André Fernando Lourenço de Souza

# *Abordagens para a Segmentação de Coronárias em Ecocardiografia*

André Fernando Lourenço de Souza

Abordagens para a Segmentação de Coronárias em Ecocardiografia

v.1
2010

São Paulo

2010

**André Fernando Lourenço de Souza**

# *Abordagens para a Segmentação de Coronárias em Ecocardiografia*

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo (EPUSP) para a obtenção do título de Mestre em Engenharia de Eletricidade no Programa de Pós-Graduação em Engenharia Elétrica.

v.1

São Paulo
2010

**André Fernando Lourenço de Souza**

# *Abordagens para a Segmentação de Coronárias em Ecocardiografia*

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo (EPUSP) para a obtenção do título de Mestre em Engenharia de Eletricidade no Programa de Pós-Graduação em Engenharia Elétrica.

Área de Concentração:
Engenharia Biomédica

Orientador:
Prof. Dr. Sérgio Shiguemi Furuie

v.1

São Paulo
2010

**Este exemplar foi revisado e alterado em relação à versão original, sob responsabilidade única do autor e com a anuência de seu orientador.**

**São Paulo, 03 de setembro de 2010.**

**Souza, André Fernando Lourenço de**
**Abordagens para a segmentação de coronárias em ecocardiografia / A.F.L. de Souza. -- São Paulo, 2010.**
**p.**

**Dissertação (Mestrado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Telecomunicações e Controle.**

**1. Processamento digital de imagens 2. Ultrassonografia 3. Fuzzy (Inteligência artificial) I. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Telecomunicações e Controle II. t.**

*“Mesmo desacreditado e ignorado por todos, não posso desistir,*
*pois para mim, vencer é nunca desistir”*
Albert Einstein

*Para minha mãe (in memoriam) e meu pai por todo o esforço*
*para minha educação, e minha família, Michela e Júnior,*
*por todo suporte e alegria.*

## AGRADECIMENTOS

Agradeço primeiramente a meu orientador, Sérgio Shiguemi Furuie, que sempre foi atencioso, paciente e confiante no meu desempenho, mesmo eu não sendo um bolsista e não tendo dedicação integral ao mestrado.

Também agradeço à minha família que sempre me suporta em todos os sentidos.

Agradeço ao Prof. Dr. Marco Túlio Andrade que me orientou no início do mestrado e me aceitou no programa.

Sou muito grato aos professores Dr. Hae Yong Kim e Dr. Marco Antonio Gutierrez que, gentilmente, aceitaram o convite para a minha banca do exame de qualificação, questionando pontos importantes para o meu desenvolvimento.

Um obrigado, também, para meus amigos Fábio Rodrigo Amaral e Eduardo Luís de Araújo por terem me ajudado na conclusão dos créditos.

Além disso, eu agradeço ao Danilo Lage, do Incor, pela colaboração em explicações e códigos de extensões do ImageJ. E ao Fernando Cardoso do LEB por me fornecer o código do Matlab para a geração de ruído *speckle* nas imagens simuladas.

E também do LEB, agradeço a Mônica Matsumoto e ao Matheus Moraes pelas discussões de temas no laboratório.

## RESUMO

A Ecocardiografia continua sendo a técnica de captura de imagens mais promissora, não-invasiva, sem radiação ionizante e de baixo custo para avaliação de condições cardíacas. Porém, é afetada consideravelmente por ruídos do tipo *speckle,* que são difíceis de serem filtrados. Por isso fez-se necessário fazer a escolha certa entre filtragem e segmentador para a obtenção de resultados melhores na segmentação de estruturas. O objetivo dessa pesquisa foi estudar essa combinação entre filtro e segmentador. Para isso, foi desenvolvido um sistema segmentador, a fim de sistematizar essa avaliação. Foram implementados dois filtros para atenuar o efeito do ruído *speckle* - *Linear Scaling Mean Variance* (LSMV) e o filtro de Chitwong - testados em imagens simuladas. Foram simuladas 60 imagens com 300 por 300 *pixels*, 3 modelos, 4 espessuras e 5 níveis de contrastes diferentes, todas com ruído *speckle.* Além disso, foram feitos testes com a combinação de filtros. Logo após, foi implementado um algoritmo de conectividade *Fuzzy* para fazer a segmentação e um sistema avaliador, seguindo os critérios descritos por Loizou**,** que faz a contagem de verdadeiro-positivos (VP) e falso-positivos (FP). Foi verificado que o filtro LSMV é a melhor opção para segmentação por conectividade *Fuzzy*. Foram obtidas taxas de VP e FP na ordem de 95% e 5%, respectivamente, e acurácia em torno de 95%. Para imagens ruidosas com alto contraste, aplicando a segmentação sem filtragem, a acurácia obtida foi na ordem de 60%.



**Palavras-chave:** processamento digital de imagens, segmentação de imagens, ecocardiografia, *fuzzy-connectedness*.

## ABSTRACT

The echocardiography is the imaging technique that remains most promising, noninvasive, no ionizing radiation and inexpensive to assess heart conditions. On the other hand, is considerably affected by noises, such as speckle, that are very difficult to be filtered. That is why it is necessary to make the right choice of filter and segmentation method to obtain the best results on image segmentation. The goal was evaluate this filter and segmentation method combination. For that, it was developed a segmentation system, to help the assessment. Two filters were implemented to mitigate the effect of speckle noise – Linear Scaling Mean Variance (LSMV) and the filter presented by Chitwong - to be tested in simulated images. We simulated 60 images, with size 300 by 300 pixels, 3 models, 4 thicknesses and 5 different levels of contrast, all with speckle noise. In addition, tests were made with a combination of filters. Furthermore, it was implemented a Fuzzy Connectedness algorithm and an evaluation system, following the criteria described by Loizou, which makes the true positives (TP) and false positives (FP) counting. It was found that the LSMV filter is the best option for Fuzzy Connectedness. We obtained rates of TP and FP of 95% and 5% using LSMV, and accuracy of 95%. Using high contrast noisy images, without filtering, we obtained the accuracy in order of 60%.

**Keywords:** digital image processing, image segmentation, echocardiography, fuzzy-connectedness.

## LISTA DE ABREVIATURAS E PALAVRAS RESERVADAS

**2D –** 2 dimensões ou bidimensional - uma imagem plana.

**3D –** 3 dimensões ou tridimensional - uma imagem composta.

**API –** *Application Programming Interface* – Interface de Programação de Aplicativos.

***Batch* –** Em lote.

**FC** - *Fuzzy Connectedness* - Conectividade *Fuzzy* ou Nebulosa.

**FP –** Falso-Positivo.

***Fuzzy* –** Antônimo de “Crisp”. Significa: vago, subjetivo, impreciso, incerto.

**GFC –** *Generic Fuzzy Connectedness,* conectividade nebulosa genérica.

**IFT –** *Image Foresting Transform* - transformada imagem-floresta.

**JNI -** *Java Native Interface* - Interface entre códigos Java e códigos nativos.

**LSMV** – *Linear Scaling Mean Variance* - variância media do escalonamento linear.

***Phantom* –** Fantoma, imagem simulada.

***Pixel* –** *Picture element* - elemento de imagem.

***Plugin* –** Extensão de funcionalidades de um programa.

***Region Growing* –** crescimento de regiões – método de segmentação simples.

**RG -** *Region Growing* – Crescimento de regiões.

**RMSE** - *Normalized Root Mean Square error* - raíz normalizada do erro médio quadrático.

***Seed* –** Semente, ponto inicial de na imagem para inicio da segmentação.

***Speckle* –** forma de ruído granular, encontrado em imagens de ultrassom.

***Spel* –** *Spacial element* - elemento espacial, pode ser um *píxel*, ou um *voxel*, 3D.

**SO –** Sistema Operacional.

**UML –** *Unified Modeling Language* – Linguagem Unificada de Modelagem.

***Voxel* –** *Volume element* - elemento de volume.

**VP –** Verdadeiro-positivo

**XOR –** *Exclusive-Or* – Ou-Exclusivo – operação lógica.

## LISTA DE FIGURAS

## LISTA DE TABELAS

**SUMÁRIO**

# 1. INTRODUÇÃO

Neste capítulo é feita a contextualização da dissertação e a apresentação dos objetivos.

## 1.1. A Segmentação em Ecocardiografia

A Ecocardiografia continua sendo a técnica de captura de imagens mais promissora, não-invasiva, sem radiação ionizante e baixo custo para avaliação de condições cardíacas [1]. Por outro lado, é afetada consideravelmente por ruídos do tipo *speckle* que são muito difíceis de serem filtrados. Com as imagens ecocardiográficas, a segmentação das artérias coronárias (figura 1.1), do ventrículo esquerdo do coração, assim como de outras estruturas e câmaras, é muito importante para uma série de tarefas relacionadas à quantificação de condições cardíacas [1].

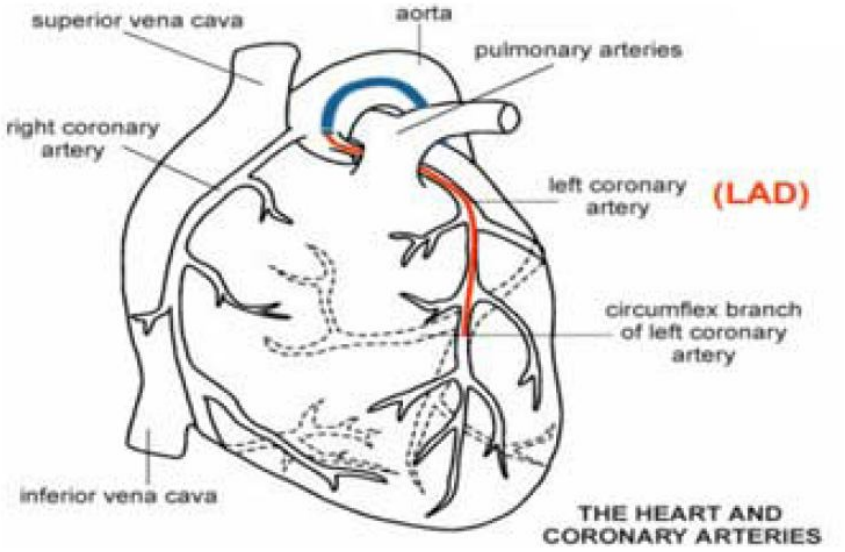



***Figura 1.1*** *– Anatomia do coração humano, destacando as coronárias [2].*

Recentemente, a ecocardiografia emerge como uma das mais prevalentes modalidades em imagens médicas na Cardiologia. Principalmente, com o advento do agente de contraste por microbolhas e da matriz de transdutores bidimensionais. Com isso possibilita-a a ser uma alternativa de baixo custo para a avaliação de estruturas do coração como as artérias coronárias e câmaras. Por outro lado, o número de ferramentas computacionais disponíveis para manipular as imagens cardiográficas, especialmente as 3D, são escassas e não tem processamento satisfatório [3]. A imagem 1.2 mostra o exemplo de uma imagem de ecocardiografia 2D (a) e 3D (b).

