologoc Image Research* da Universidade de Chicago, ou do *Germam Câncer Research Center*. Estes centros de Pesquisa têm como objetivo principal criar possibilidades para uma maior gama de opções no tratamento do câncer com o mínimo de inconveniências para o usuário, além de alcançar resultados que complementem as informações reunidas pelo médico ao analisar um caso.

Sob uma perspectiva mais abrangente, o câncer é definido por Sampat *et al.,* (2005a) como o crescimento desordenado de células no nosso organismo, criando assim massas que são denominadas neoplasias ou tumores.

As neoplasias podem ser caracterizadas como malignas e ocorrem quando o crescimento é desordenado e acelerado além de se infiltrar em diversas partes do corpo, que é a principal característica das neoplasias malignas que são denominadas metástase.

Geralmente, o objeto de pesquisa para os sistemas de auxílio ao diagnóstico em câncer de mama que são desenvolvidos é a imagem resultante da mamografia, que é definida por Nunes (2001) como uma técnica especial de radiografia com o objetivo de registrar as imagens da mamas utilizando níveis de tensões em intervalos específicos para detecção de formas indicativas de doenças, como o câncer.

2.1. Conceitos Básicos

Dentro do processo de análise de um mamograma, a etapa de detecção de áreas suspeitas é uma das mais importantes, pois tem o objetivo de auxiliar o

radiologista na seleção de anormalidades que podem ser classificadas como massas, microcalcificações ou distorções arquiteturais. (SAMPAT *et al.,*2005a)

Um dos principais motivos da importância da seleção das áreas de interesse ou *Regions of Interest* (ROI), é a sua enorme similaridade em um grande número de casos com o tecido da mama. (NISHIKAWA *et al.,*2000) Pode-se destacar um caso em especial sendo a análise de anormalidades em mamas densas, que geralmente está inserida dentro de uma faixa etária de mulheres jovens e representam um grande contingente de casos em potencial. A alta densidade do tecido da mama constitui um fator muito importante na elaboração da nitidez da imagem, como também diminui o contraste da mamografia, necessitando de etapas extras no processamento. (NUNES 2001; SANTOS 2002 ; NETO *et al.*, 2004).

2.2. Sistemas de auxilio ao diagnóstico

Existem inúmeras técnicas e algoritmos desenvolvidos para extração de ROIs que possuem como principal objetivo oferecer melhores parâmetros para a elaboração de um diagnóstico mais robusto. Destacam-se os esquemas ilustrados na Figura 19, proposto por Sampat *et al.,* (2005a) e o esquema ilustrado na Figura 20, proposto por Singh *et al.* (2002). Na ilustração de Sampat *et al.,* (2005a), um fluxograma que mostra as etapas principais envolvidas em sistemas de auxílio à detecção e no sistema de auxilio ao diagnóstico. A maioria dos algoritmos da detecção consiste de dois estágios. No estágio 1 o alvo é detectar as áreas suspeitas com um alto grau de sensibilidade. No estágio 2 o alvo é reduzir o número de diagnósticos falsos positivos sem diminuir a sensibilidade. As etapas que são envolvidas na execução dos algoritmos para ambos os estágios são mostradas na Figura 19 (b).

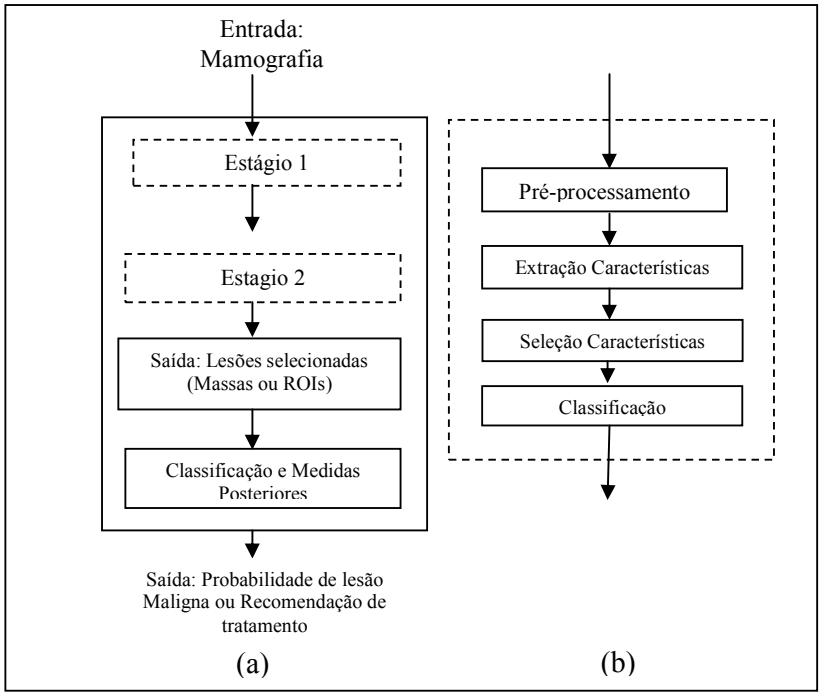


Figura 19 – Esquema CAD proposto por Sampat *et al.* (2005a)

Na Figura 20, apresentada por Singh *et al.* (2002) , são demarcadas as etapas que compõem a detecção e a classificação das áreas de interesse.

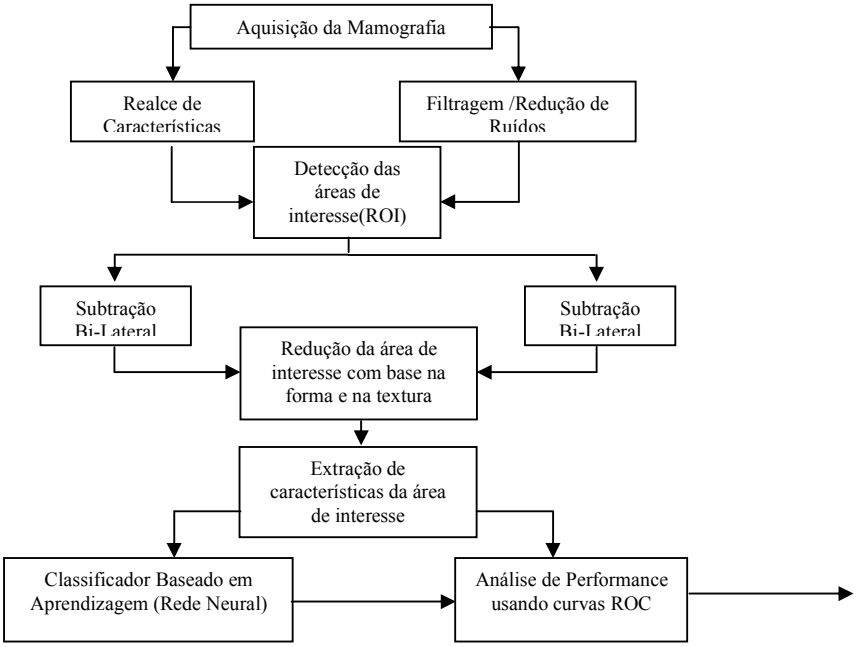


Figura 20 – Esquema CAD adaptado de Singh *et al.* (2002)

Outros exemplos de Sistemas CAD que não seguem necessariamente estes modelos, mas que são similares estão descritos em (Belloti *et al.,* 2006); Tweed e Miguet, (2002); Adorno *et al.* (2002);Sampat *et al,* (2005a) ;Santos (2002); (Zwiggelaar *et al.,* 1999); Sampat *et al.* (2005c); Junior *et al.* (2006); Chan *et al.* (2005); Poel *et al*. (2006).

Além da detecção precoce de uma lesão, um fator importante no projeto de um sistema CAD está relacionado ao desempenho, que consiste na porcentagem de lesões que são demarcadas corretamente pelo sistema.

Segundo Sampat *et al.,* (2005a), existem discussões sobre como os resultados obtidos na detecção podem ser usados para mensurar a performance de um sistema CAD. Uma grande maioria realiza cálculos com base no número de todas ROIs selecionadas, não importando a sua espécie, enquanto que alguns trabalhos estão usando como base o número de ROIs que são consideradas malignas, já que as ROIs malignas são os principais elementos a serem detectados (Sampat *et al.,* 2005a).

Uma forma de avaliação que tem se tornado amplamente aceita é o cálculo da sensibilidade e especificidade do CAD e a conseqüente construção da curva ROC (*Receiver Operating Characteristic)*, a partir destas duas medidas.

Braga (2000) define sensibilidade como a capacidade que o sistema possui de detectar a área de interesse no indivíduo e é calculada como a razão entre o número de decisões Verdadeiro-Positivos (VP) e o total de imagens realmente positivas, conforme equação 21. A especificidade é definida como a capacidade com que um sistema descarta os indivíduos que não possuem características significativas da doença, e que é calculada como a razão entre o número de decisões verdadeiras negativas e o total de casos realmente negativos, conforme equação 22**.**

$$\text{Sensibilidade} = \frac{\text{Número de decisões verdadeiro positivas}}{\text{Número de casos realmente positivos}} \quad (21)$$

$$\text{Especificidade} = \frac{\text{Número de decisões verdadeiras negativas}}{\text{Número de casos realmente positivos}} \quad (22)$$

2.3. Detecção de nódulos

Um nódulo, também conhecido como massa, é classificado em uma mamografia como uma lesão que pode ser detectada a partir de duas visões ou projeções distintas (Sampat *et al.,* 2005a).

Os parâmetros necessários para a detecção de uma massa devem ser aqueles que são mais acentuados em massas, em comparação com o tecido da mama (Nishikawa *et al.,* 2000). Como exemplo, alguns parâmetros importantes que devem ser definidos e realçados para posterior análise de um sistema CAD são a forma da lesão, a definição da borda, e a assimetria, quando há uma comparação com a mamografia da mama esquerda com a direita, por exemplo.

Dentre os esquemas citados na seção anterior, enquanto alguns voltaram os seus esforços para a detecção de qualquer tipo de massa ou nódulo, outros se concentraram no estudo de casos particulares como o caso das lesões espiculadas (Sampat *et al.,* 2005a), (Zwiggelaar *et al.,* 1999), que é um caso bastante visado devido à grande importância de que geralmente uma lesão espiculada apresenta. Quando estes tipos de lesão são detectados, existe uma grande chance de ser um câncer maligno, ou o caso de estudo em mamas densas, devido à enorme semelhança com o tecido fibroglandular da mama (SANTOS (2002),SAMPAT *et al.,* (2005a)).

A seguir são apresentadas técnicas de Detecção Baseada em Pixels e Detecção Baseada em Regiões.

2.3.1. Detecção baseada em métodos de análise pixel-a-pixel

Nas técnicas ou categorias que se enquadram nesta classificação, as imagens mamográficas são analisadas pixel-a-pixel, sendo extraída delas características relevantes como o nível de intensidade dos pixels ou a textura.

Feita a verificação, o pixel é classificado como suspeito ou não. Pode-se realizar este processo com um simples *Thresholding* (Limiarização), ou com técnicas mais complexas. (Sampat *et al.,* 2005a)

Na análise pixel-a-pixel foram desenvolvidos diversos métodos e algoritmos para os diversos tipos de nódulos, como os que possuem características que são próprias, como as lesões espiculadas.