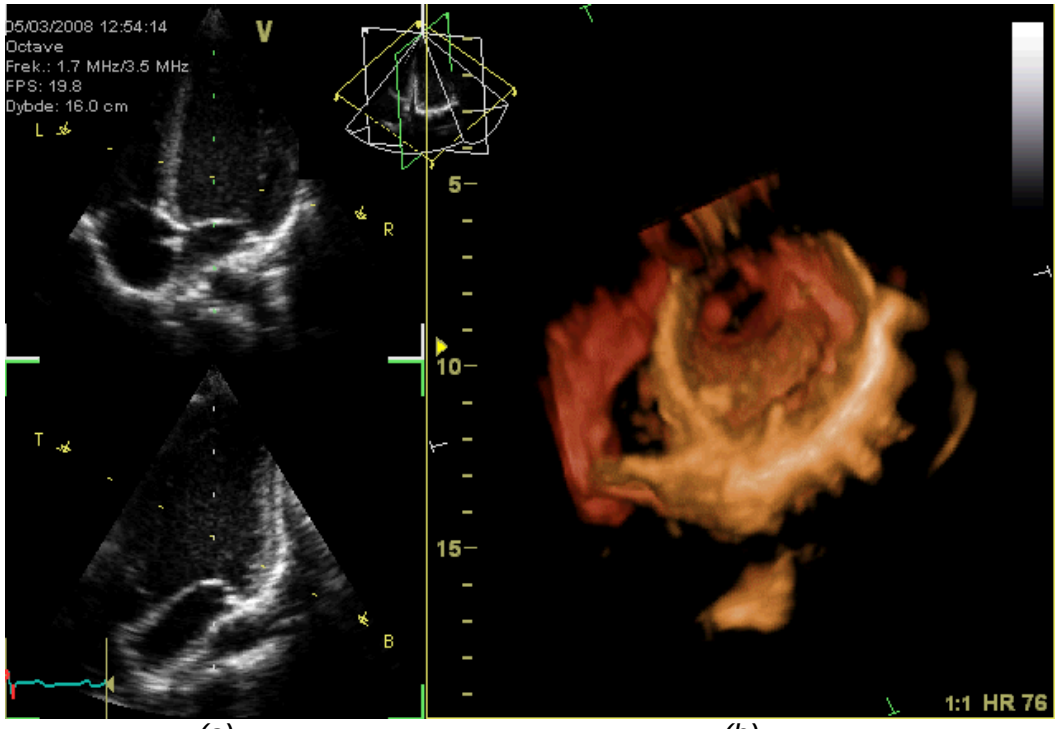
Medidas ecocardiográficas de tamanho e função do ventrículo esquerdo (VE) do coração humano são largamente usadas clinicamente em pacientes com problemas cardíacos no diagnóstico e no prognóstico [3].

A segmentação é largamente usada no processamento de imagens para a solução de tais avaliações. Ela é um problema complexo que relaciona o conhecimento de várias disciplinas e algoritmos. Ela é dividida em duas partes [4]: reconhecimento e delineamento. Reconhecimento é a definição de alguma estrutura em uma imagem, e delineamento é a definição do contorno da estrutura a ser identificada na imagem. Para o reconhecimento, os seres humanos possuem grande acurácia, mas não para o delineamento; nesse caso máquinas são melhores que os seres humanos especialmente quanto à precisão e desempenho.

A avaliação dos métodos de segmentação pode ser dividida em: acurácia (determinação exata do objeto), precisão e desempenho do sistema. A idéia é sempre maximizar esses três pontos [3]. Infelizmente, para a ecocardiografia, a segmentação de imagens é muito difícil por causa de intensidades não homogêneas, distorções e ruído *speckle*, provocando falhas na maioria dos métodos. Portanto, um passo de pré-processamento é necessário para melhorar a acurácia da segmentação [1].

Há vários métodos descritos na literatura: alguns muito simples, como a limiarização e crescimento de região; e outros complexos, como o *Fuzzy*

*Connectedness* (FC) [5]. Para simplificar os métodos, podemos aplicar uma transformada *Image Foresting Transform* (IFT) na imagem de entrada [6].




*(a)* *(b)*

***Figura 1.2** – Ecocardiografia 2D (a) e 3D (b) – Imagem real cedida pelo Instituto do Coração de São Paulo (INCOR).*

O IFT, descrito por [5], transforma uma imagem em um grafo orientado e ponderado. O algoritmo manipula esse grafo, usando uma função de custo, que é dependente da saída esperada e do método de segmentação.

A segmentação por FC é uma metodologia baseada em crescimento de região, na qual o delineamento do objeto é dependente de uma semente, colocada pelo usuário do sistema. Não é um processo totalmente automático. Em algumas propostas, é possível selecionar outra semente para indicar o fundo da imagem. Ambas as sementes são contribuições do usuário para o passo do reconhecimento. Então o resultado do algoritmo de FC é um objeto definido pelos elementos da imagem que apresentarem uma taxa de conectividade acima de um determinado limiar, em relação à semente imposta [US-03].

Normalmente o algoritmo de *Fuzzy Connectedness* apresenta como saída um mapa das conectividades em relação à semente (pode ser representado como uma outra imagem). A partir desse mapa, deverá ser aplicada alguma tomada de decisão, como por exemplo, uma limiarização para saber se um determinado *spacial element (spel)* faz parte do fundo ou do objeto a ser segmentado.

## 1.2. Objetivo da Pesquisa

A partir de imagens ecocardiográficas simuladas, filtrar e segmentar estruturas de interesse tais como artérias coronárias e avaliar a combinação entre o filtro e o método. E com esses dados, discutir quais são as melhores opções.

Além disso, apresentar um sistema em blocos de um segmentador para facilitar a avaliação de filtros e segmentadores.

Nessa pesquisa são implementados os filtros para atenuar o *speckle*, a serem testados em imagens simuladas, inclusive a combinação dos mesmos.

***Figura 1.3*** *– Exemplo de imagem 2D simulada e com ruído.*

Depois disso, é implementado um algoritmo de FC, capaz de segmentar estruturas em imagens simuladas 2D (figura 1.3).

Finalmente, com os dados obtidos, seguindo os critérios descritos por Loizou [7]**,** é feita uma análise para se concluir qual o melhor filtro para o melhor método.

## 1.3. Estrutura do Texto

Primeiramente é apresentado um referencial teórico abordando assuntos como estudo de filtros, segmentadores e métodos de avaliação.

Depois no capítulo de metodologia, é explicado como foi feita a pesquisa, implementação de códigos e avaliadores. Além disso, são discutidos os métodos escolhidos, aspectos técnicos e de arquitetura de sistemas. Há, ainda, no final um apêndice com a publicação de alguns códigos-fonte.

Logo após, no capítulo de resultados, há tabelas com os dados obtidos para cada filtro, dados estatísticos, imagens e parâmetros de entrada utilizados na geração das mesmas.

Finalmente é feita a discussão com conclusões, contribuições e pontos a serem melhorados no futuro.

# 2. REFERENCIAL TEÓRICO

O referencial teórico está dividido da seguinte maneira: primeiramente, são discutidos alguns aspectos sobre ruído *speckle*, sua origem, formulação e métodos de filtragem. Depois, dois filtros que tratam o *speckle*: *Linear Scaling Mean Variance* (LSMV) e de Chitwong, explicando também a sua formulação. Logo após, é discutida a segmentação de imagens, com o método *Fuzzy Connectedness* (FC). Finalmente é feita a discussão sobre a avaliação de imagens.

## 2.1. *Ruído Speckle e Filtragem*

O artigo do Loizou, *"Comparative Evaluation of Speckle Filtering in Ultrasound Imaging of the Carotid Artery"* [8]**,** é uma boa fonte de conhecimento sobre ruído s*peckle* e filtragem. O *speckle* é uma forma de ruído correlacionada localmente e multiplicativo que corrompe as imagens médicas de ultrassonografia. Assim como o artigo do Loizou, Noble publicou um artigo sobre ultrassom e segmentação *"Ultrasound Image Segmentation: A Survey"* [1], que diz que o s*peckle* possui uma forma granular e ele é inerente ao processo de ultrassonografia. Ele não é um ruído propriamente dito, porque num típico conceito de engenharia o *speckle* pode ter informação útil, devido a sua textura.

Em outras aplicações, o *speckle* pode ser muito útil como rastreamento de *speckle* [9], porém nessa aplicação, segmentação de estruturas, ele deve ser removido sem destruir características importantes na imagem. Porque do contrário, torna-se difícil à observação visual, especialmente para não especialistas, para interpretar e detectar baixo contraste e pequenas lesões. Além disso, ele diminui a acurácia e precisão da maioria dos métodos de segmentação, assim como outros tipos de ruído fazem [8].

Yongjian em seu artigo, "*Speckle Reducing Anisotropic Diffusion*" [10], faz uma descrição matemática de como o ruído *speckle* é formado, que foi importante na geração de imagens simuladas nesse estudo.

Loizou, em [8], faz uma comparação de vários tipos de filtro *speckle,* como por difusão anisotrópica, filtragem geométrica, *wavelets*, mediana etc. Todos esses filtros devem preservar bordas e características chave dos objetos a serem segmentados. Yongjian diz, em seu artigo [10], que filtros baseados em janelamento não são bons porque a janela é dependente de tamanho e forma, não são direcionais e não ressaltam as bordas, somente as preserva. Porém, Loizou provou que o melhor filtro é o baseado em estatísticas locais, no caso o *Linear Scaling Mean Variance* (LSMV), seguido pelos demais outros filtros baseado em janelamento.

A seguir, são descritos esse filtro e um outro proposto por Chitwong no artigo *"Segmentation on Edge Preserving Smoothing Image based on Graph Theory"* [11]. Primeiramente é descrita a formulação do ruído *speckle*.

### *2.1.1. Ruído speckle*

Segundo [YA-02] e [BD-80], define-se uma imagem simulada com ruído *speckle*, $V(x, y)$, como:

$$V(x, y) = h(x, y) * T(x, y) \tag{1}$$

Onde * é uma convolução espacial. E $h(x, y)$ é uma função de ponto de espalhamento ou resposta a impulso de um sistema de ultrassom hipotético:

$$h(x, y) = h_1(x) h_2(y) \tag{2}$$

Onde $h_1(x)$ é uma função senoidal ponderada por uma Gaussiana (função de *Gabor*):

$$h_1(x) = sen(k_o x)\exp[\frac{-x}{2\sigma_x^2}] \quad (3)$$

e

$$k_o = 2\pi \frac{f_0}{c} \quad (4)$$

Onde *c* é a velocidade do som no tecido humano, $f_0$ freqüência central e $\sigma_x$ é a largura de pulso axial da onda ultrassônica transmitida.

O outro termo, $h_2(y)$ é a resposta espacial para a transmissão e recepção determinada por:

$$h_2(y) = \exp[\frac{-y^2}{2\sigma_y^2}] \quad (5)$$

Onde $\sigma_Y$ é a largura lateral da onda ultrassônica transmitida.