Adorno *et al*. (2002) aperfeiçoaram dois métodos propostos por Guliato (1998), que detectam as massas utilizando o crescimento de região, utilizando como características principais os níveis de cinza na região das bordas selecionadas. O primeiro método faz a seleção das áreas de interesse (ROI), determinando o contorno da região. O aperfeiçoamento proposto é que ao se utilizar o crescimento de região, ao invés de se selecionar somente um pixel que representaria a área selecionada, denominado "pixel semente", fossem selecionados vários pixels, criando assim uma "região semente", que proporciona à etapa de detecção um melhor acerto nas áreas selecionadas, se comparado com a seleção feita por um radiologista.

O segundo método também utiliza o crescimento de região baseada nos conjuntos difusos, tratando da incerteza presente na região de transição da borda.

Junior *et al*. (2006), desenvolveram um trabalho que realiza a detecção das áreas de interesse com base na extração de características de textura de cada pixel através de métodos estatísticos, como a matriz de co-ocorrência dos níveis de cinza, que é um dos métodos mais utilizados. Também são extraídas características estatísticas de segunda ordem da matriz de co-ocorrência, as características de Haralick, que dentre outras são: Contraste, Energia, Entropia e Homogeineidade, conforme mostram as equações 23, 24, 25 e 26.

$$\text{Contraste} = \sum_{i=0}^{G-1}\sum_{j=0}^{G-1} \mathrm{P}_{ij}(i-j)^2 \tag{23}$$

$$\text{Energia} = \sum_{i=0}^{G-1}\sum_{j=0}^{G-1} \mathrm{P}_{ij}{}^2 \tag{24}$$

$$\text{Entropia} = -\sum_{i=0}^{G-1}\sum_{j=0}^{G-1} \mathrm{P}_{ij}\log(\mathrm{P}_{ij}) \tag{25}$$

$$\text{Homogeneidade} = \sum_{i=0}^{G-1}\sum_{j=0}^{G-1} \frac{(\mathrm{P}_{ij})^2}{1+|i-j|} \tag{26}$$

A segmentação e a delimitação das áreas de interesse são feitas com o algoritmo K-Means, que realiza a segmentação por agrupamento. O algoritmo demarca K centros, cada um com o pixel central servindo de referência sobre a região, tendo uma importância muito elevada na sua escolha.

Após a seleção da região, determina-se a região de propriedade de cada centro, com o agrupamento de elementos que se adequam às propriedades de determinada região. Após a primeira etapa, inicia-se a etapa interativa do algoritmo, na qual os pixels de cada centro são recalculados de modo a reduzir a capacidade de seleção da função

objetivo, que é definida pelo cálculo dos mínimos quadrados (equação 27), que define a distância do pixel analisado até o pixel que está no centro da região.

$$\text{MQ} = \sum_{j=1}^{k} \sum_{i=1}^{n} \left\| x_i^{(j)} - c_j \right\|^2 \tag{27}$$

A detecção baseada na análise pixel-a-pixel tem a vantagem de possuir inúmeras técnicas que já foram desenvolvidas, podendo-se aperfeiçoá-las. Mas deve-se levar em conta o arranjo espacial dos pixels na imagem, que constitui um fator de extrema importância na distinção das áreas de interesse selecionadas do tecido da mama (SAMPAT *et al.,* 2005a).

2.3.2. Detecção baseada em métodos de análise por região

Na detecção por regiões tem-se que as áreas de interesse podem ser selecionadas com base em um *thresholding* ou algum filtro de análise. As características mais estudadas e que têm a maior relevância na detecção são a forma da área e a textura.

Depois da divisão da imagem em regiões, as características são extraídas de cada uma delas.

Belloti *et al.,* (2006) desenvolveram um método de detecção baseado em regiões, que é aplicado em mamografias no qual a mama deve estar localizada do lado direito. A primeira etapa do algoritmo é a delimitação da área na mamografia que corresponde à mama, realizando um *thresholding* linha por linha. Os pixels que se encontram à esquerda do pixel selecionado para ser limiar recebem o valor zero. Desta forma delimita-se o contorno da mama.

Na segmentação da imagem, a área é dividida em sub-regiões de tamanhos iguais e é verificado se os pixels da sub-região possuem o nível de cinza semelhante ao pixel central da sub-região, que é caracterizado por Y.

Um contorno é delimitado com um *thresholding* cujo limiar é definido pela equação (28), que demarca a área da região selecionada. O limiar Z é incrementado ou decrementado, até que o valor entre dois limiares de etapas consecutivas seja menor que a média dos níveis selecionados.

A extração forma a matriz de co-ocorrência, que utiliza os coeficientes de segunda ordem para a classificação da região como uma massa ou um tecido normal, como energia, entropia, etc.

$$Z = Y/2 \tag{28}$$

Tweed e Miguet (2002), detectaram regiões de interesse com base na análise das características da textura e do histograma com máscaras circulares.

Para extração das características, são utilizadas máscaras de 70, 150 e 300 pixels sendo que as características extraídas devem atender a três aspectos que são:

- seletividade, onde certo número de pixels deve ser mantido na imagem;
- pertinência onde uma área que contem uma lesão não pode ser excluída da análise ;
- homogeneidade onde a variação das características da área selecionada deve ser menor do que a variação na área inteira.

O processo de seleção é composto pelo cálculo da distância euclidiana entre a máscara circular e a região selecionada, e por duas etapas de *thresholding*.

A extração de características pelo histograma também se mostrou importante, pois uma grande quantidade de pixels que fazem parte de uma região selecionada possui

níveis de cinza elevados. São realizadas duas etapas de *thresholding* para delimitar a ROI que possui muitos pixels com valor do nível de cinza alto. A máscara mais efetiva foi a de 150 pixels, mas a extração de características pelo histograma é mais seletiva, porém, nem as características de textura, nem as do histograma detectaram ROIs pequenas.

2.4. Considerações Finais

Neste capítulo foram apresentados os principais conceitos sobre a detecção de *Regions of Interest* (ROI) em exames de diagnóstico de câncer de mama. Foram apresentados os principais aspectos para a elaboração de um sistema CAD. Trabalhos que utilizam diversas técnicas e abordagens foram exemplificados demonstrando que para cada tipo de lesão é possível escolher um conjunto de técnicas que retornam resultados satisfatórios.

CAPÍTULO 3. TECNOLOGIAS UTILIZADAS NA IMPLEMENTAÇÃO

3.1. Linguagem JAVA

3.1.1. Compilador

Um programa em Java é compilado para o chamado "byte-code", que é próximo às instruções de máquina, mas não de uma máquina real. O "byte-code" é um código de uma máquina virtual idealizada pelos criadores da linguagem. Por isso Java pode ser mais rápida do que se fosse simplesmente interpretada. (MANOEL 2003)

3.1.2. Portabilidade de plataforma

Java foi criada para ser portável. O "byte-code" gerado pelo compilador para a sua aplicação específica pode ser transportado entre plataformas distintas que suportam Java (Solaris 2.3, Windows-NT, Windows-95, Mac/Os etc). Não é necessário recompilar um programa para que ele rode numa máquina e sistema diferentes, ao contrário do que acontece, por exemplo, com programas escritos em C e outras linguagens.

Esta portabilidade é importante para a criação de aplicações para a heterogênea Internet. Em Java um inteiro, por exemplo, tem sempre 32 bits, independentemente da arquitetura. O próprio compilador Java é escrito em Java, de modo que ele é portável para qualquer sistema que possua o interpretador de "byte-codes". (MANOEL 2003)

3.1.3. Orientação a Objetos

A portabilidade é uma das características que se inclui nos objetivos almejados por uma linguagem orientada a objetos. Em Java ela foi obtida de maneira inovadora com relação ao grupo atual de linguagens orientadas a objetos.

Java suporta herança, mas não herança múltipla. A ausência de herança múltipla pode ser compensada pelo uso de herança e interfaces, onde uma classe herda o comportamento de sua superclasse além de oferecer uma implementação para uma ou mais interfaces. (MANOEL 2003)

3.1.4. Segurança

A presença de coleta automática de lixo, evita erros comuns que os programadores cometem quando são obrigados a gerenciar diretamente a memória (C , C++, Pascal). A eliminação do uso de ponteiros, em favor do uso de vetores, objetos e outras estruturas substitutivas traz benefícios em termos de segurança.

O programador é proibido de acessar a memória que não pertence ao seu programa, além de não ter chances de cometer erros comuns tais como "reference aliasing" e uso indevido de aritmética de ponteiros. Estas medidas são particularmente úteis quando pensarmos em aplicações comerciais desenvolvidas para a Internet.

A presença de mecanismos de tratamento de exceções torna as aplicações mais robustas, não permitindo que elas abortem, mesmo quando rodando sob condições anormais. O tratamento de exceções é bastante útil para modelar situações tais como falhas de transmissão e formatos incompatíveis de arquivos. (MANOEL 2003)

3.1.5. Eficiência

Como Java foi criada para ser usada em computadores pequenos, ela exige pouco espaço, pouca memória. Java é muito mais eficiente que grande parte das linguagens existentes, embora seja cerca de 20 vezes mais lenta que C, o que não é um marco definitivo. (MANOEL 2003)

Com a evolução da linguagem, serão criados geradores de "byte-codes" cada vez mais otimizados que trarão as marcas de performance da linguagem mais próximas das de C++ e C. Além disso um dia Java permitirá a possibilidade de gerar código executável, tudo a partir do "byte-code". (MANOEL 2003)

3.1.6. Classes existentes

Um dos grandes atrativos da programação em Java tambem é a possibilidade de adaptar funcionalidades oferecidas em classes existentes às necessidades de cada aplicação. Java oferece essa facilidade, aliando-a ao oferecimento de um amplo conjunto de classes organizadas nos pacotes da API padrão. (MANOEL 2003)

Juntamente com esta linguagem de programação foi utilizada a API (*Application Program Interface*) JAI (*Java Advanced Imaging*) (SUN, 2007) que, segundo Santos (2004), possibilita a representação, o processamento e a visualização de imagens.

3.2. API JAI (*Java Advanced Imaging*)

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.

Segundo ROSSI (2005) ela 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, será possível realizar o realce rapidamente.

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.

Segundo ROSSI (2005), 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, 2007).

3.3.1. Armazenamento e representação

De acordo com ROSSI (2005), na JAI a classe básica para representar uma imagem é a classe *PlanarImage*, que 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 de leitura. Para a alteração dos pixels deve ser usada uma subclasse da *PlanarImage*, a *TiledImage*.

A implementação de "*TiledImages"* permite que uma imagem seja 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).

3.3.2. Operações com imagens

A Figura 21 mostra o método *create*, que é o método principal da classe JAI.

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

Figura 21 – Utilização do método Created 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 22 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 22 – Utilização da classe PlanarImage da 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 22. Esses valores são passado por uma classe chamada *ParameterBlock*) (SUN, 2007).