$T(x,y)$ modela as não-homogeneidades da impedância acústica do tecido, ou objeto, devido a variações de densidade e velocidade, gerando dispersões:

$$T(x,y) = t(x,y).G(x,y) \quad (6)$$

Onde $t(x,y)$ é o modelo de ecogeneidade do objeto a ser visualizado, que varia se o *pixel* representa fundo ou artéria, e $G(x,y)$ é um ruído branco Gaussiano com média zero e variância definida.

A representação real da imagem de saída é:

$$V_\alpha(x,y) = V(x,y) + j\hat{V}(x,y) \quad (7)$$

Onde $\hat{V}(x,y)$ é a transformada de Hilbert de $V(x,y)$ com respeito à x. Como essa representação é um número complexo, para a implementação, aplica-se o módulo para obter os valores dos *pixels* na imagem simulada com ruído.

### *2.1.2. LSMV*

*Linear Scaling Mean Variance* (LSMV), descrito por Loizou [8], é um filtro baseado em janelas quadradas que coleta informações estatísticas locais de primeira e segunda ordem. Ele utiliza a média e a variância da vizinhança de um *pixel* (elemento de imagem).

De uma maneira simples, ele é a média ponderada usando estatísticas de sub-regiões (um quadrado deslizando pelos *spels* da imagem, como 3x3, 5x5 e 7x7 – para casos 2D).

A imagem de saída é formada por:

$$f_{i,j} = \bar{g} + k_{i,j}(g_{i,j} - \bar{g}) \tag{8}$$

Onde $f_{i,j}$ é o *pixel* de saída estimado em uma janela deslizante, $g_{i,j}$ é o *spel* de entrada ruidoso no centro da janela, $\bar{g}$ é o valor da media local de uma janela que contenha $g_{i,j}$, $k_{i,j}$ é um fator ponderante com $k_{i,j}$ entre [0…1], e *i* e *j* sendo as coordenadas do *spel*.

O fator $k_{i,j}$ é uma função de estatísticas locais:

$$k_{i,j} = \frac{\sigma^2 - \sigma_n^2}{\sigma^2} \tag{9}$$

Onde $\sigma^2$ é a variância dentro da janela e $\sigma_n^2$ é a variância do ruído em toda a imagem.

A variância do ruído deve ser computada em uma região homogênea e em uma janela consideravelmente maior que a janela de filtragem, usando, por exemplo, o fator característico do speckle ($\alpha$):

$$\alpha = \frac{1}{P}\sum_{p=1}^{P}\frac{\sigma_p^2}{\bar{g}_p} \quad (10)$$

Onde $\sigma_p^2$ e $\bar{g}_p$ são a variância e a média do ruído em uma janela selecionada de região homogênea, e p é um índice de janela em toda a imagem. A quantidade total de janelas utilizada é dada por P (que pode ser somente 1). Se $k_{i,j}$ é 1 (nas bordas) o *spel* permanece igual, e quando ele é 0 (em áreas uniformes) ele será trocado pela média local.

O ruído *Speckle* pode ser aproximado a uma distribuição de Rayleigh. Para resultados melhores, o filtro deve ser aplicado na mesma imagem de três a quatro vezes, iterativamente.

### *2.1.3. Chitwong*

No artigo [11], Chitwong descreve um filtro que tem como maior objetivo preservar as bordas das imagens, também chamado de processo de suavização e preservação de bordas.

Assim como o filtro LSMV, ele calcula a média e a variância em torno de um *spel*, mas ele não faz uso de janelas quadradas. Ele utiliza janelas com os formatos da figura 2.1. Elas são doze janelas, onde cada quadrado é um *spel*. O quadrado marcado com "*(x,y)"* é o *spel* "central" da janela, e é quem recebe o valor filtrado.

Para cada janela, *k*, calcula-se a variância (para medir a homogeneidade de cada janela) com respeito a *(x,y)*, o *spel* "central" da janela:

$$V(k) = \frac{1}{m}\sum_{i,j\in S_k}[f(i,j) - f(x,y)]^2; k = 1,2,...,N$$

(11)

Onde *f(i,j)* são os outros *spels* dentro da janela, *N* é o número de janelas (12) e *m* é o número de pixels que compõe a máscara.

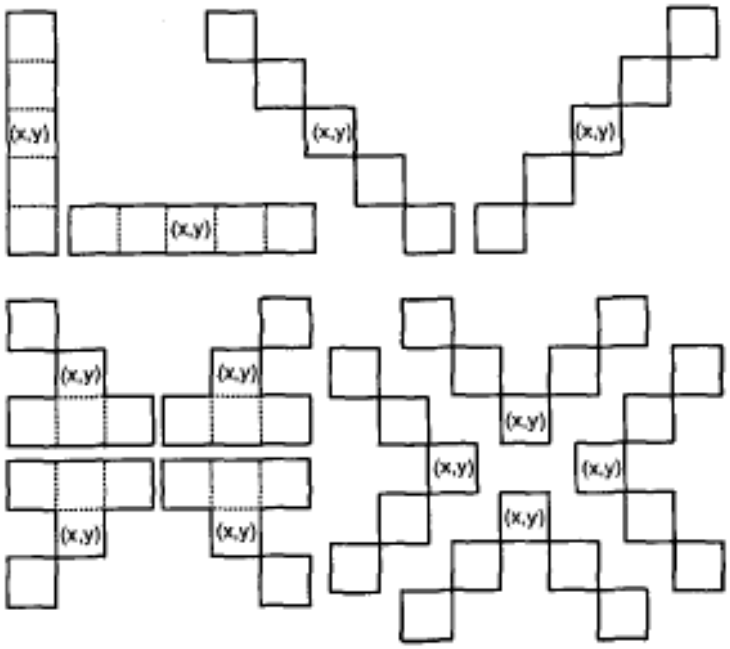



***Figura 2.1*** *– Formatos de janelas propostas por Chitwong [11]*

Então, selecione a janela com menor variância e troque o valor do *spel* pela média calculada para esta janela:

$$\bar{x}(k) = \frac{1}{m} \sum_{i, i \in S_k} f(i, j) \qquad (12)$$

Onde ***N*** é o número de janelas, *S,* é um subconjunto dos *spels* existentes em cada janela, *f (i,j)* é o valor do nível de cinza da coordenada *(i,j)* e *m* é o número de *spels* em cada janela.

Faça isso para todos os *spels* da imagem. Finalmente, repita o processo para toda a imagem até que os *spels* não alterem mais os seus valores.

Executar esse algoritmo é o mesmo que detectar a direção da borda. Se fosse usada uma janela quadrada, as bordas seriam degradadas ou o ruído perto das bordas não seria removido. Somente funcionaria se a imagem filtrada tivesse linhas ortogonais à janela. A figura 2.1 mostra o conjunto de janelas para o caso bidimensional.

## 2.2. Segmentação de Imagens

Segmentação de imagens é um método de processamento de imagens que tem como objetivo reconhecer e delinear objetos em uma imagem [7]. Ela pode ser feita usando uma variedade de métodos descritos na literatura como *Snakes* [12,13], *Level Set* [14], transformada *Watershed* [15,16], *Fuzzy Connectedness* [5,17,18], *IFT* [6,16], limiarização [19,20] etc. Porém, aqui é descrito somente o *Fuzzy Connectedness* (FC), que foi o método de segmentação escolhido.

### *2.2.1. Fuzzy Connectedness*

Udupa em seu artigo *"Fuzzy Connectedness and Image Segmentation"* [5], descreve o método *Fuzzy Connectedness* (FC) que é um método baseado na metodologia de crescimento de regiões. Este método requer a interação do usuário para selecionar uma ou mais “sementes” que indicam pontos de partida para que o algoritmo inicie a detecção de quais *spels* pertencem ao objeto (o objeto da segmentação) e quais pertencem ao fundo da imagem. Em algumas propostas, é possível selecionar uma semente para indicar o fundo da imagem. Ambas as sementes são contribuições do usuário no passo do reconhecimento. Então, o resultado do algoritmo de *Fuzzy Connectedness* é um objeto definido

por elementos da imagem que apresentam altos valores de conectividade com relação à semente.

Esse método cria um conjunto *Fuzzy*, que pode ser, por exemplo, dois elementos: um objeto na imagem a ser detectado (por exemplo, as artérias coronárias) e o fundo da imagem. Então, para cada *spel* um grau de pertinência é calculado (0 até 1). Se ele é zero, o *spel* não pertence ao objeto, e se ele é igual a 1, ele pertence. Com isso, a tomada de decisão deve ser feita para selecionar os *spels* que pertencem ao objeto e os que pertencem ao fundo da imagem. Para isso, pode ser aplicada uma simples limiarização.

FC é baseado em duas relações *Fuzzy*: uma relação *Fuzzy* local chamada afinidade que é o *"hanging togetherness"* de dois *spels*, baseado na proximidade no espaço e a intensidade. E uma relação *Fuzzy* global chamada conectividade de um conjunto de *spels* baseada no caminho mais forte (de maior peso) entre dois *spels*. A força de um caminho pode ser definida pela menor afinidade encontrada em um caminho. Fazendo uma analogia com uma corrente, onde a força da mesma é definida pela força do elo mais fraco.

Para definir a afinidade entre dois *spels*, *'c'* e *'d'*:

$$\mu_k(c,d) = \frac{1}{2}\mu_\psi(c,d) + \frac{1}{2}\mu_\phi(c,d) \tag{13}$$

Onde $\mu_\psi(c,d)$ é a componente da homogeneidade (equação 14) e $\mu_\phi(c,d)$ a componente da intensidade (equação 15):

$$\mu_\psi(c,d) = \exp\left[-\frac{1}{2}\left(\frac{|f(c)-f(d)|-m_1}{s_1}\right)^2\right] \tag{14}$$

$$\mu_\phi(c,d) = \exp\left[-\frac{1}{2}\left(\frac{\left(\frac{f(c)+f(d)}{2}\right)-m_2}{s_2}\right)^2\right] \tag{15}$$

Onde $m_1$ e $s_1$ são, respectivamente, a média e o desvio padrão de homogeneidades locais e, $m_2$ e $s_2$ a média e o desvio padrão das intensidades dos objetos.

A conectividade entre *'c'* e *'d'* é:

$$\mu_K(c,d) = \max_k(\min_{1<i<N_p}(\mu_k(s_{i-1}, s_i))) \quad (16)$$

Onde $N_p$ é o número de *spels* entre *'c'* e *'d'* e, $s_{i-1}$ e $s_i$ são os vizinhos para um determinado caminho k. Resumindo, é definido pelo caminho mais forte, e a força de um caminho pela afinidade mais fraca. Como numa corrente, a força é definida pelo elo mais fraco.

Após esse cálculo da conectividade, é aplicado o algoritmo de Dijkstra [21] para a otimização dos caminhos.

Finalmente, uma limiarização é aplicada à conectividade (que pode ser visualizada como uma imagem, onde a intensidade dos pixels é a conectividade), separando os *spels* pertencentes ao fundo da imagem dos que pertencem ao objeto. Pode ser um *thresholding* (limiarização) simples, ou um automático com o método de Otsu [19].

## 2.3. Avaliação

Loizou em seu outro artigo, *"Quality Evaluation of Ultrasound Imaging in the Carotid Artery based on normalization and Speckle Reduction Filtering"* [7], descreve critérios de avaliação de filtros e segmentadores.

Um de seus critérios, onde dois especialistas observam cada imagem segmentada e dão uma nota para elas (1-ruim a 5-excelente). Assim faz-se uma estatística colocando como parâmetro de entrada o filtro utilizado com um segmentador e como saída essas notas.

Em outro critério de avaliação eles irão segmentar as imagens manualmente para serem os padrões de segmentação, em caso de imagens reais. Com isso, pode-se calcular a porcentagem de falsos-positivos e de

verdadeiros-positivos detectados, fazendo uma combinação lógica *XOR* (Ou-Exclusivo) entre a imagem padrão (segmentada pelos especialistas ou a imagem original sem ruído, se for simulada) e a imagem de saída do segmentador.

# 3. METODOLOGIA

Nesse capítulo é apresentado o sistema de segmentação e detalhes e escolhas feitas para essa pesquisa.

## 3.1. Sistema de Segmentação

Nessa dissertação é apresentada um modelo de segmentação completo, que foi implementado para o estudo dos filtros e algoritmos de segmentação. O sistema é esquematizado na figura 3.1. A idéia é que com esse sistema criado, possa-se simplesmente substituir os blocos por outras técnicas, para assim poder fazer uma comparação com mais acurácia e robustez.

Como primeiro passo, foram geradas imagens 2D de tamanho 300 por 300 *pixels*, fantomas de artérias coronárias, (3 modelos de estruturas tubulares e finas) com e sem ruído, e com variações de contraste (5 níveis) e espessura (4 valores). Esses fantomas simularão variações em imagens reais. Estas serão as entradas do sistema de segmentação proposto. Nesse estudo não serão usadas imagens ecocardiográficas reais, tampouco 3D.

No passo de pré-processamento serão feitas três investigações: com o LSMV, Chitwong e ambos combinados (primeiro LSMV e depois o de Chitwong). O primeiro, descrito por Loizou [8], assume que o ruído *speckle* tem uma distribuição de Rayleigh e define uma metodologia similar ao filtro média ponderada para a redução do ruído.

O outro filtro usado, descrito por Chitwong et al [11], usa um conjunto de janelas com formatos não convencionais. Então, um novo valor de *spel* é definido como o valor médio definida pela janela de menor variância. Este filtro propõe destacar as bordas de estruturas tubulares, justamente o que essa dissertação busca para segmentação de artérias coronárias.

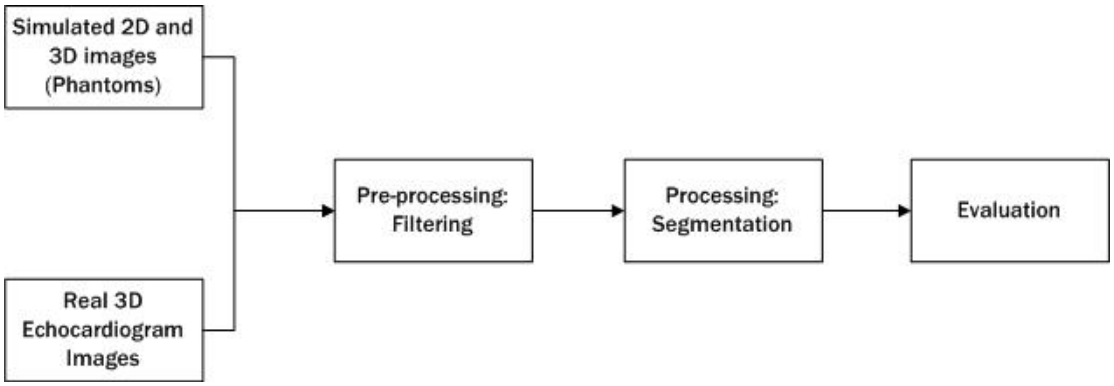



***Figura 3.1*** *– Diagrama em blocos do sistema de segmentação utilizado [23,24].*

O próximo passo será a segmentação em si. Em [16] é feita uma comparação dos métodos *Watershed* e *Fuzzy Connectednes* utilizando ou não o IFT. Lá, ele diz que o resultado é o mesmo utilizando tal abordagem. Por conta disso, e como o escopo dessa pesquisa é a comparação dos filtros, foi implementado somente o FC sem o IFT.

A avaliação será feita utilizando um dos critérios descritos por Loizou [22], que é a contagem de verdadeiro-positivos e falso-positivos.

## 3.2. Justificativa para a Conectividade *Fuzzy*

Nessa pesquisa foi escolhido o método de segmentação *Fuzzy Connectedness*. Pois, para a segmentação de imagens de ecocardiografia, não é possível utilizar métodos de segmentação simples, como limiarização [19] ou *region growing* [20], porque as imagens possuem ruído *speckle* significativo que causa borramentos e não-homogeneidades nas imagens.

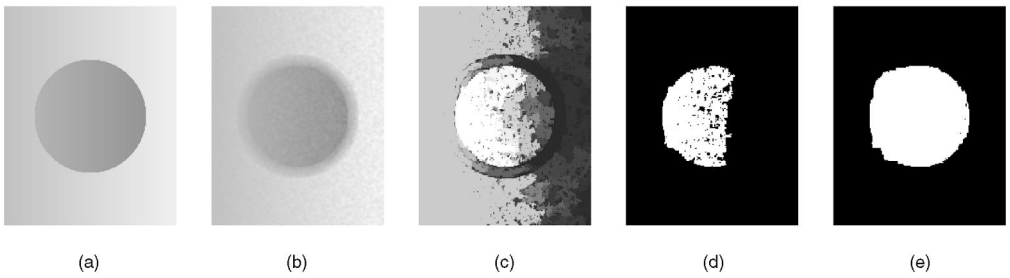



***Figura 3.2*** *– Exemplo de segmentação do círculo na imagem simulada para demonstrar a efetividade da Conectividade Fuzzy em relação à um método mais simples, como o region growing - (a) Imagem simulada original, (b) Imagem com ruído Gaussiano, (c) relação de afinidade de cada spel, (d) Círculo segmentado por crescimento de região (percebe-se que parte dele foi perdida), (e) Círculo segmentado por conectividade Fuzzy [5].*

O método *Fuzzy Connectedness* tem se mostrado eficiente para imagens de ecocardiografia [5] e tem sido muito utilizado para a segmentação. Mesmo comparado a outros métodos como o *snakes* [12,13], e dependendo da aplicação, ele é mais interessante, pois é um método robusto, principalmente quando se trata de ecocardiografia e imagens com intensidades não-uniformes. Além disso, ele exige pouca interação com o usuário, no caso somente o fornecimento de uma “semente”, para inicializar a estrutura a ser segmentada, e também requer o ajuste de poucos parâmetros comparado com o *snakes*.

A figura 3.2 mostra um exemplo de segmentação por *region growing* e por *Fuzzy Connectedness.* A figura (a) é a imagem original simulada e a (b) é a mesma com ruído gaussiano. Em (c) é mostrado o mapa de afinidades após a execução do *Fuzzy Connectedness*. Com esse mapa, foi feita uma limiarização, obtendo-se (e). E em (d) foi aplicado o *region growing,* que por não se basear em variações locais, para esse tipo de imagem ele tem baixa acurácia, diferentemente do *Fuzzy Connectedness,* como pode ser visto.

## 3.3. Implementação de Código

Todo o sistema foi implementado sob a plataforma *ImageJ* [25], que é de código aberto e desenvolvida em linguagem Java.

Os filtros serão implementados como extensões dele. Haverá uma extensão para a geração de fantomas, porém a geração de ruído *speckle* nas mesmas será feita por um programa no Matlab, por causa da disponibilidade da transformada de *Hilbert* nesta plataforma.

Também foi criada uma extensão para fazer a avaliação das imagens geradas com as processadas pelo sistema de segmentação.

### *3.3.1. ImageJ*

O *ImageJ* [25] é uma plataforma feita em linguagem Java para a visualização e processamento de imagens. Desenvolvida pelo "*National Institutes of Health*", agência americana responsável por pesquisas na área biomédica, localizada em Bethesda, Maryland. A figura 3.3 mostra como é a tela principal do ImageJ.

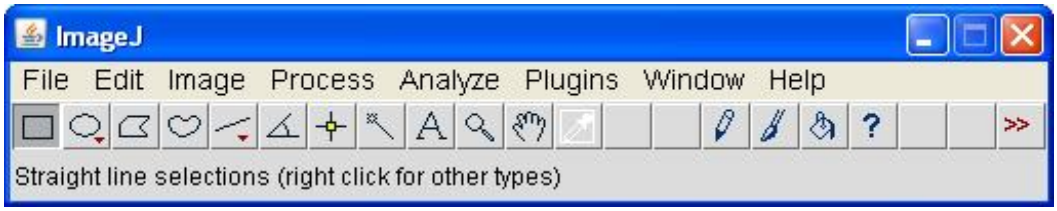



***Figura 3.3*** *– Tela principal do ImageJ [25]*

Essa plataforma foi escolhida, pois:

- É de domínio público, tendo seu código livre, facilita o desenvolvimento em conjunto e a melhoria contínua do sistema;
- É feito em Java, podendo ser desenvolvido e executado em qualquer S.O. (Windows, Linux, MacOS etc);
- Há uma grande comunidade inscrita na lista de discussão da página do Instituto;

- Podem ser criadas extensões para o código da plataforma, facilitando a distribuição de códigos;
- Manipula vários formatos de arquivos, (BMP, JPEG, GIF, TIFF, PNG, DICOM etc), e possui uma grande quantidade de funcionalidades prontas para o processamento, visualização de imagens e criação de interfaces gráficas;
- As imagens podem ser agrupadas como pilhas, importante para a manipulação de imagens 3D e reconstruções tomográficas;
- Permite a criação de "*macros*", que é uma forma simples de programar uma seqüência de ações, automatizando processos;
- Possui uma extensa documentação, facilitando o desenvolvimento (como diagrama de classes e exemplos).

## 3.4. Extensões do ImageJ

Para cada passo do sistema de segmentação proposto (figura 3.1), foi criada uma extensão do ImageJ. Depois de pronto cada passo, pode-se usar uma *macro* para executar todos os passos em seqüência.

Para cada extensão criada, é gerado um arquivo "jar", que deve ser copiado para o diretório "plugins" do ImageJ. Com isso, ao ser iniciado, a plataforma lê o diretório e lista as extensões disponíveis no menu "Plugins".

Para a criação de uma extensão, deve-se criar uma classe Java derivada da classe "PlugIn", "PlugInFilter" ou "PluInFrame". A "PlugIn" é tipo mais básico de extensão que ao ser executada, não requer uma imagem na entrada, com isso todas a verificações e criação de interface gráfica devem ser programadas. A "PlugInFrame" é a "PlugIn" associada a uma janela e a "PlugInFilter" requer uma imagem como entrada e faz essa verificação automaticamente [25]. Para os três casos, é necessário criar um método "*run",* que é chamado quando a extensão é executada (como pode ser visto no exemplo do apêndice B). Todo o código do processamento esta dentro de "*run*".