A Figura 23 tem um exemplo do uso do *ParameterBlock*, onde uma imagem tem seu tamanho dobrado na largura e triplicado na altura.

```
ParameterBlock pb = new ParameterBlock();
pb.addSource(imagenEntrada);
pb.add(2.0f);
pb.add(3.0f);
pb.add(0.0f);
pb.add(0.0f);
pb.add(new InterpolationNearest());
PlanarImage imagemSaida = JAI.create("scale", pb);
```

Figura 23– Realização de escala com o ParameterBlock

Ao todo o *ParameterBlock pb* recebe 6 argumentos. O primeiro parâmetro adicionado é a imagem que será tratada, os próximos dois são os valores da escala, os dois seguintes são os valores da translação e o último é uma instância da classe *InterpolationNearest* que fará a interpolação dos pixels durante o processo de escalar a imagem.

3.3.3. Visualização

A API JAI disponibiliza a 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 é utilizado o código da Figura 24, onde "imagem" é uma instância de *PlanarImage* ou de *TiledImage*

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

Figura 24– Exibição de uma imagem na API JAI

3.3.4. 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 (SUN, 2007).

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.

3.3.5. 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 (SUN, 2007).

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).

Os formatos de arquivos que a API JAI suporta para armazenar as imagens são:

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.

3.3. Considerações Finais

Java é uma linguagem poderosa e excelente para implementar aplicações que necessitam de uma boa interação com o usuário. A API JAI é usada na análise e processamento de fotografias digitais em diversas áreas como as pesquisas biomédicas e processamento de dados geo-espaciais.

A linguagem Java, junto com a API JAI formam um conjunto poderoso para o desenvolvimento e implementação da aplicação proposta.

No próximo capítulo são descritas as etapas de desenvolvimento do sistema.

CAPÍTULO 4. SISTEMA IMPLEMENTADO

4.1. Introdução

Este trabalho consiste na implementação de um algoritmo de detecção de nódulos em mamografias digitalizadas.

Posteriormente, este trabalho poderá ser integrado em um esquema computadorizado para auxílio ao diagnóstico de mamografias digitais, que utiliza técnicas de processamento de imagens digitais para identificar, realçar e classificar estruturas de interesse clínico.

O sistema implementado neste trabalho é uma aplicação Java independente de plataforma e relativamente de baixo custo computacional. A implementação foi feita com base nas informações sobre o desenvolvimento de sistemas para auxílio a diagnóstico, que estão descritos no capítulo 2, e com informações vindas da documentação da API JAI encontrada em Sun (2007).>>>>>.

Esta aplicação foi testada na plataforma Microsoft Windows XP, utilizando um processador de 1,8 GHz e 512 MB de Memória.

4.2. Características das Imagens

As imagens utilizadas neste projeto fazem parte de um banco de imagens desenvolvido pelo LAPIMO (Laboratório de Processamento de Imagens Médicas e Odontológicas, a EESC/USP. Na digitalização das Imagens foram utilizados dois digitalizadores a laser, ambos da marca Lumisys (Lumiscan 50 e Lumiscan 75) e que,

segundo o fabricante, possibilitam obter imagens com até 12 bits de resolução de contraste (4096 níveis de cinza).

4.3. Desenvolvimento

Para a implementação da aplicação e obtenção dos resultados foram executados os passos mostrados na Figura 25.

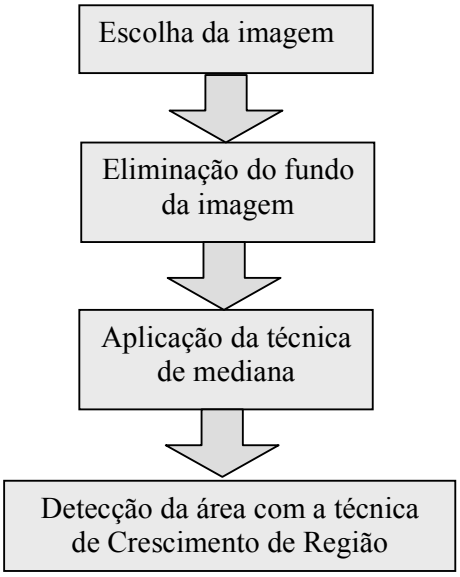


Figura 25 – Esquema geral para implementação do sistema

A imagem é escolhida digitando-se o nome na caixa de diálogo inicial do sistema, como está ilustrado na Figura 26.

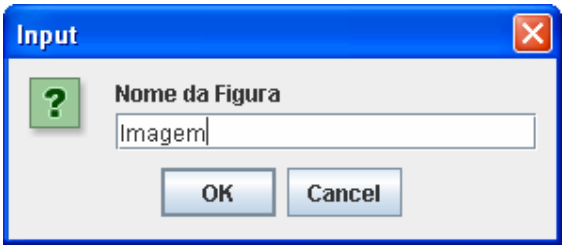


Figura 26 – Inserção do nome da imagem a ser processada

Com a imagem escolhida ela é carregada e o algoritmo para a remoção do fundo da imagem Santos (2006), é executado retirando todas as informações que não

são relevantes uma vez que as características que serão obtidas das imagens devem ser computadas considerando apenas a área da mama, e a imagem resultante é salva para o processamento posterior. Um exemplo da execução do algoritmo de remoção do fundo é ilustrado na Figura 27, e o trecho de código que efetua o salvamento da imagem é ilustrado na Figura 28.

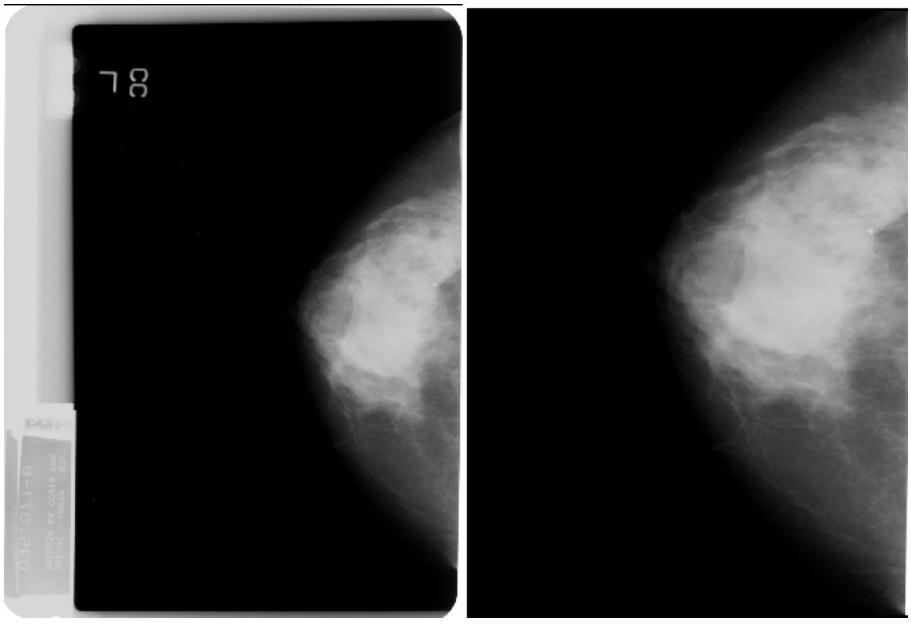


Figura 27 – Imagem Original e Imagem Resultante do processamento para remoção do fundo.

```
for (h=0; h<height; h++)
    for (int w=0; w<width; w++)
    {
        offset = h*width*nbands+w*nbands;
        offset2 = h*width*nbands2+w*nbands2;
        for (int band=0; band<nbands; band++)
            if (pixels[offset+band] == 0)
                pixels2[offset2+band] = 0;
            else
                pixels2[offset2+band] = pixels1[offset+band];
    }
mamograma.salvaImagem(pixels2,"sfundo.tiff");
```

Figura 28 – Trecho de código que armazena a imagem processada.

Após a imagem resultante ter sido armazenada, a aplicação a carrega automaticamente e aplica a técnica da mediana, que suaviza a imagem sem borrar em

demasia suas bordas, com intuito de induzir pontos com intensidades desiguais a se tornarem iguais aos seus vizinhos, suavizando pontos que sejam diferentes na imagem.

Com a aplicação da técnica da mediana, a imagem é exibida para o usuário na interface do programa do lado esquerdo, como mostra a Figura 29.

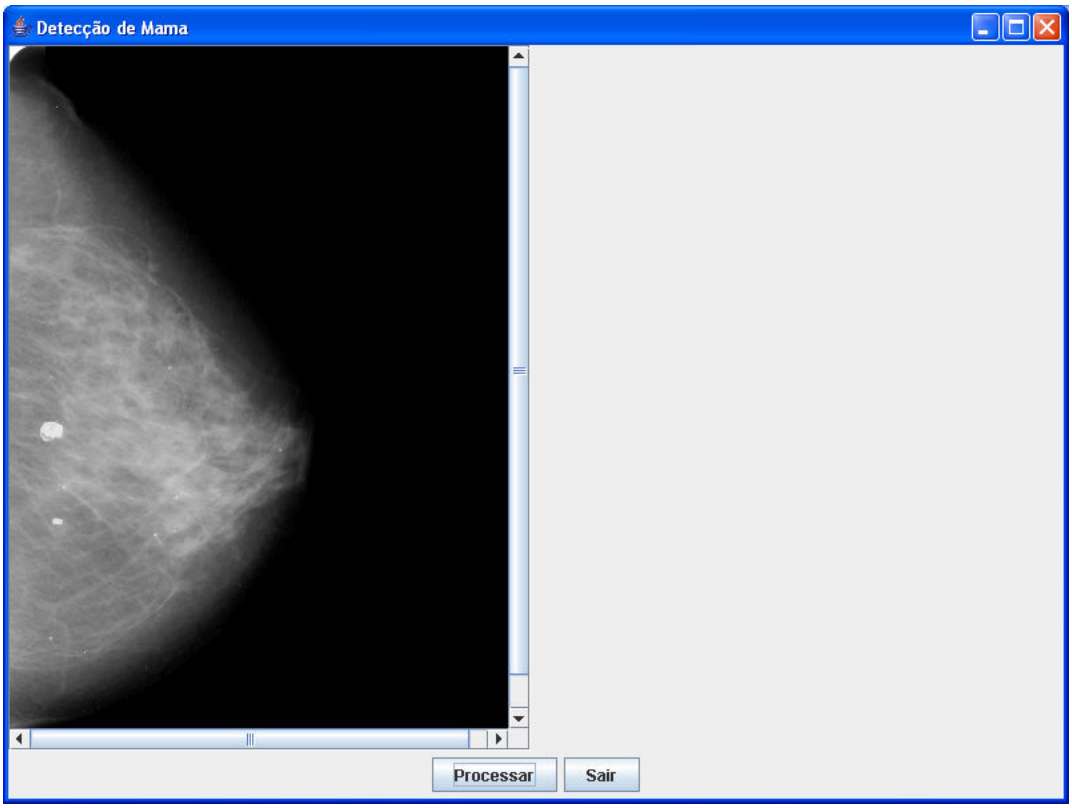


Figura 29 – Imagem processada com a técnica da mediana

Para a execução da técnica de Crescimento de Região, o usuário deve escolher o valor do nível de cinza que será usado como limiar após o cálculo que verifica a diferença entre o pixel analisado e os pixels que compõem a sua vizinhança, como mostra a Figura 30. Deve escolher, ainda, o tamanho em pixels do quadrante que será usado como modelo na busca da área que contém a maior média de nível de cinza, como mostra a Figura 31. Para a escolha destes valores, o sistema não faz nenhuma limitação ao usuário, pode-se inserir qualquer valor que o sistema retornará uma imagem resultante.

Os valores que retornam uma imagem com uma área próxima ao tamanho da área demarcada pelo radiologista variam de 15 a 50 para o limiar e de 40 a 100 para o tamanho do quadrante.

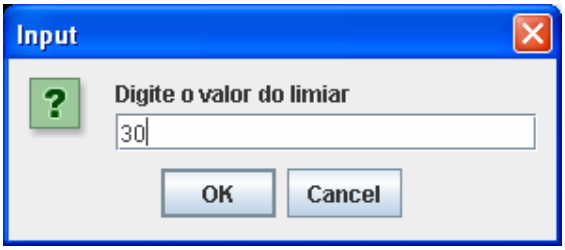


Figura 30 – Entrada do valor de limiar.

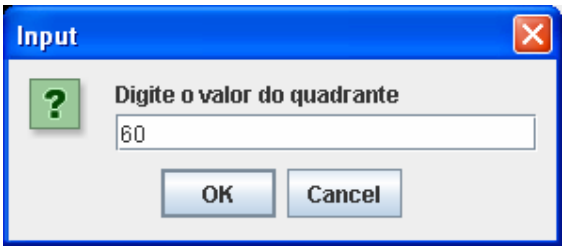


Figura 31 – Entrada do valor que determina o tamanho do quadrante a ser utilizado.

Com estes parâmetros informados pelo usuário, é feita uma análise na imagem em busca da área com o valor determinado pelo usuário que possui a maior média de nível de cinza da imagem.

A área que contiver a maior média de nível de cinza da imagem tem alta probabilidade de conter o nódulo, pois tem uma quantidade elevada de pixels com um alto valor de nível de cinza e que está ilustrado na Figura 32, com o trecho de código correspondente na Figura 33.

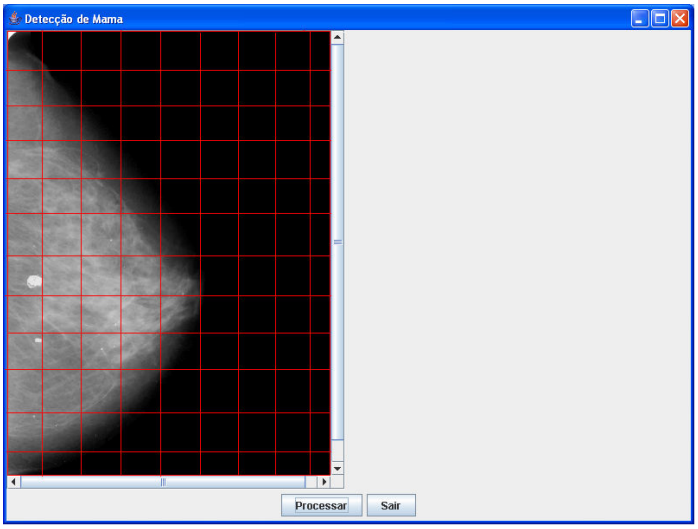


Figura 32 – Esquema de divisão da imagem com base no valor do quadrante

```
public int MediaQuadrante(){

  int altura   = image.getAltura();
  int largura  = image.getLargura();
  int nbands   = image.getNbans();
  int inicioQuadrante=0;
  FiltroMedia();
  String mask = JOptionPane.showInputDialog(null,"Digite o valor do quadrante");
  @SuppressWarnings("unused")
  int limiar = Integer.parseInt(JOptionPane.showInputDialog("Digite o valor do limiar"));
  int linhasQuadrante = Integer.parseInt(mask);
  int colunasQuadrate = Integer.parseInt(mask);

  float maiorMedia = mediaGlobal();

  for(int linhaGlobal = 0; linhaGlobal<altura; linhaGlobal +=linhasQuadrante)
    for(int colunaGlobal = 0; colunaGlobal<largura; colunaGlobal +=colunasQuadrate)
    {
      int   offset = (linhaGlobal*largura*nbands+colunaGlobal*nbands);
      int   somaVizinhanca = 0;

      for(int linhaVizinhanca = linhaGlobal;linhaVizinhanca<linhaGlobal+linhasQuadrante;
        linhaVizinhanca++)
        if(linhaVizinhanca < altura)
          for(int colunaVizinhanca = colunaGlobal;colunaVizinhanca<colunaGlobal+colunasQuadrate;
          colunaVizinhanca++)
          {
            if(colunaVizinhanca < largura)
              {
```

Figura 33 – Implementação da busca pelo quadrante com a maior média nível de cinza

Com a área encontrada é localizado o ponto central desse quadrante que será utilizado como ponto inicial no processamento da imagem com a técnica de crescimento de região.

O trecho de código que implementa a localização do ponto central do quadrante selecionado está ilustrado na Figura 34 e a Figura 35 ilustra qual ponto será selecionado.

```
//Localiza o ponto central de cada quadrante selecionado.

int sementel = (inicioQuadrante/(largura*nbands)) + linhasQuadrante/2;
int sementec = (inicioQuadrante-((inicioQuadrante/(largura*nbands))*largura*nbands))+ linhasQuadrante/2;
return  ((sementel*largura*nbands)+(sementec*nbands));
```

Figura 34 – Implementação da localização do ponto “semente”

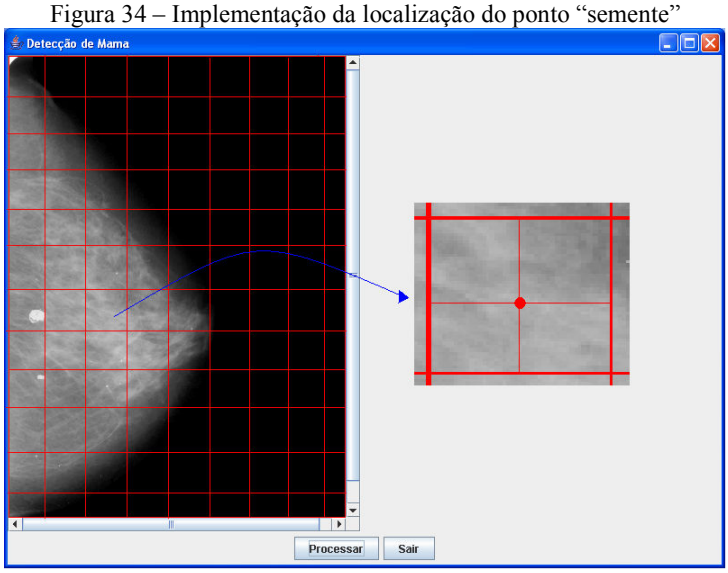


Figura 35 – Esquema da localização do ponto “semente”

Com o ponto central selecionado ele é usado como pixel “semente” no crescimento de região.

Com o Crescimento de Região em execução, a vizinhança do pixel “semente” é analisada e a diferença entre os níveis de cinza do pixel analisado, além disso, cada vizinho é comparado ao limiar digitado na caixa de diálogo ilustrado na Figura 30. Se um dos valores computados estiver abaixo do limiar estabelecido, então o pixel central da máscara analisado é adicionado à região, e os pixels que compõem a sua vizinhança são adicionados na lista de pixels que serão analisados pelo método.

O método é executado até que não existam mais pixels a serem analisados e então os pixels que foram selecionados e que formam a região recebem o valor

correspondente ao preto, formando a região existente de acordo com o limiar estabelecido.

O trecho de código que implementa o Crescimento de Região está ilustrado na Figura 36 e as imagens obtidas nas Figuras 37, 38, 39 e 40.

```
do{
    int semente = (Integer) listaVeriSemente.elementAt(0);

    while(vetorAux[semente] == 0){
        listaVeriSemente.remove(0);
        if(listaVeriSemente.isEmpty())
            break;
        semente = (Integer) listaVeriSemente.elementAt(0);
    }

    int altura = (semente/(image.getLargura()*nbands));
    int largura = (semente-((semente/(image.getLargura()*nbands))*image.getLargura()*nbands));
    int[] listaCandidatosSemente = new int[9];
    int aux=0;
    boolean quadranteValido = false;

    for (int i = (altura - 1);i <=(altura + 1);i++ )
        for (int j = (largura - 1);j <=(largura + 1);j++)
        {
            int offset = (i*image.getLargura()*nbands+j*nbands);
            if(semente != offset)
            {
                double limiarvari = Math.abs(pixels[semente] - pixels[offset]);
                if(limiarvari < limiar)
```

Figura 36 – Implementação do Crescimento de Região

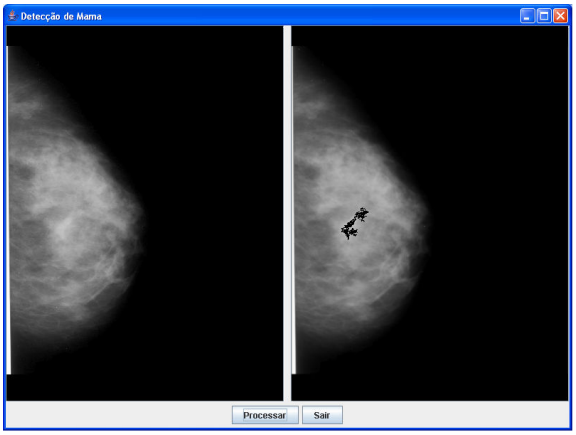


Figura 37 – Região selecionada na Imagem “999985291100CCE3.TIFF”

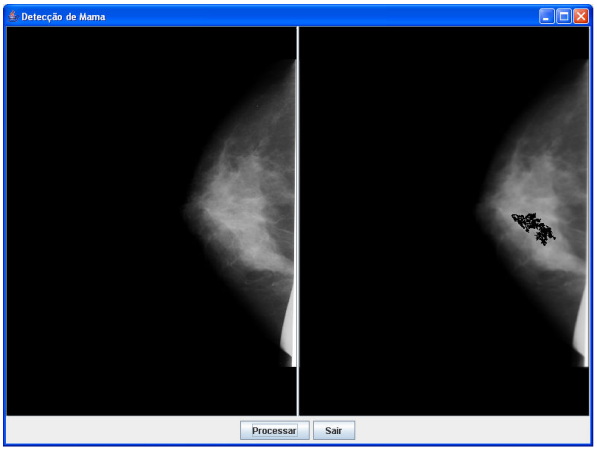


Figura 38 – Região selecionada na Imagem “50130230804CCD1.TIFF”