Para essa pesquisa foram criadas estas extensões, todas derivadas da classe “PlugIn”:

- Gerador de imagens – para a geração de imagens simuladas;
- Filtro *LSMV* – aplica filtro LSMV;
- Filtro *Chitwong* – aplica filtro de Chitwong;
- Segmentador *Fuzzy Connectedness* – para segmentar uma imagem;
- Avaliador de imagens – avalia imagem ou diretórios (imagens padrões e segmentadas).

Somente o filtro LSMV não foi implementado para ser executado em modo em lote, não permitindo a automatização do sistema, pois para ele é necessário selecionar uma janela com ruído constante e isso é muito particular de cada imagem de entrada.

### *3.4.1. Gerador de Imagens*

A partir de uma imagem padrão binária (nível 0 para o fundo e nível 255 para o objeto), ou um diretório de imagens (modo “*batch”)*, a extensão gera imagens com contrastes diferentes entre fundo e objeto, e aplica dilatações na imagem para que a artéria simulada fique mais espessa, como pode ser vista na figura 3.4, que apresenta alguns exemplos.

*(a)* *(b)* *(c)* *(d)* *(e)*

***Figura 3.4*** *– Exemplos de imagens simuladas. Imagem (a): imagem padrão binária (entrada) e a extensão gerou as demais (b, c, d, e) com “Object gray level” = 128, “# of contrast samples” = 2, “# of diameter samples” = 2.*

Para a primeira imagem da esquerda (padrão) foi utilizado os parâmetros "*Object gray level*" = 128, indicando que o objeto terá nível 128 (somente o nível de fundo irá variar para gerar as amostras com contrastes diferentes), "*# of contrast samples*" = 2, que significa que dois contrastes diferentes serão gerados e "*# of diameter samples*" = 2, que diz que serão aplicadas duas dilatações na imagem padrão.

A opção "*Batch Process"* indica se será usada como entrada uma imagem ou um diretório de imagens.

Ao executar essa extensão, abre-se uma janela como a da figura 3.5.

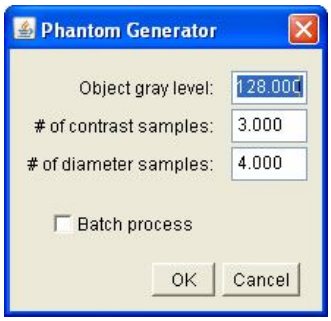



***Figura 3.5*** *– Tela da extensão "Gerador de Imagens".*

Se optar por "*Batch process*", a extensão irá abrir uma tela para indicar o usuário diretório de entrada. Para ambos os casos, ele irá pedir também que seja indicado o diretório das imagens geradas.

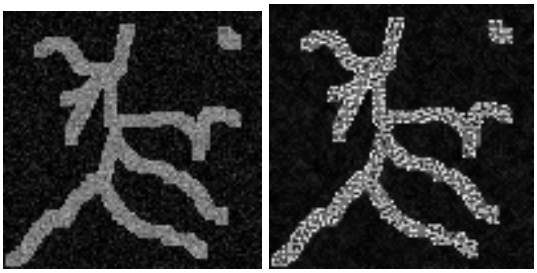
### *3.4.2. Filtro Chitwong*

Ao executar essa extensão, abre-se uma janela como a mostrada na figura 3.6.


***Figura 3.6*** *– Tela da extensão "Filtro Chitwong".*

A figura 3.7 mostra uma imagem com ruído e a mesma imagem filtrada. Nesse exemplo pode ser visto que as bordas foram destacadas, comparando-se com a imagem de entrada.

Há as opções 2D e 3D, porém só está implementada a opção 2D.



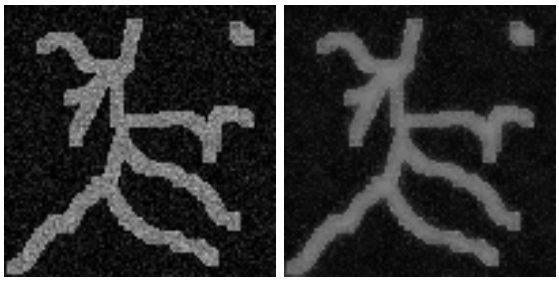
*(a)* *(b)*

***Figura 3.7*** *– Exemplos de imagem com ruído speckle e imagem filtrada por Chitwong.*

E assim como o gerador de imagens, há uma opção de "*Batch process*" podendo ser filtrada somente a imagem aberta ou um diretório inteiro.

### *3.4.3. Filtro LSMV*

Para esta extensão não há a interface gráfica, tampouco opção "*batch*", pois é necessária a seleção de uma área de ruído homogêneo com a ferramenta de seleção retangular (figura 3.3, primeiro botão da esquerda). Portanto, deve-se abrir a imagem, selecionar uma área de ruído homogêneo (não selecionar uma borda entre fundo e objeto) e então executar a extensão.



*(a)* *(b)*

***Figura 3.8** – Exemplos de imagem com ruído speckle (a) e imagem filtrada por LSMV (b).*

A figura 3.8 mostra o mesmo exemplo da figura 3.8 e a mesma filtrada pelo LSMV.

### *3.4.4. Segmentador Fuzzy Connectedness*

Essa extensão faz a segmentação de uma imagem utilizando a conectividade *Fuzzy* genérica. E assim, como o gerador de imagens, há uma opção de "*Batch process"* podendo ser filtrada somente a imagem aberta ou um diretório inteiro. Ao executar essa extensão, abre-se uma janela como a da figura 3.9.

Quando é selecionada a opção "*batch*", deve-se definir a posição da semente (*seed*) para todas as imagens a serem segmentadas, ou seja, todas as imagens do diretório de entrada, vão usar a mesma posição para a semente. Essa definição é feita nas opções "*Def. seed X*" (para a posição X da semente) e "*Def. seed Y*" (para a posição Y).

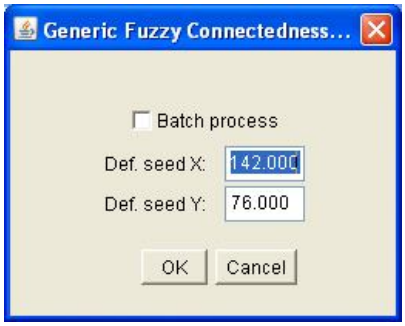


***Figura 3.9*** *– Tela da extensão "Segmentador Fuzzy Connectedness Genérico".*

Quando não é utilizada essa opção, deve-se abrir a imagem pelo ImageJ, e selecionar um ponto antes da execução da extensão (ferramenta "*Point selections*", sétimo botão da figura 3.3). Após a escolha da semente, abrirá uma janela pedindo o valor do limiar para a binarização do mapa de afinidades.

*(a)* *(b)*

***Figura 3.10*** *– Exemplos de imagem de entrada com ruído speckle (a) e imagem de saída segmentada por GFC e* limiarizada (b)*.*

A figura 3.10 mostra exemplos de uma imagem com ruído e a mesma imagem segmentada. A imagem mostrada já é o mapa de afinidades limiarizado, (usando o limiar configurado) por isso observa-se uma imagem binarizada (somente contém *pixels* com nível 0 para fundo ou com nível 255 para objeto).

### *3.4.5. Avaliador de imagens*

Essa extensão faz a contagem e cálculo da porcentagem de falso-positivos e verdadeiro-positivos, comparando a imagem gerada, sem ruído, com a imagem segmentada. Ela não possui interface gráfica e só tem opção "*batch*". Ela somente

pede ao usuário que especifique um diretório com as imagens padrão e o diretório das imagens segmentadas.

Ela faz a comparação, seguindo um critério no nome dos arquivos de ambos diretórios (esse critério é gerado pelas outras extensões quando selecionado o modo "*batch*").

O padrão do nome do arquivo é:

- <segmentada><filtro><id><contraste><largura>

Onde:

<segmentada>

- "s_" ->se foi segmentada ou não;

<filtro>

- "l_" -> imagem filtrada pelo LSMV;
- "c_" -> imagem filtrada pelo Chitwong;
- "c_l_" -> imagem filtrada pelo LSMV e pelo Chitwong;

<id>

- Número de dois dígitos (índice para identificar a imagem);

<contraste>

- "I_" – nível 1 de contraste (maior possível);
- "II_" – nível 2 de contraste e assim sucessivamente.

<largura>

- Largura da "artéria" simulada: quanto maior, indica que mais vezes foi aplicada a dilatação na imagem padrão.

Por exemplo, o nome "s_l_n_01_II_3.bmp" significa:

s_ -> imagem segmentada (as imagens padrões não possuem);

l_ -> imagem filtrada pelo LSMV;

01_ -> identificador 01;

II_ -> nível 2 de contraste;

03 -> 3 *pixels* de largura da "artéria" simulada.

Outro exemplo, do seu padrão correspondente, “01_II_3.bmp” significa:

“” -> imagem não segmentada;

“” -> imagem não filtrada;

01_ -> identificador 01;

II_ -> nível 2 de contraste;

03 -> 3 *pixels* de largura da “artéria” simulada.

A extensão então verifica os identificadores para comparar as imagens corretas (com o mesmo identificador).

A comparação é feita com base na imagem padrão, que só possui dois níveis de *pixel:* um menor que identifica o fundo e um maior que idêntica o objeto. Primeiramente o código faz essa identificação na imagem padrão (quais *pixels* são fundo e quais são objeto). Depois, observando a imagem segmentada (que é binarizada: 0 identifica fundo e 255 o objeto), aplica-se XOR entre as imagens, para fazer a contagem de falso-positivos e verdadeiro-positivos.

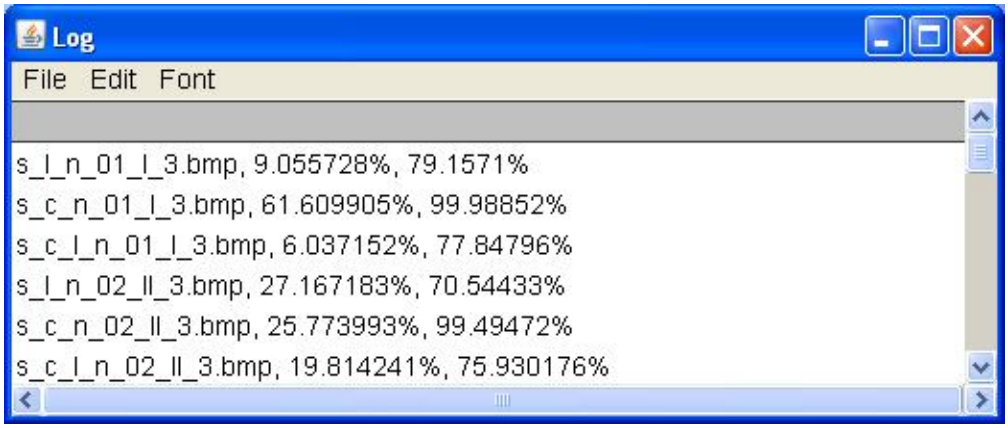



***Figura 3.11*** *– Caso hipotético de dados de saída do avaliador: nome do arquivo, porcentagem de falso-positivos e porcentagem de verdadeiro-positivos.*

O código então gera como saída essas informações, como o exemplo da figura 3.11, que devem ser copiados manualmente para uma tabela, ou planilha, para analise.