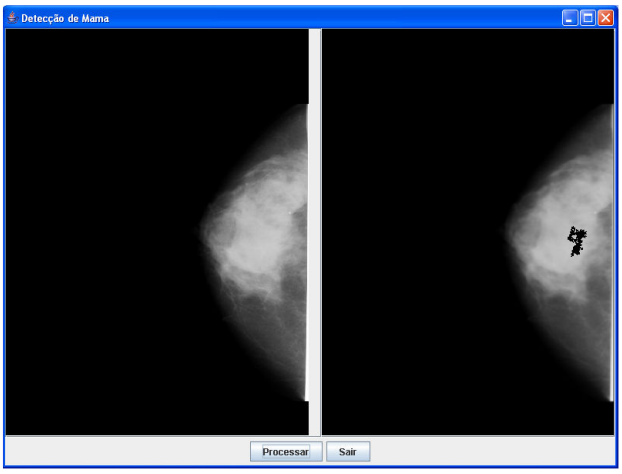


Figura 39 – Região Selecionada na Imagem “50130230804CCE3.TIFF”

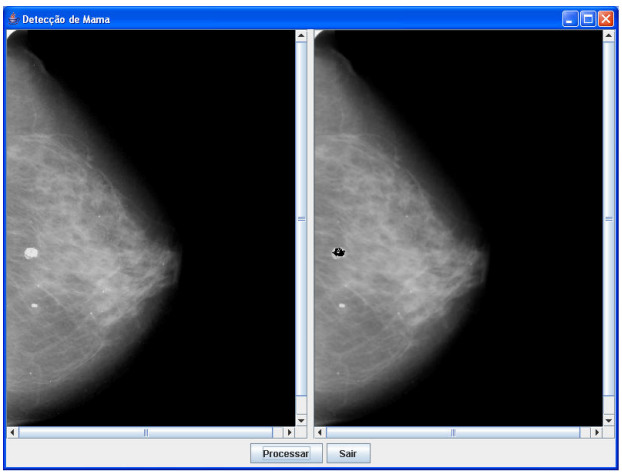


Figura 40 - Região Selecionada na Imagem “999984291100CCE3.TIFF”

CAPÍTULO 5. RESULTADOS E DISCUSSÕES

Para os testes do sistema foram utilizadas 11 imagens fornecidas pelo LAPIMO da EESC/USP provenientes de diversos hospitais, citados no item 2.2 desta monografia. Estas imagens foram digitalizadas com 16 bits de resolução de contraste (65536 níveis de cinza). Este conjunto é composto por imagens de visão crânio-caudal e médio-lateral, que estão exemplificadas na Figura 41..

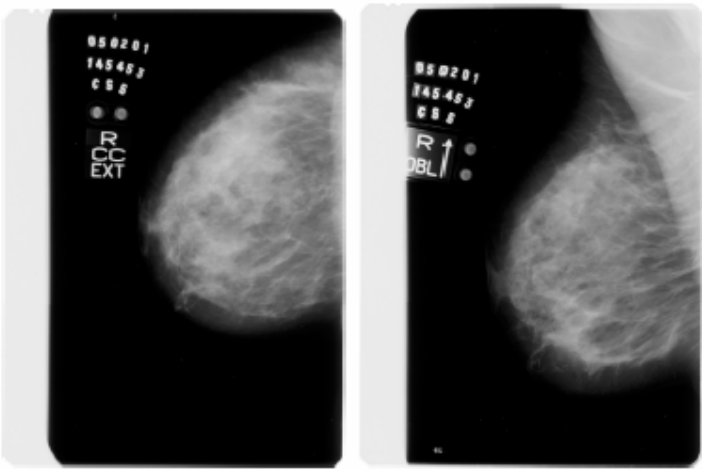


Figura 41 – Exemplo de imagens com visão crânio-caudal a esquerda e médio-lateral a direita

Durante a execução dos testes foi verificado que a utilização de valores elevados, que podem ser acima de 50 ou de valores muito baixos, que podem chegar a 10, para o limiar e também para o tamanho do quadrante (abaixo de 20 e acima de 100) não detectavam regiões internas da mama, detectando bordas ou da imagem ou da mama, gerando assim uma região que não corresponde a expectativa do sistema.

Quando os limiares utilizados para a comparação dos níveis de cinza calculados são baixos, a região era formada por um único pixel, não formando assim uma região contendo uma quantidade de pixels significativa, como mostra a Figura 42.

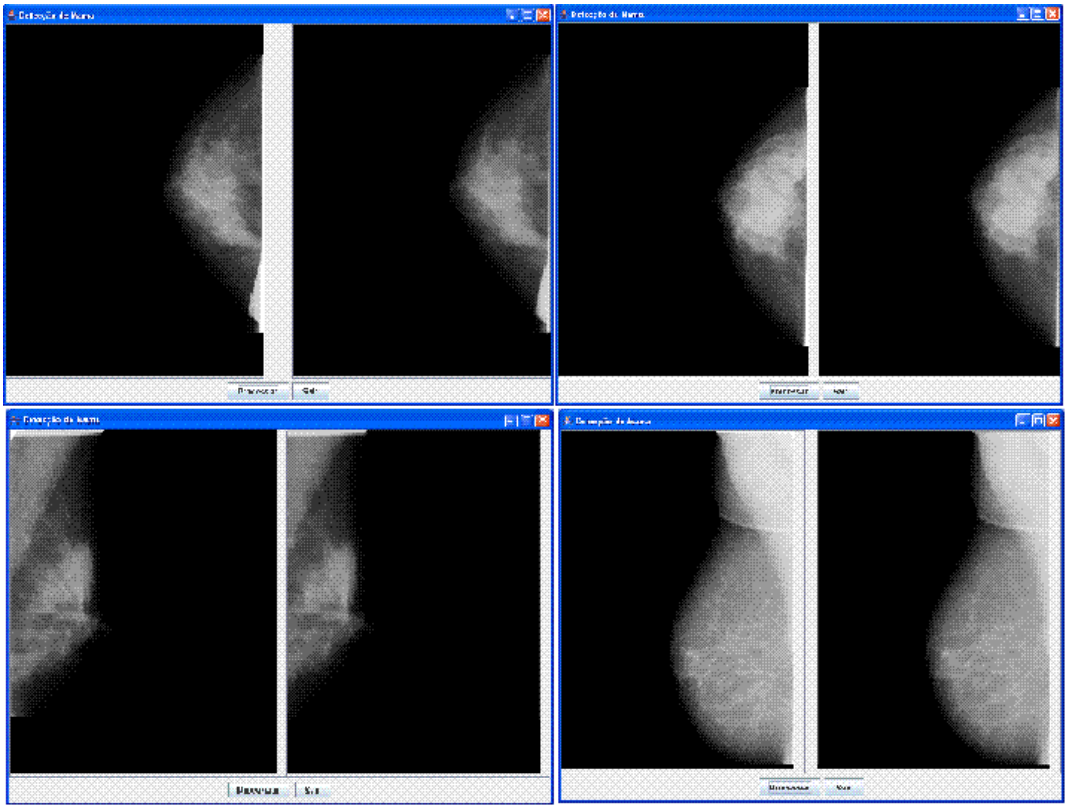


Figura 42 – Exemplo de imagens processadas com um limiar baixo

Na Figura 42, o limiar utilizado foi 10, e as imagens mostram que a imagem da direita que é a imagem processada teve a região criada somente com o pixel “semente”, pois o limiar baixo não possibilitou a inserção de novos pixels.

Quando o limiar é muito alto, a região é composta por inúmeros pixels que, criando uma região que poderia abranger uma grande parte da mama, como mostra a Figura 43.

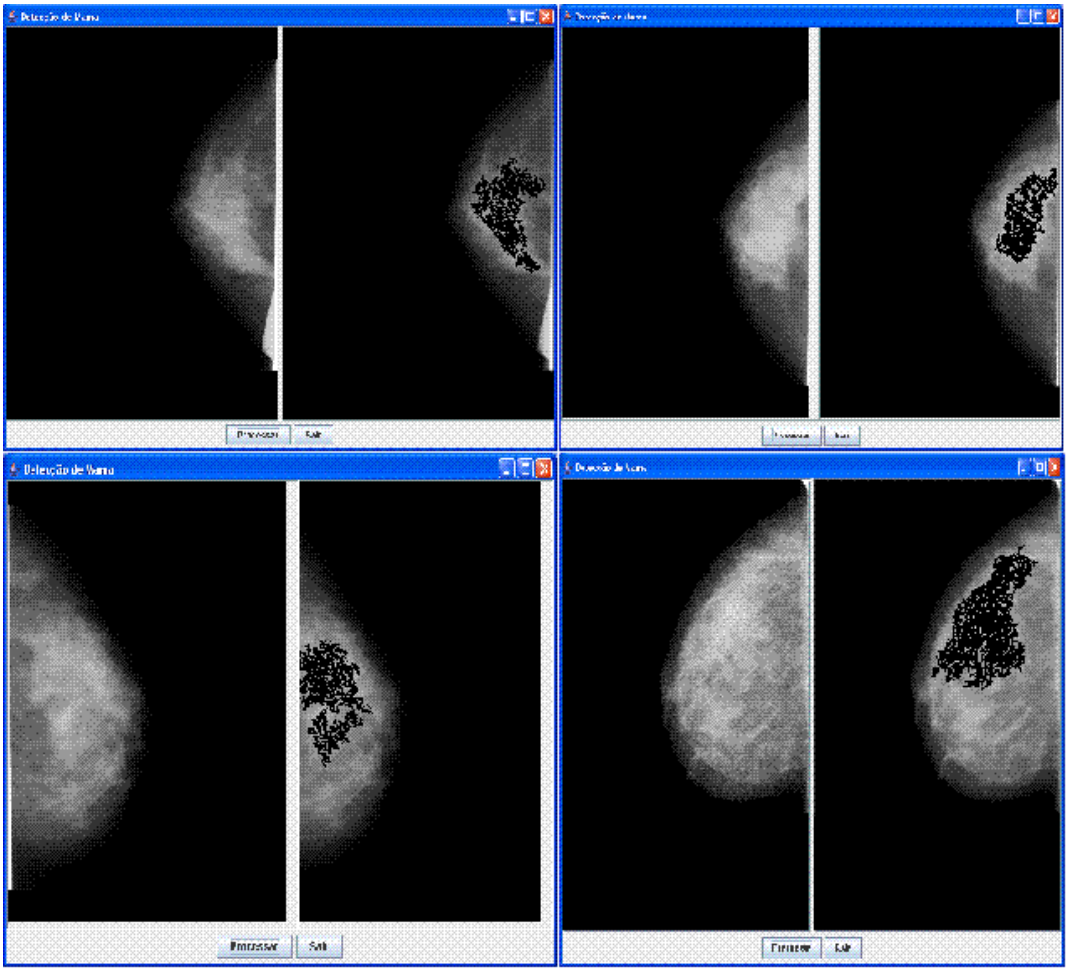


Figura 43 – Exemplo de imagens processados com um limiar alto

Quando os valores que correspondem ao tamanho do quadrante a ser analisado são elevados, o ponto “semente” que é utilizado pelo método de Crescimento de Região não estava localizado em uma região que continha pixels com valores elevados, gerando assim, uma região com poucos ou somente o pixel semente, assim como mostra a Figura 44.

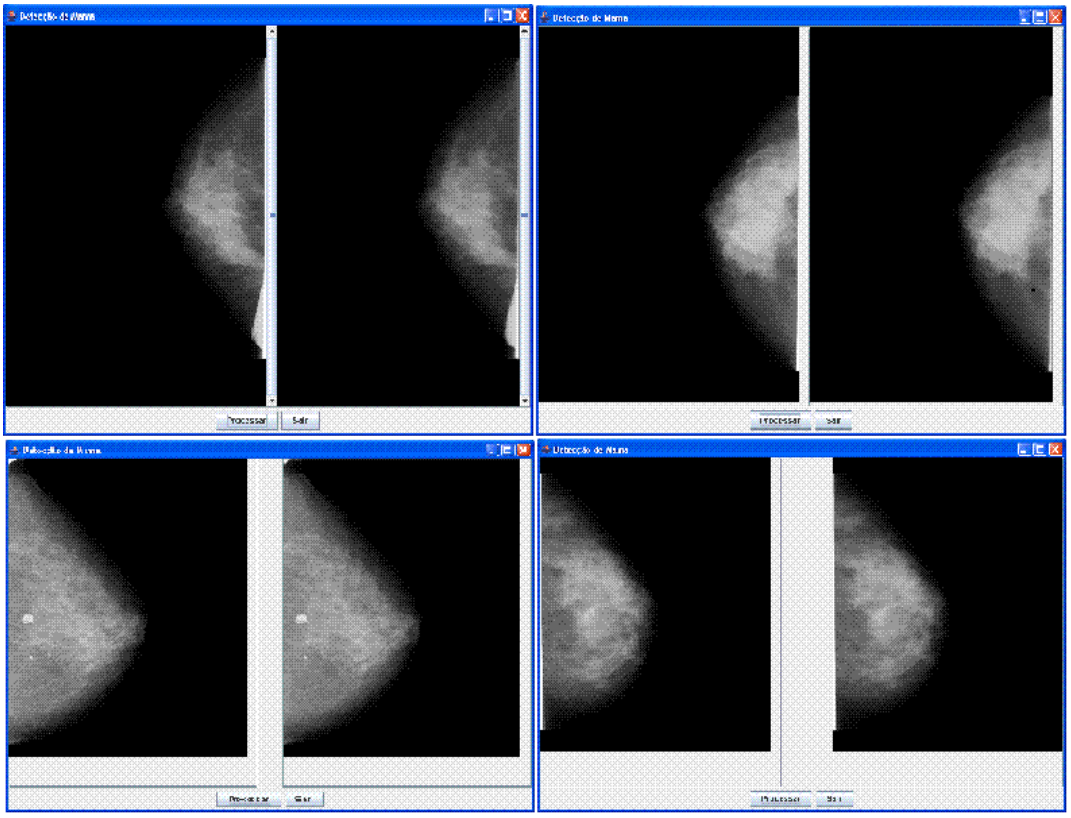


Figura 44 – Exemplo de imagens processadas com um valor de quadrante alto

Comparando a imagem processada com uma imagem que contém a demarcação feita por um radiologista destacando o nódulo, foi verificado que a escolha do ponto “semente” utilizando somente o nível de cinza retorna uma área que, em grande parte dos testes realizados, não é a que contém o nódulo.

A utilização de outras características para a determinação do ponto “semente” ou a seleção de vários pontos “sementes” podem ajudar a determinar áreas que sejam de interesse clínico

CAPÍTULO 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 do sistema para detecção de nódulos em mamogramas digitalizados , sendo capaz de efetuar as operações necessárias para a detecção da região na mamografia. A implementação teve como ponto inicial dar ao usuário todas as informações necessárias para que ele soubesse exatamente o que está acontecendo com a imagem processada.

Este trabalho pode evoluir principalmente na questão da definição dos parâmetros iniciais como o limiar utilizado e o tamanho do quadrante a ser analisado. Um modo de melhorar este ponto seria implementar algum método para avaliar automaticamente os parâmetros que devem ser usados na imagem, assim que esta for escolhida.

Outro ponto que se pretende explorar é a análise da região detectada com a aplicação de técnicas para a classificação da área encontrada.

REFERÊNCIAS BIBLIOGRÁFICAS

BALLARD, D.H; BROWN, C.M. Computer Vision. Englewood Cliffs, New Jersey, Prentice-Hall Inc., 1982.

BRAGA, A.C.S Curvas ROC: Aspectos Funcionais e Aplicações. Tese (Doutorado em Engenharia de Produção e Sistemas) – Universidade do Minho, Braga, 2000.

GONZALEZ, R.C.; WOODS, R.E. Processamento de Imagens Digitais. Ed Edgard Blücher LTDA. São Paulo, 2002.

NUNES, F. L. S. Investigações em processamento de imagens mamográficas para auxílio ao diagnóstico de mamas densas. Tese (Doutorado). Instituto de Física de São Carlos, 2001.

RUSS, J.C. The Image Processing Handbook. 2. ed. Boca Raton; CRC Press, 1995.

NISHIKAWA, R. M., GIGER, M. L.; VYBORNY, C. J. ‘‘Computer-aided detection and diagnosis of breast cancer,’’ Radiologic Clinics of North America 38, 725–740, 2000.

SANTOS, V. T. Segmentação de imagens mamográficas para a detecção de nódulos em mamas densas. Tese Mestrado, Escola de Engenharia de São Carlos, 2002

SAMPAT, M. P. MARKEY, M. K. BOVIK, A. C. “Computer-aided detection and diagnosis in mammography”, in Handbook of Image and Video Processing (ed. Bovik), 2nd edition 2005, p.1195-1217.

BELLOTI, R. *et al*. "A completely automated CAD system for mass detection in a large mammographyc database" Medical Physics, vol. 33, p.3066-3075 2006

ZWIGGELAAR R, PARR T.C., SCHUMM J.E., HUTT I.W., TAYLOR C.J., ASTLEY S.M., BOGGI C.R. Model-based detection of spiculated lesions in mammograms. Medical Image Analysis Volume 3, Issue 1, March 1999, p. 39-62

TWEED, T.;MIGUET, S. Automatic detection of regions of interest in mammographies based on a combined analysis of texture and histogram Pattern Recognition, 2002. Proceedings. 16th International Conference on Publication Date: 2002 Volume: 2, p: 448- 452 vol.2

S. SINGH, R. AL-MANSOORI, Identification of regions of interest in digital mammograms, *Journal of Intelligent Systems*, vol. 10, no. 2, p. 183-217, 2000

POEL, J. V. D. ; BATISTA, L. V. ; PIRES, G. M. ; NASCIMENTO, Tiago Dias Carvalho do . Análise de Imagens de Tumores de Câncer de Mama Baseada Em

Conteúdo: Método da Full Curvature Scale Space. In: X Congresso Brasileiro de Informática em Saúde, 2006, Florianópolis. Anais do X Congresso Brasileiro de Informática em Saúde, 2006. p. 16-21.

GULIATO, D. ; ADORNO, F. ; GUIMARAES, M. M. R. . Análise e Classificação de Tumores de Mama usando Métodos de Segmentação baseados em Conjuntos Difusos. Uberlândia: Pró-reitoria de Pesquisa e Pós-Graduação, http://www.propp.ufu.br/revistaeletronica/index.html, 2004

SCHALKOFF, R. J. Digital Image Processing and Computer Vision. John Wiley & Sons, Inc, Singapura, 1989.

Toroidal Gaussian Filters for Detection and Extraction of Properties of Spiculated Masses M.P. Sampat A.C. Bovik M.K. Markey G.J. Whitman T.W. Stephens. In: Proceedings. 2006 IEEE International Conference on Acoustics, Speech and Signal Processing, 2006, Volume: 2, p.: II-II

PECCINI, G.; D'ORNELLAS, M. Segmentação de Imagens por Watersheds: Uma Implementação Utilizando a Linguagem Java, In: Revista de Iniciação Científica, ano V, número IV, 2005.

SAMPAT, M.P.; BOVIK, A.C. Detection of spiculated lesions in mammograms. In: Proceedings of the 25th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, 2003, Volume: 1, p. 810- 813 Vol.1

SANTOS, R. D. C. **Java Advanced Imaging API: A Tutorial**. Revista de Informática Teórica e Aplicada, Rio Grande do Sul, v. 11, p. 93-123, 2004.

SUN Microsystems, JAI (Java Advanced Imaging) Application Programming Interface document home page. Disponível em: http://java.sun.com/products/javamedia/jai/forDevelopers/jai-apidocs/index.html. Acesso em Junho, 2007a.

SUN Microsystems, JAVA document home page. Disponível em: http://java.sun.com/j2se/1.5.0/docs/api/. Acesso em Junho, 2007b.

APENDICE A – CODIGO FONTE DA CLASSE PRINCIPAL.JAVA

```
import java.awt.BorderLayout;
import java.awt.Container;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;

import com.sun.media.jai.widget.DisplayJAI;

public class Principal extends JFrame {

  /**
   *
   */
  private static final long serialVersionUID = 1L;
  private Container contentPane;
  private Image     image;
  private static Principal principal;

  public Principal(){
      super("Detecção de Mama");
       this.image = new Image("sfundo.tiff");
       inid();
  }

  public void inid(){
      contentPane = getContentPane();
      contentPane.setLayout(new BorderLayout ());
      JButton abrir = new JButton("Processar");
      JButton sair  = new JButton("Sair");
      JPanel painel = new JPanel();
      painel.add(abrir);
      painel.add(sair);
      contentPane.add(painel,BorderLayout.SOUTH);
      contentPane.add(new JScrollPane(new DisplayJAI(
        image.escalaImagem(image.getImagem()))),BorderLayout.WEST);
      ButtonHandler handler = new ButtonHandler();
      sair.addActionListener  (handler);
      abrir.addActionListener (handler);
      setSize(800,600);
      setVisible(true);
  }

  private class ButtonHandler implements ActionListener {
      //trata evento de botão
      public void actionPerformed( ActionEvent event)
        {
            if (event.getActionCommand() == "Processar"){
              Nodulo nd = new Nodulo(image);
              contentPane.add(new JScrollPane(new DisplayJAI(
                image.escalaImagem(nd.CrescimentoRegiao()))),
                   BorderLayout.EAST);
            }
            if (event.getActionCommand() == "Sair")
```

```
                System.exit(1);
            }
    }

    public static void main (String args[]){

        String nome = JOptionPane.showInputDialog("Nome da
Imagem")+".TIF";
        new TiraFundo(nome);
        principal = new Principal();
   }
}
```