Essa saída mostra o nome da imagem segmentada, a porcentagem de falso-positivos e de verdadeiro-positivos.

## 3.5. Gerador de Ruído Speckle no Matlab

A geração de ruído *speckle* é feita por um programa em Matlab, que é uma plataforma comercial. Ele utiliza como entrada as imagens criadas pela extensão "*Phantom Generator*".

Do ponto de vista de arquitetura de sistemas, o ideal seria se esse programa fosse implementado como uma extensão, assim como todo o sistema, porém foi utilizado o Matlab, que possui uma funções prontas para calculo de convoluções, utilizadas para a geração do ruído.

O ruído *speckle* foi modelado matematicamente conforme a explicação do artigo de Yongjian, "*Speckle Reducing Anisotropic Diffusion*" [10], descrito anteriormente nesta pesquisa. Os parâmetros de entrada utilizados foram: c=1540 (velocidade do som no tecido), f=50E6 (frequência do ultrassom = 50MHz), alfa=1.2023 (constante de atenuação do sangue 0,18dB/MHz/cm(1,0423) ou 0,8dB/Mhz/cm(1,2023)) e desvio padrão de 0.3 para a parte Gaussiana do ru.

# 4. RESULTADOS OBTIDOS

Este capítulo descreve o experimento, como parâmetros de entrada para a formação das imagens e os resultados obtidos.

Estão demonstradas tabelas e gráficos obtidos com a segmentação por *Fuzzy Connectedness*, utilizando os filtros Gaussiano, LSMV, Chitwong e ambos combinados (primeiro LSMV e depois Chitwong). Além disso, há os dados dos testes *t-student*, comparando os tipos de filtragem.

## 4.1. Formação das Imagens Simuladas

Para a geração das imagens simuladas, de tamanho 300 por 300 *pixels,* foi utilizado a extensão do *ImageJ* descrita anteriormente, com parâmetros *"Object gray level"* = 250, *"# of contrast samples"* = 5 (5 níveis de contraste) e*"# of diameter samples"* = 4 (4 espessuras diferentes) em 3 amostras de entrada (nomeadas 'A', 'B' e 'C'), gerando 20 por amostra, totalizando 60 imagens geradas. Essas imagens sem ruído são guardadas para o uso da extensão avaliadora de imagens.

A figura 4.1 mostra os padrões 'A', 'B' e 'C' (as três primeiras imagens, ao topo) e as demais foram geradas a partir do padrão 'A', com 4 variações de contraste e espessura.

As imagens com nível 1 de ruído, são as que possuem menor contraste (pequena diferença entre o nível de cinza do objeto e o nível do fundo) e as de nível 5 possuem fundo = 0 (contraste máximo). O nível do objeto é sempre o mesmo (definido pelo parâmetro "*Object gray level*"), variando-se somente o nível do fundo.

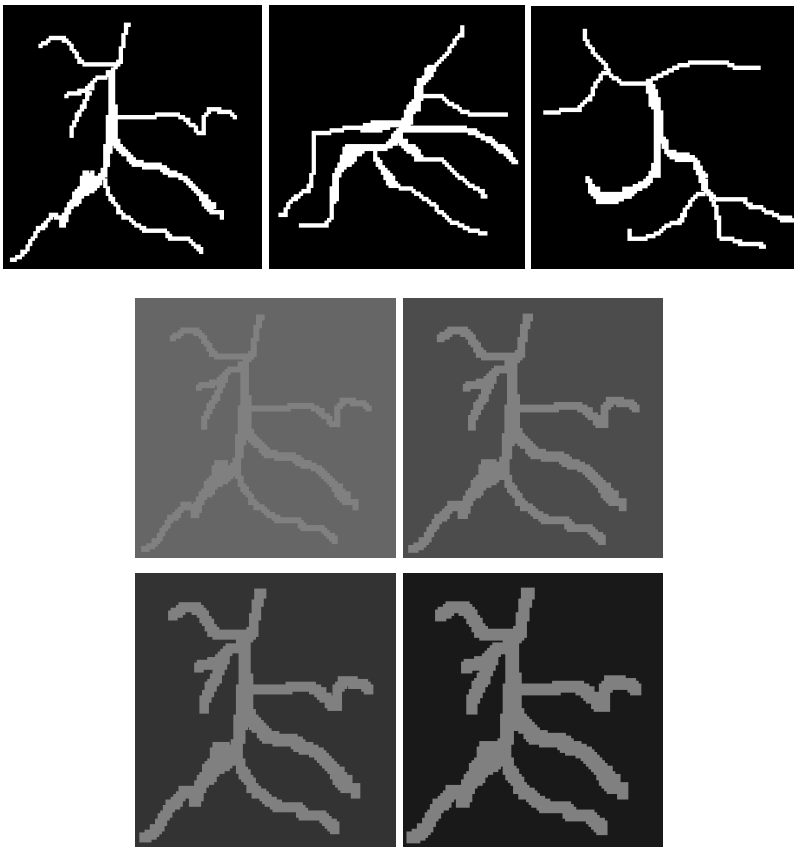


***Figura 4.1*** *– Imagens ao topo: padrões 'A', 'B' e 'C' para a geração de amostras pela extensão. As demais foram geradas a partir do padrão 'A'.*

Depois, aplica-se ruído *speckle* nas mesmas utilizando o código *Matlab*, como mostra a figura 4.2.

***Figura 4.2*** *– Exemplos anteriores com ruído speckle aplicado.*

## 4.2. Testes Realizados

Utilizando as imagens geradas, serão feitos seis testes:

- Segmentação com imagens sem ruído;
- Segmentação em imagens com ruído, sem filtrar;
- Segmentação em imagens com ruído e aplicando filtro Gaussiano;
- Segmentação em imagens com ruído e filtrando por LSMV;
- Segmentação em imagens com ruído e filtrando por Chitwong;
- Segmentação em imagens com ruído e filtrando por ambos os filtros;

Para todos os testes, após a segmentação por FC, foi aplicada uma limiarização de 5%. Logo após, foi feita a contagem de VP e FP utilizando a extensão “Avaliador de Imagens” do ImageJ, descrita anteriormente.

A seguir há a explicação de como cada teste foi feito e os resultados.

### *4.2.1. Imagens sem ruído*

A segmentação de imagens sem ruído, e sem filtragem, tem o intuito de indicar se a implementação do método de segmentação esta correta. Como estão sendo utilizadas imagens simuladas (com somente dois níveis de cor), a acurácia do segmentador tem que ser de aproximadamente 100% de VP e 0% de FP, resultados facilmente alcançados pela maioria dos segmentadores.

Neste teste, somente os padrões ‘A’ foram utilizados, pois como é somente uma certificação de que o algoritmo está bem implementado, foi decidido que não é necessário o uso de tantas amostras. E como esperado, foi constatado que a implementação foi feita com sucesso, observando-se a tabela 4.1, que mostra, para todos os casos, valores de 100% para VP e de 0% para FP.

***Tabela 4.1 –*** *Porcentagem de falso-positivos e verdadeiro-positivos usando o GFC com imagens sem ruído.*

| Diâmetro | Contraste | Imagem sem ruído | |
|---|---|---|---|
| | | %VP | %FP |
| 3 | I | 100 | 0 |
| | II | 100 | 0 |
| | III | 100 | 0 |
| | IV | 100 | 0 |
| | V | 100 | 0 |
| 5 | I | 100 | 0 |
| | II | 100 | 0 |
| | III | 100 | 0 |
| | IV | 100 | 0 |
| | V | 100 | 0 |
| 7 | I | 100 | 0 |
| | II | 100 | 0 |
| | III | 100 | 0 |
| | IV | 100 | 0 |
| | V | 100 | 0 |
| 9 | I | 100 | 0 |
| | II | 100 | 0 |
| | III | 100 | 0 |
| | IV | 100 | 0 |
| | V | 100 | 0 |

### *4.2.2. Imagens com ruído*

A seguir do teste com imagens sem ruído, foram utilizadas imagens ruidosas, variando-se os filtros usados na etapa de pré-processamento. Foi testada também a segmentação de imagens sem a etapa de filtragem, para demonstrar que a ela realmente auxilia na segmentação. Além da porcentagem de VP e FP, é feito o cálculo da média e desvio padrão.

As imagens foram pré-processadas pelos filtros: Gaussiano, LSMV, de Chitwong e ambos (primeiro LSMV e depois o de Chitwong). Na tabela 4.2 são mostrados estes resultados (sem e com filtragem).

***Tabela 4.2 –*** *Porcentagem de falso-positivos e verdadeiro-positivos para o GFC sem filtragem, com filtro Gaussiano, LSMV, de Chitwong e ambos combinados.*