APENDICE B – CODIGO FONTE DA CLASSE IMAGE.JAVA

```
import java.awt.image.*;
import java.awt.image.renderable.*;
import javax.media.jai.*;

public class Image {

  private PlanarImage imagem;
  private int         altura;
  private int          largura;
  private int                nbands;
  private SampleModel sampleModel;
  private ColorModel  colorModel;

  public Image(){}

  public Image(String nomeImagem){
      PlanarImage image = JAI.create("fileload",nomeImagem);
      this.imagem       = image;
      this.altura       = image.getHeight();
      this.largura      = image.getWidth ();
      this.sampleModel  = image.getSampleModel ();
      this.nbands       = sampleModel.getNumBands();
      this.colorModel   = image.getColorModel();
  }

  public PlanarImage getImagem(){
    return this.imagem;
  }

  public int getAltura(){
    return this.altura;
  }

  public int getLargura(){
      return this.largura;
  }

  public int getNbans(){
      return this.nbands;
  }

  public int[] getPixels(){
      Raster inputRaster = imagem.getData();
      int[] pixels = new int[nbands*altura*largura];
      inputRaster.getPixels(0,0,largura,altura,pixels);
      return pixels;
  }

  public void salvaImagem(int[] pixels, String nomeImagem){
      Raster inputRaster = imagem.getData();
      WritableRaster outputRaster =
inputRaster.createCompatibleWritableRaster();
      TiledImage tiledimage = new TiledImage(
            0,0,largura,altura,0,0,sampleModel,colorModel);
      outputRaster.setPixels (0,0,largura,altura,pixels);
      tiledimage.setData(outputRaster);
      JAI.create("filestore",tiledimage,nomeImagem,"TIFF");
  }
```

```
  public PlanarImage escalaImagem(PlanarImage image){
      ParameterBlock pb = new ParameterBlock();
        pb.addSource(image);
        pb.add(0.19F);
        pb.add(0.19F);
        pb.add(0.0F);
        pb.add(0.0F);
        PlanarImage image2 = JAI.create("scale",pb,null);
      return image2;
  }
}
```

APENDICE C – CODIGO FONTE DA CLASSE NODULO.JAVA

```
import java.util.Vector;
import javax.media.jai.PlanarImage;
import javax.swing.JOptionPane;

public class Nodulo {

  private Image image;
  private int[] pixels;
  private int limiar;

  public Nodulo(){}

  public Nodulo(Image imagem){
      this.image = imagem;
      this.pixels = image.getPixels();
  }

  public float mediaGlobal(){
      int sgeral=0;
      for (int w = 0;w < pixels.length;w++)
        sgeral += pixels[w];

      return (sgeral/(image.getLargura()*image.getAltura()));
  }

  public int[] FiltroMedia(){

      int altura   = image.getAltura();
      int largura  = image.getLargura();
      int nbands   = image.getNbans();
      int posicao = 0;
      int posicao2 = 0;
      int soma = 0;
      for (int h=0;h < altura;h++){
        for(int w = 0;w < largura; w++){
            soma = 0;
            posicao = (h*largura*nbands+w*nbands);
            for(int hh =h - 3 ;hh <= h + 3 ;hh++)
            {
              for(int ww = w - 3; ww <= w + 3;ww++)
              {
                  posicao2 = (hh*largura*nbands+ww*nbands);
                  if(posicao2 < 0)
                  {
                    posicao2 = 0;
                  }// fim IF
                  if(posicao2 >= altura*largura)
                  {
                    posicao2 = altura*largura-1;
                  }// fim IF
                soma = soma + pixels[posicao2];
              }
              pixels[posicao] = soma/49;
            }
        }
      }
      return pixels;
  }
```

```
  public int MediaQuadrante(){

      int altura   = image.getAltura();
      int largura  = image.getLargura();
      int nbands   = image.getNbans();
      int inicioQuadrante=0;
      FiltroMedia();
      String mask = JOptionPane.showInputDialog(null,"Digite o valor
do quadrante");
      @SuppressWarnings("unused")
      int limiar = Integer.parseInt(
        JOptionPane.showInputDialog("Digite o valor do limiar"));
      int linhasQuadrante = Integer.parseInt(mask);
      int colunasQuadrate = Integer.parseInt(mask);
      float maiorMedia = mediaGlobal();

      for(int linhaGlobal = 0; linhaGlobal<altura;
        linhaGlobal     +=linhasQuadrante)
        for(int colunaGlobal = 0; colunaGlobal<largura;
          colunaGlobal +=colunasQuadrate)
        {
            int   offset =
            (linhaGlobal*largura*nbands+colunaGlobal*nbands);
            int   somaVizinhanca = 0;

            for(int linhaVizinhanca = linhaGlobal;linhaVizinhanca<
              linhaGlobal+linhasQuadrante;
              linhaVizinhanca++)
              if(linhaVizinhanca < altura)
                  for(int colunaVizinhanca = colunaGlobal;
colunaVizinhanca<colunaGlobal+colunasQuadrate;colunaVizinhanca++)
                  {
                    if(colunaVizinhanca < largura)
                    {
                        int offset2 =
(linhaVizinhanca*largura*nbands+colunaVizinhanca*nbands);
                        somaVizinhanca = somaVizinhanca +
pixels[offset2];
                    }
                  }

                  float media = somaVizinhanca/(linhasQuadrante *
colunasQuadrate);
                  if(media > maiorMedia){
                  maiorMedia = media;
                    inicioQuadrante = offset;
                }
        }
        //Localiza o ponto central de cada quadrante selecionado.

        int sementel = (inicioQuadrante/(largura*nbands)) +
linhasQuadrante/2;
        int sementec = (inicioQuadrante-
((inicioQuadrante/(largura*nbands))*largura*nbands))+
linhasQuadrante/2;
        return  ((sementel*largura*nbands)+(sementec*nbands));
  }

  public PlanarImage CrescimentoRegiao(){
```

```
   int nbands = image.getNbans();
   byte[] vetorAux = new
byte[image.getAltura()*image.getLargura()*image.getNbans()];

       for(int i=0;i<vetorAux.length;i++){
         vetorAux[i]= 1;
       }

   int indice = 0;

      Vector listaSemente = new Vector();
      Vector listaVeriSemente = new Vector();
      listaVeriSemente.add(MediaQuadrante());

      do{
          int semente = (Integer) listaVeriSemente.elementAt(0);
            while(vetorAux[semente] == 0){
              listaVeriSemente.remove(0);
                  if(listaVeriSemente.isEmpty())
                    break;
                  semente = (Integer) listaVeriSemente.elementAt(0);
            }

            int altura = (semente/(image.getLargura()*nbands));
            int largura = (semente-
((semente/(image.getLargura()*nbands))*image.getLargura()*nbands));
            int[] listaCandidatosSemente = new int[9];
            int aux=0;
            boolean quadranteValido = false;

            for (int i = (altura - 1);i <=(altura + 1);i++ )
              for (int j = (largura - 1);j <=(largura + 1);j++)
                  {
                    int offset =
(i*image.getLargura()*nbands+j*nbands);
                        if(semente != offset)
                        {
                          double limiarvari = Math.abs(pixels[semente]
- pixels[offset]);
                              if(limiarvari < limiar)
                              {
                                if(vetorAux[offset] != 0)
                                    {
                                      quadranteValido = true;
                                    }
                              }
                        }
                  }//end for

            vetorAux[semente] = 0;

            if(quadranteValido == true){
              for (int i = (altura - 1);i <=(altura + 1);i++ )
                  for (int j = (largura - 1);j <=(largura + 1);j++)
                    {
                        int offset =
(i*image.getLargura()*nbands+j*nbands);
                        if(semente != offset){
                          listaCandidatosSemente[aux]=offset;
                          listaSemente.add(offset);
                          aux++;
```

```
                    }
                  }
                listaSemente.add(semente);

                for(int t = 0 ;t < aux; t++){
                  if(vetorAux[listaCandidatosSemente[t]] != 0){

      listaVeriSemente.add(listaCandidatosSemente[t]);
                  }
                }//end for
            }//end if

            if(listaVeriSemente.isEmpty())
              break;

            listaVeriSemente.remove(0);
            indice++;
      }while(listaVeriSemente.size() != 0);//end while

      int[] pixels2 = pixels;
        for(int i=0;i<listaSemente.size();i++){
            int posicao = (Integer) listaSemente.elementAt(i);
            pixels2[posicao]= 0;
        }
        image.salvaImagem(pixels2,"resultado.tif");
        Image imagemresultado = new Image("resultado.tif");
        return imagemresultado.getImagem();
    }
}
```