| Contraste | Diâmetro | Padrão | Sem Filtro | | Gaussiano | | Chitwong | | LSMV | | Ambos | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | %VP | %FP | %VP | %FP | %VP | %FP | %VP | %FP | %VP | %FP |
| I | 3 | A | 99.85 | 97.24 | 99.24 | 22.88 | 83.3 | 13.1 | 99.28 | 12.16 | 97.42 | 24.5 |
| | | B | 99.71 | 96.99 | 99.13 | 12.69 | 74.8 | 10.31 | 93.42 | 6.35 | 96.33 | 9.904 |
| | | C | 99.71 | 94.06 | 94.57 | 4.095 | 97.36 | 23.59 | 96.81 | 4.251 | 92.5 | 8.838 |
| | 5 | A | 99.8 | 96.66 | 98.79 | 17.77 | 95.49 | 9.823 | 99.39 | 12.88 | 93.3 | 18.8 |
| | | B | 99.67 | 97.45 | 98.39 | 12.02 | 91.58 | 35.64 | 97.83 | 6.593 | 84.33 | 23.26 |
| | | C | 99.46 | 85.4 | 96.26 | 4.836 | 95.91 | 17.75 | 91.39 | 2.458 | 87.5 | 5.905 |
| | 7 | A | 99.59 | 91.06 | 99.16 | 17.11 | 64.24 | 12.3 | 97.98 | 9.48 | 97.46 | 22.05 |
| | | B | 99.49 | 88.59 | 97.51 | 9.539 | 78.18 | 7.036 | 95.38 | 5.019 | 96.51 | 9.714 |
| | | C | 99.82 | 95.82 | 98.66 | 11.54 | 93.63 | 18.83 | 97.39 | 5.973 | 97.66 | 17.22 |
| | 9 | A | 99.84 | 95.69 | 97.57 | 9.804 | 92.96 | 60.6 | 99.22 | 13.01 | 93.84 | 23.72 |
| | | B | 99.57 | 89.76 | 94.83 | 5.653 | 94.02 | 13.51 | 93.94 | 4.53 | 97.03 | 21.38 |
| | | C | 99.78 | 96.74 | 98.1 | 13.91 | 91.76 | 39.68 | 98.12 | 8.492 | 96.8 | 16.44 |
| II | 3 | A | 99.74 | 87.81 | 93.24 | 3.579 | 87.18 | 13.16 | 85.66 | 3.134 | 96.59 | 28.78 |
| | | B | 99.84 | 98.11 | 99.05 | 12.58 | 80.87 | 18.39 | 98.81 | 6.76 | 98.76 | 11.98 |
| | | C | 99.91 | 95.35 | 98.6 | 7.488 | 82.4 | 25.12 | 98.62 | 5.379 | 97.87 | 8.647 |
| | 5 | A | 99.61 | 90.35 | 96.04 | 6.407 | 87.93 | 52.73 | 96.11 | 5.835 | 93.44 | 25.53 |
| | | B | 99.5 | 79.78 | 94.22 | 3.736 | 94.88 | 25.86 | 94.55 | 3.321 | 96.98 | 5.369 |
| | | C | 99.62 | 83.17 | 97.7 | 5.517 | 94.93 | 18.1 | 93.54 | 2.73 | 84.63 | 6.03 |
| | 7 | A | 99.78 | 91.52 | 96.86 | 6.719 | 90.72 | 15.4 | 98.62 | 8.767 | 95.87 | 15.87 |
| | | B | 99.7 | 93.63 | 96.38 | 7.239 | 92.58 | 24.75 | 94.52 | 4.609 | 63.97 | 3.206 |
| | | C | 99.88 | 97.46 | 97.94 | 8.772 | 93.63 | 56.49 | 97.8 | 5.065 | 94.3 | 7.395 |
| | 9 | A | 99.86 | 96.92 | 98.26 | 10.98 | 96.42 | 16.21 | 99.11 | 12.26 | 96.06 | 15.24 |
| | | B | 98.21 | 91.99 | 98.9 | 25.08 | 91.55 | 48.39 | 99.27 | 14.44 | 96.72 | 17.96 |
| | | C | 99.71 | 88.35 | 98.03 | 7.286 | 80.12 | 10.39 | 95.8 | 3.566 | 95.11 | 4.733 |
| III | 3 | A | 99.63 | 82.49 | 93.9 | 3.412 | 95.61 | 20.43 | 91.69 | 3.031 | 58.86 | 3.941 |
| | | B | 99.9 | 95.34 | 98.45 | 6.84 | 89.82 | 22.06 | 98.06 | 4.22 | 84.1 | 4.729 |
| | | C | 99.89 | 97.77 | 99.69 | 13.73 | 97.58 | 26.48 | 99.03 | 5.303 | 79.41 | 4.484 |
| | 5 | A | 99.74 | 85.57 | 96.13 | 4.518 | 87.37 | 7.927 | 97.68 | 5.946 | 90.29 | 11.14 |
| | | B | 99.64 | 58.63 | 95.47 | 3.743 | 96.79 | 11.73 | 93.26 | 2.687 | 86.65 | 4.512 |
| | | C | 99.88 | 98.4 | 98.67 | 7.958 | 95.08 | 31.11 | 98.03 | 4.621 | 82.71 | 4.024 |
| | 7 | A | 99.73 | 87.36 | 97.67 | 6.213 | 93.91 | 9.33 | 97.16 | 5.302 | 97.85 | 8.668 |
| | | B | 99.75 | 81.57 | 97.07 | 4.74 | 94.8 | 16.3 | 92.45 | 2.784 | 90.98 | 9.43 |
| | | C | 99.86 | 92.01 | 97.92 | 4.503 | 96.32 | 8.76 | 95.44 | 2.662 | 97.96 | 6.729 |
| | 9 | A | 99.8 | 94.51 | 97.6 | 7.675 | 94.29 | 14.56 | 98.53 | 7.74 | 93.17 | 11.14 |

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | B | 99.57 | 78.28 | 96.82 | 5.437 | 96.11 | 11.5 | 95.5 | 4.329 | 97.59 | 8.335 |
| | | C | 99.64 | 87.91 | 98.33 | 8.507 | 95.33 | 20.5 | 96.98 | 4.096 | 95.04 | 99.84 |
| IV | 3 | A | 99.81 | 90.36 | 96.63 | 4.942 | 84.72 | 18.57 | 97.2 | 4.135 | 97.84 | 11.44 |
| | | B | 99.72 | 62.68 | 93.17 | 3.125 | 97.46 | 13.72 | 90.58 | 2.209 | 89.63 | 5.188 |
| | | C | 99.71 | 90.61 | 97.25 | 4.142 | 95.75 | 28.56 | 98.44 | 3.86 | 83.75 | 5.284 |
| | 5 | A | 99.61 | 78.24 | 95.05 | 4.098 | 95.47 | 20.84 | 97 | 5.001 | 96.64 | 14.45 |
| | | B | 99.39 | 27.49 | 95.51 | 3.03 | 89.28 | 8.084 | 82.55 | 1.636 | 76.02 | 3.424 |
| | | C | 99.88 | 85.25 | 97.27 | 3.09 | 97.82 | 13.26 | 93.94 | 2.064 | 96.54 | 4.147 |
| | 7 | A | 99.76 | 93.81 | 97.63 | 6.784 | 57.03 | 9.273 | 98.64 | 7.16 | 62.76 | 3.419 |
| | | B | 99.47 | 38.24 | 98.42 | 6.002 | 96.56 | 13.72 | 93.06 | 2.798 | 92.4 | 99.9 |
| | | C | 99.37 | 28.7 | 95.64 | 3.558 | 96.31 | 7.063 | 93.29 | 2.402 | 96.57 | 4.89 |
| | 9 | A | 99.7 | 87.12 | 94.45 | 5.103 | 96.23 | 15.25 | 98.23 | 6.869 | 88.92 | 8.042 |
| | | B | 98.7 | 16.68 | 93.01 | 2.918 | 94.52 | 12.4 | 88.35 | 1.838 | 69.83 | 3.155 |
| | | C | 99.42 | 59.46 | 93.19 | 3.232 | 94.84 | 18.06 | 89.21 | 2.2 | 94.41 | 99.9 |
| V | 3 | A | 99.7 | 30.97 | 91.98 | 2.296 | 91.84 | 11.09 | 89.96 | 1.928 | 92.14 | 99.94 |
| | | B | 99.53 | 51.1 | 96.83 | 4.086 | 97.92 | 19.92 | 93.58 | 2.502 | 82.44 | 49.97 |
| | | C | 99.97 | 99.23 | 99.19 | 6.74 | 95.14 | 14.53 | 99.78 | 7.022 | 87.41 | 4.858 |
| | 5 | A | 99.38 | 28.88 | 94.38 | 3.055 | 97.18 | 18.28 | 95.99 | 3.547 | 62.52 | 3.092 |
| | | B | 99.77 | 88.95 | 94.14 | 3.504 | 96.34 | 12.05 | 79.92 | 2.076 | 94.2 | 99.94 |
| | | C | 99.76 | 89.62 | 98.25 | 5.3 | 97.02 | 9.106 | 98.26 | 3.877 | 73.42 | 3.318 |
| | 7 | A | 99.59 | 26.67 | 94.77 | 3.086 | 97.52 | 12.52 | 93.83 | 3.073 | 93.37 | 7.938 |
| | | B | 99.45 | 28.1 | 93.78 | 3.008 | 97.03 | 11.71 | 89.29 | 2.109 | 96.29 | 99.96 |
| | | C | 99.68 | 86.18 | 97.5 | 5.79 | 96.95 | 14.78 | 96.83 | 4.157 | 96.48 | 99.98 |
| | 9 | A | 99.49 | 22.75 | 93.67 | 2.992 | 97.63 | 12.93 | 95.09 | 3.678 | 61.35 | 3.013 |
| | | B | 99.14 | 27.33 | 91.99 | 3.28 | 95.94 | 16.37 | 88.98 | 2.399 | 91.03 | 22.77 |
| | | C | 99.74 | 89.25 | 97.42 | 5.006 | 96.17 | 16.86 | 97.45 | 3.994 | 88.77 | 83.94 |
| Média | | | 99.64 | 78.124 | 96.6045 | 7.04455 | 91.869 | 18.947 | 95.097 | 5.039 | 89.34 | 22.857 |
| Desvio Padrão | | | 0.2796 | 24.852 | 2.068 | 4.80645 | 7.9255 | 11.781 | 4.1982 | 3.046 | 10.52 | 30.921 |

## 4.3. Comparação dos Resultados e Estatísticas

Nas seções anteriores foram apresentados os dados coletados nos experimentos propostos utilizando imagens sem ruído, com ruído sem filtragem e com filtragem, mostrados na tabela 4.2.

Para uma melhor avaliação estatística dos resultados, foi feito o teste *t-student* para cada filtro, para demonstrar que cada filtro possuem resultados distintos. Além disso, temos o intuído de demonstrar que eles modificam os resultados obtidos, em relação à segmentação sem pré-processamento (se p <

0.05). O teste foi feito para a porcentagem de verdadeiro-positivos e falso-positivos.

***Tabela 4.3 –*** *Comparação dos dados, de %VP, usando Teste T (t-student).*

| ***p:Prob. Teste-T VP*** | ***Sem Filtros*** | ***Gaussiano*** | ***Chitwong*** | ***LSMV*** | ***Ambos*** |
|---|---|---|---|---|---|
| **Sem Filtros** | X | 1.80286E-17 | 1.48067E-10 | 3.47835E-12 | 1.26766E-10 |
| **Gaussiano** | 1.80286E-17 | X | 5.00936E-05 | 0.000269111 | 4.80447E-07 |
| **Chitwong** | 1.48067E-10 | 5.00936E-05 | X | 0.005776322 | 0.074199831 |
| **LSMV** | 3.47835E-12 | 0.000269111 | 0.005776322 | X | 6.8292E-05 |
| **Ambos** | 1.26766E-10 | 4.80447E-07 | 0.074199831 | 6.8292E-05 | X |

Os resultados (p) do teste *t-student* para os valores de porcentagem de VP são apresentados na tabela 4.3 e na tabela 4.4 os de FP. Eles serão analisados no próximo capítulo: “Conclusões”. Nessas tabelas são feitas comparações entre todas as possibilidades.

***Tabela 4.4 –*** *Comparação dos dados, de %FP, usando Teste T (t-student).*

| ***p:Prob. Teste-T FP*** | ***Sem Filtros*** | ***Gaussiano*** | ***Chitwong*** | ***LSMV*** | ***Ambos*** |
|---|---|---|---|---|---|
| **Sem Filtros** | X | 1.52137E-32 | 2.59485E-27 | 1.31411E-32 | 2.20185E-14 |
| **Gaussiano** | 1.52137E-32 | X | 1.29235E-11 | 5.21298E-07 | 0.00014459 |
| **Chitwong** | 2.59485E-27 | 1.29235E-11 | X | 3.10705E-14 | 0.187067931 |
| **LSMV** | 1.31411E-32 | 5.21298E-07 | 3.10705E-14 | X | 2.56785E-05 |
| **Ambos** | 2.20185E-14 | 0.00014459 | 0.187067931 | 2.56785E-05 | X |

A tabela 4.5 mostra os valores de acurácia, calculados a partir das porcentagens de VP e FP da tabela 4.2, usando a equação 17:

$$A = (VP/100 + (1 - FP/100)).100 \quad (17)$$

Onde VP e FP são dados em porcentagens.