APENDICE D - CÓDIGO FONTE DA CLASSE TIRAFUNDO.JAVA

```
import java.awt.image.ColorModel;
import java.awt.image.Raster;
import java.awt.image.SampleModel;
import java.awt.image.WritableRaster;

import javax.media.jai.TiledImage;
import javax.swing.JFrame;

public class TiraFundo extends JFrame
{

	/**
	 *
	 */
	private static final long serialVersionUID = 1L;

	public TiraFundo(final String nomeImage) {
		final Image mamograma = new Image(nomeImage);
		final int width = mamograma.getLargura();
		final int height = mamograma.getAltura();
		final int nbands = mamograma.getNbans();
		int[] pixels = new int[nbands*width*height];
		pixels = mamograma.getPixels();

		int offset, flagFim = 0, thr = 62300;
		long somaPri=0, valorPri=0, somaSeg=0, valorSeg=0,
		somaPen=0, valorPen=0, somaUlt=0, valorUlt=0,
		PixelInicial = 0, PixelFinal = 0, xCentro, yCentro;
		// Guarda o valor de um terço da imagem.
		final int h1_3 = (int) height/3;
		final long hist[] = new long[65536];

/**
* Trecho que faz o histograma e que calcula a média dos
* valores dos pixels das colunas iniciais (10ª e 11ª) e das
* finais correspondentes da imagem.
*/

		for (int h=0; h<height; h++)
			for (int w=0; w<width; w++) {

				offset = h*width*nbands+w*nbands;
				for (int band=0; band<nbands; band++) {
					if (w == 10)
					{
						somaPri = somaPri + 1;
						valorPri = valorPri +
pixels[offset+band];
					}
					else if (w == 11)
					{
						somaSeg = somaSeg + 1;
						valorSeg = valorSeg +
pixels[offset+band];
					}
					else if (w == width-12)
					{
						somaPen = somaPen + 1;
```

```
                                    valorPen = valorPen +
pixels[offset+band];
                              }
                              else if (w == width-11)
                              {
                                    somaUlt = somaUlt + 1;
                                    valorUlt = valorUlt +
pixels[offset+band];
                              };
                              // histograma
                              hist[pixels[offset+band]]++;
                        }
                  }

/**
* Trecho que encontra o valor do threshold (maior valor do
* histograma)
*/

            for (int i = 0; i < 65536; i++)
                  if (hist[i] > hist[thr])
                  {
                        thr = i;
                  }

// Trecho que faz o thresholding (limiarização) da imagem

            for (int h=0; h<height; h++)
                  for (int w=0; w<width; w++)
                  {
                        offset = h*width*nbands+w*nbands;
                              for (int band=0; band<nbands; band++)
                                    if ( pixels[offset+band] > thr )
                                          pixels[offset+band] = 65535;
                                    else
                                          pixels[offset+band] = 0;
                  }
/**
* Calcula a diferença das médias de valor de cinza das
* primeiras e das últimas colunas para determinar o lado da
* mama.
*/

            final long dif1 = Math.max((long)valorPri/somaPri,
(long)valorSeg/somaSeg)
                  - Math.min((long)valorPri/somaPri,
(long)valorSeg/somaSeg);
            final long dif2 = Math.max((long)valorPen/somaPen,
(long)valorUlt/somaUlt)
                  - Math.min((long)valorPen/somaPen,
(long)valorUlt/somaUlt);

/**
* Se as últimas colunas tem uma maior diferença, então a mama
* esta do lado direito e deve-se encontrar o ponto central a
* partir da direita.
*/

            if (dif2 > dif1)
            {
```

```
/**
* percorre a última coluna procurando o centro da mama,
* caso não encontre, vai para a penúltima e assim por
* diante, até que seja encontrado pelo menos um pixel
* preto.
*/

            int w = width;
            do {
                        w = w -1;
                        for (int h=0; h<height; h++)
                        {
                  offset = h*width*nbands+w*nbands;
                  for (int band=0; band<nbands; band++)
                        if (pixels[offset+band] == 0)
                      {
                              flagFim = 1;
                              if ( h < h1_3 )
                              {
                                    flagFim = 0;
                                    PixelInicial = h;
                              }
                              else
                                    PixelFinal = h;
                        }
            }
      } while (flagFim == 0);
      xCentro = w;
      yCentro = ( (PixelFinal - PixelInicial) /2 ) + PixelInicial;
}

/*
* Caso as primeiras colunas tenham uma maior diferença, então
* a mama está do lado esquerdo e deve-se procurar o ponto
* central a partir da esquerda.
*/

      else
      {
      //percorre a primeira coluna procurando o centro da mama
            int w = 0;
            do {
                  w = w + 1;
                  for (int h=0; h<height; h++)
                  {
                        offset = h*width*nbands+w*nbands;
                        for (int band=0; band<nbands; band++)
                              if (pixels[offset+band] == 0)
                              {
                                    flagFim = 1;
                                    if ( h < h1_3 )
                                    {
                                          flagFim = 0;
                                          PixelInicial = h;
                                    }
                                    else
                                          PixelFinal = h;
                              }
                  }
            } while (flagFim == 0);
            xCentro = w;
```

```
            yCentro = ( (PixelFinal - PixelInicial) /2 ) +
PixelInicial;
      }

/*
* Lançamento de raios a partir do ponto central definido acima
* a fim de encontrar a borda da mama.
*/

      final long vetorX[] = new long[362];
      final long vetorY[] = new long[362];
      int x, y;
      double Angulo, AnguloRadiano, Xaux, Yaux;
      int posVetor = 0;

      long menorY = height, maiorY = 0;
// Se a mama esta do lado direito

      if (dif2 > dif1) {
// Vai indo de 0.5 em 0.5 graus, de 90° a 270°

      for ( Angulo = 90.0; Angulo < 271; Angulo += 0.5 )
      {
            AnguloRadiano = Math.toRadians(Angulo);
            flagFim = 0; // ainda não chegou ao fim
            x = (int)xCentro;
            y = (int)yCentro;
            Xaux = (double)xCentro;
            Yaux = (double)yCentro;
            while ((flagFim == 0) && ( x < width ) && ( x > 0 ) && ( y
> PixelInicial) && ( y < PixelFinal ))
            {
                  Xaux = Xaux + Math.cos(AnguloRadiano);
                  x = (int)Xaux;
                  Yaux = Yaux - Math.sin(AnguloRadiano);
                  y = (int)Yaux;
                  offset = y*width*nbands+x*nbands;
                  for (int band=0; band<nbands; band++)
                        if (pixels[offset+band] == 0)
                              flagFim = 1;
                  }
      vetorX[posVetor] = x;
vetorY[posVetor] = y;
if (y > maiorY)
maiorY = y;
if ( y < menorY)
menorY = y;
posVetor++;
Xaux = x;
Yaux = y;
}
}
// Se a mama está do lado esquerdo
else {
// Vai indo de 0.5 em 0.5 graus, de 90° a 270°
for ( Angulo = 90.0; Angulo > -89.0; Angulo -= 0.5 ) {
if ( Angulo < 0 )
AnguloRadiano = Math.toRadians(360+Angulo);
else
AnguloRadiano = Math.toRadians(Angulo);
flagFim = 0; // ainda não chegou ao fim
```

```
x = (int)xCentro;
y = (int)yCentro;
Xaux = (double)xCentro;
Yaux = (double)yCentro;
while ((flagFim == 0) && ( x < width ) && ( x > 0 ) &&
( y > PixelInicial) && ( y < PixelFinal )) {
Xaux = Xaux + Math.cos(AnguloRadiano);
x = (int)Xaux;
Yaux = Yaux - Math.sin(AnguloRadiano);
y = (int)Yaux;
offset = y*width*nbands+x*nbands;
for (int band=0; band<nbands; band++)
if (pixels[offset+band] == 0)
flagFim = 1;
}
vetorX[posVetor] = x;
vetorY[posVetor] = y;
if (y > maiorY)

maiorY = y;
if ( y < menorY)
menorY = y;
posVetor++;
Xaux = x;
Yaux = y;
}
}
int flagSobre = 0;
long posMuda = height;
int h;
/**
* Encontrando e traçando linhas dos pontos encontrados
* e dos intemediários para delimitar a borda da mama.
*/
for (int j=0; j<posVetor-1; j++){
final int vetYj = (int)vetorY[j];
int vetYj1 = (int)vetorY[j+1];
/**
* tirando a média dos dois pontos para ver se estão a
* uma distância maior que 5% do tamanho da imagem, pois
* se estiverem, é um sinal que o ponto encontrado pode
* não pertencer a mama, uma vez que está muito distante
* do anterior (distante da borda)
*/
final int dist = vetYj > vetYj1 ? vetYj - vetYj1 : vetYj1 - vetYj;
if (dist > height*0.05) {
vetorY[j+1] = vetYj > vetYj1 ? vetorY[j+1]+(dist/2) :
vetorY[j+1]-(dist/2);
if (vetYj1 == maiorY)
maiorY = (int)vetorY[j+1];
else if(vetYj1 == menorY)
menorY = (int)vetorY[j+1];
vetYj1 = (int)vetorY[j+1];
}
/**
* Este trecho analisa se o ponto encontrado está
* sobrepondo ou não uma outra parte já encontrada, e se
* estiver é feito o tratamento necessário
* para não se perder nenhuma parte da imagem.
*/
if ((flagSobre == 0) && (vetYj > vetYj1)) {
```

```
flagSobre = 1;
posMuda = vetYj;
}
else if ((flagSobre == 1) && (vetYj < vetYj1))
flagSobre = 2;
else if ((flagSobre == 2) && (vetYj > vetYj1))
flagSobre = 1;
else if ((flagSobre == 2) && (vetYj > posMuda))
flagSobre = 0;
// Tratamento se os pontos entiverem em ordem crescente
for (int auxY = vetYj; auxY < vetYj1+1; auxY++) {
int auxX;
if (vetorX[j] == vetorX[j+1])
auxX = (int)vetorX[j];
else {
final double m = (double)(vetorY[j+1] - vetorY[j]) /

(vetorX[j+1] - vetorX[j]);
final double b = vetorY[j] - m*vetorX[j];
auxX = ( auxY - (int)b );
auxX /= m;
}
h = auxY;
for (int w=0; w<width; w++) {
offset = h*width*nbands+w*nbands;
for (int band=0; band<nbands; band++) {
// se mama está do lado direito
if (dif2 > dif1)
if (flagSobre == 0)
if (w < auxX)
pixels[offset+band] = 0;
else
pixels[offset+band] = 65535;
else {// se flag == 2
if (w >= auxX)
pixels[offset+band] = 65535;
if ((auxY > posMuda) && (w < auxX))
pixels[offset+band] = 0;
}
else // mama está do lado esquerdo
if (flagSobre == 0)
if (w > auxX)
pixels[offset+band] = 0;
else
pixels[offset+band] = 65535;
else {// se flag == 2
if (w <= auxX)
pixels[offset+band] = 65535;
if ((auxY > posMuda) && (w > auxX))
pixels[offset+band] = 0;
}
}
}
}

// Tratamento se os pontos entiverem em ordem decrescente
      for (int auxY = vetYj; auxY > vetYj1-1; auxY--)
      {
            int auxX;
            if (vetorX[j] == vetorX[j+1])
                  auxX = (int)vetorX[j];
```

```
            else {
                  final double m = (double)(vetorY[j+1] - vetorY[j])
/(vetorX[j+1] - vetorX[j]);
                  final double b = vetorY[j] - m*vetorX[j];
                  auxX = ( auxY - (int)b );
                  auxX /= m;
      }

      h = auxY;
      for (int w=0; w<width; w++)
      {
            offset = h*width*nbands+w*nbands;
            for (int band=0; band<nbands; band++)
                  // se mama está do lado direito
                  if (dif2 > dif1)
                  {
                        if (w >= auxX)
                              pixels[offset+band] = 0;
                  }
                  else
                        if (w <= auxX)
                              pixels[offset+band] = 0;
      }
      }
}

/*
* Todas as linhas que estiverem abaixo do menorY encontrado e
* que estiveram acima do maiorY encontrado ficam inteiras
* pretas.
*/

      for (h=0; h<height; h++)
            for (int w=0; w<width; w++)
            {
                  offset = h*width*nbands+w*nbands;
                  for (int band=0; band<nbands; band++)
                        if ((h < menorY) || (h > maiorY))
                              pixels[offset+band] = 0;
            }
/**
* Criação das variáveis necessárias para duas novas imagens,
* um que será a imagem original, e outra a imagem resultante
* (sem o fundo)
*/
      final SampleModel sm1 = mamograma.getImagem().getSampleModel();
      final int nbands1 = sm1.getNumBands();
      final Raster inputRaster1 = mamograma.getImagem().getData();
      final WritableRaster outputRaster1 =
inputRaster1.createCompatibleWritableRaster();
      final int[] pixels1 = new int[nbands1*width*height];
      final ColorModel cm1 = mamograma.getImagem().getColorModel();
      final TiledImage original = new
TiledImage(0,0,width,height,0,0,sm1,cm1);
      inputRaster1.getPixels(0,0,width,height,pixels1);
      final SampleModel sm2 = mamograma.getImagem().getSampleModel();
      final int nbands2 = sm2.getNumBands();
      final Raster inputRaster2 = mamograma.getImagem().getData();
      final WritableRaster outputRaster2 =
      inputRaster2.createCompatibleWritableRaster();
      final int[] pixels2 = new int[nbands2*width*height];
```

```
      int offset2;

/**
* Neste trecho é avaliado se o pixel na imagem que foi
* limiarizada e que está com o fundo preto e a mama branca é
* preto, se for, a imagem resultante recebe 0 (preto), caso
* contrário, a mesma recebe o valor da imagem original.
*/

      for (h=0; h<height; h++)
            for (int w=0; w<width; w++)
            {
                  offset = h*width*nbands+w*nbands;
                  offset2 = h*width*nbands2+w*nbands2;
                  for (int band=0; band<nbands; band++)
                        if (pixels[offset+band] == 0)
                              pixels2[offset2+band] = 0;
                        else
                              pixels2[offset2+band] =
pixels1[offset+band];
            }
      mamograma.salvaImagem(pixels2,"sfundo.tiff");

// Grava a imagem resultante com o nome sfundo.tiff
      }
}
```