***Tabela 4.5 –*** *Acurácia calculada para cada filtro.*

| ***Acurácia*** | ***Sem Filtros*** | ***Gaussiano*** | ***Chitwong*** | ***LSMV*** | ***Ambos*** |
|---|---|---|---|---|---|
| **Porcentagens** | 60.75 | 94.77 | 86.46 | 95.02 | 83.24 |

# 5. CONCLUSÕES

Para o sistema segmentador, utilizando o *Fuzzy Connectedness,* pode-se observar pelas acurácias da tabela 4.5*,* que a melhor filtragem é a feita pelo LSMV. Pode-se ver também que mesmo não tendo a melhor taxa de VP médio, possui a menor média de FP. Para o VP, quanto maior melhor e para FP quanto menor melhor, porém deve-se haver um equilíbrio entre esses valores, por isso é utilizada a acuarácia. Altas taxas de VP e FP, ao mesmo tempo, indicam que o segmentador não tem um bom critério para definir o que é objeto e o que é fundo na imagem.

As tabelas 4.3 e 4.4 mostram os resultados do teste-t de *student* (o cruzamento linha x coluna da tabela), para cada filtro, para verificar a hipótese nula de que os dois métodos têm desempenhos similares. Na realidade o interesse é na rejeição da hipótese nula, e afirmar com alguma significância estatística ($p< 0,05$) que um método é estatisticamente distinto do outro. E isso é confirmado pelos resultados, confirmando que o desempenho é consistentemente diferente, apenas para o filtro de Chitwong e o uso combinado possui resultados semelhantes.

Verificou-se também, que o uso do Chitwong após o LSMV, ao contrário do esperado, não diminuiu as taxas de FP, e tampouco aumentou as de VP. E que o Chitwong sozinho não é uma boa opção.

Também foi testado o filtro Gaussiano, e o mesmo apresentou bons resultados para VP, porém em relação ao FP, não foi melhor que o LSMV.

## 5.1. Discussão

Nos testes, foi observado que quando o filtro é aplicado, ele tende a piorar a qualidade das bordas da imagem, fazendo com que a segmentação de

uma imagem sem filtragem, em termos de VP, seja melhor do que com as imagens filtradas.

Essa pesquisa constatou o que foi dito em estudos anteriores [LPC-05], que provou que o LSMV é bom para a filtragem de ruído *speckle*. Isso pode ser visto nos nossos resultados também: alta %VP, porém com baixa %FP.

Já o filtro Chitwong, quando é utilizado em imagens ruidosas, e de baixo contraste, apresenta péssimos resultados, pois a granulação do ruído Gaussiano, que compõe o ruído *speckle,* tende a ser acentuado.

Era esperado que a combinação de filtros mostrasse melhores resultados, pois o filtro LSMV tem como característica suavizar a imagem, retirando boa parte do ruído Gaussiano, e o Chitwong que tem como característica destacar bordas. Quando é feita a suavização da imagem com o LSMV, as bordas são atenuadas também, e então o Chitwong deveria recuperá-las. Porém foi constatado que para ruído *speckle*, não é uma opção interessante.

## 5.2. Contribuições

A maior contribuição foi a investigação do uso combinado de filtros, e estudos parecidos como esses, envolvendo a combinação de filtro. A combinação escolhida não apresentou resultados satisfatórios, porém com pequenos ajustes dos filtros, pode abrir a possibilidade de resultados melhores.

A utilização do sistema segmentador também foi importante, assim como a organização das provas e a utilização do teste *t-student* foram importantes para o sucesso dessa pesquisa.

## 5.3. Trabalhos Futuros

O primeiro ponto que pode ser estudado, ainda utilizando imagens simuladas 2D, é a utilização de outros métodos de segmentação, como o *watershed*(), *snakes*(), além do próprio *Fuzzy Connectedness* utilizando a

abordagem IFT (), para verificar se são obtidos os mesmo resultados, se não utilizasse tal abordagem.

E com a variação dos métodos de segmentação, utilizar os mesmo filtros: LSMV, Chitwong, ambos e sem filtros, sendo assim possível a identificação da melhor opção filtro/segmentador.

Depois dessa abordagem, seria interessante a utilização de mais opções de filtragem, como a difusão anisotrópica, outras combinações, além da própria melhoria do filtro LSMV, no ponto de vista da adaptabilidade ao meio, e interface com usuário, tentando criar uma forma de seleção de área de ruído local automaticamente.

Importante também, utilizando a mesma metodologia, aplicar em imagens reais 2D e em imagens 3D. A maior dificuldade será no uso do filtro Chitwong, pois seria necessária a criação de várias possibilidades de janela, para a detecção do sentido da borda do volume de interesse.

Outro ponto a ser estudado são outras abordagens do *Fuzzy Connectedness,* como o uso de várias sementes etc.

Além disso, com o uso de imagens reais de ecocardiografia é interessante que seja usados na avaliação os outros critérios citados por Loizou, que é o uso de especialistas na área para a segmentação manual das imagens de teste, para servir como padrão de comparação.

# APÊNDICE A - PRODUÇÃO BIBLIOGRÁFICA

Sibgrapi 2009 – "***Coronary Segmentation from Echocardiography using Fuzzy Connectedness****",* A.F.L.Souza, D. M. Lage e S. S. Furuie*.*

VI Simpósio de Instrumentação e Imagens Médicas 2009 – *"**Investigação de filtros adaptativos em imagens ecocardiográficas para segmentação de estruturas**"*, A. F. L. Souza e S. S. Furuie.

# APÊNDICE B – CÓDIGOS JAVA DAS EXTENSÕES DO IMAGEJ

## B.1 – Gerador de Imagens

```
import ij.plugin.*;
import java.awt.*;
import java.io.*;

import ij.*;
import ij.io.*;
import ij.process.*;
import ij.gui.*;

/**
 * Phantom generator
 * @author: Andre Fernando Lourenco de Souza
 * @version: 1.0
 * @created: Aug-2009
 * EPUSP - Polytechnic School of Sao Paulo - Brazil
 * LEB - Laboratorio de Engenharia Biomedica
 * andreflsouza@usp.br
*/
public class Phantom_Generator implements PlugIn {

	int maxContrast = 128;
	int qtdContrast = 3;
	int qtdDiameter = 4;
	String dirDest;
	String dirSrc;
	boolean batchProcess = false;

	public void run(String arg) {
		if (!showDialog())
			return;

		ImagePlus img = null;
		String[] list = null;

		if (batchProcess) {
			list = new File(dirSrc).list();
			if (list == null) {
				return;
			}
		} else {
			IJ.open("");
			img = WindowManager.getCurrentImage();
			if (img == null) {
			IJ.noImage();
			return;
		  }
		}

		for (int i = 0; !batchProcess || i < list.length; i++) {
			if (batchProcess) {
```

```
                        if (list[i] == "Thumbs.db")
                              continue;
                        img = IJ.openImage(dirSrc+list[i]);
                        if (img == null)
                              continue;
                  }

                  int ptr = 1;
                  for (int d = 1; d <= qtdDiameter; d++) {
                        img.getProcessor().invert();
                        img.getProcessor().dilate();
                        img.getProcessor().invert();
                        img.updateAndRepaintWindow();
                        String nameC = "I";

                        ImageProcessor imgOrig =
img.getProcessor().duplicate();
                        if (img == null)
                              return;

                        for (int c = qtdContrast-1; c >= 0; c--) {
                              String fileName = "P" + i + "_" + "0" +
ptr++ + "_" + nameC + "_" + ((2*d)+1) + ".bmp";
                              System.out.println("dirDest = [" + dirDest
+ "] Nome saida = [" + fileName + "]");

                              setContrast (imgOrig, img, maxContrast,
(int)((float)maxContrast*((float)c/(float)qtdContrast)));

                              WindowManager.setTempCurrentImage(img);
                              IJ.save(dirDest + fileName);

                              nameC = nameC + "I";
                        }
                        img.setProcessor("", imgOrig);  /* just to
remove the modifications done in contrast */
                  }

                  if (!batchProcess) {
                        break;
                  }
            }
      }

      /*
       * Shows the input dialog.
       */
      boolean showDialog() {
            GenericDialog gd = new GenericDialog("Phantom Generator");
            gd.addNumericField("Contrast max:", maxContrast, 3);
            gd.addNumericField("Qtd Contrast:", qtdContrast, 3);
            gd.addNumericField("Qtd diameter:", qtdDiameter, 3);
            gd.addCheckbox("Batch process", false);
            gd.showDialog();
            if (gd.wasCanceled())
                  return false;
            maxContrast = (int)gd.getNextNumber();
            qtdContrast = (int)gd.getNextNumber();
            qtdDiameter = (int)gd.getNextNumber();
```

```
            batchProcess = gd.getNextBoolean();
            if (batchProcess) {
                  dirSrc = IJ.getDirectory("Select source folder...");
                  if (dirSrc == null)
                        return false;
            }

            dirDest = IJ.getDirectory("Select output folder...");
            if (dirDest == null)
                  return false;

            return true;
      }

      /*
       * We use the imgOrig just to see if the desired point is
background or object.
       */
      ImagePlus setContrast (ImageProcessor imgOrig, ImagePlus img, int
valueHigh, int valueLow) {
            int width = img.getWidth();
            int height = img.getHeight();

            for (int y = 0; y < height; y++) {
                  for (int x = 0; x < width; x++) {
                        if (imgOrig.get(x, y) > 0)
                              img.getProcessor().set(x, y, valueHigh);
                        else
                              img.getProcessor().set(x, y, valueLow);
                  }
            }

            return img;
      }
}
```

## B.2 – Filtro Chitwong

```
import java.awt.*;
import java.io.File;
import java.util.*;

import ij.*;
import ij.gui.*;
import ij.plugin.*;
//import ij.plugin.filter.*;
import ij.process.*;

/**
 * Chitwong Filter
 * @author: Andre Fernando Lourenco de Souza
 * @version: 1.0
 * @created: Aug-2009
 * EPUSP - Polytechnic School of Sao Paulo - Brazil
 * LEB - Laboratorio de Engenharia Biomedica
 * andreflsouza@usp.br
*/
public class Filter_Chitwong  implements PlugIn/*PlugInFilter*/ {
	/*
	 * Some attribute declaration.
	 */
	int stackSize;
	int width;
	int height;
	boolean is3D;
	boolean batchProcess = true;
	String dirSrc;
	String dirDest;
	int iterations = 1;
	boolean showInterImages = true;

	/*
	 * Constants
	 */
/*
	static final int [][][] masks2D = { {  {0, 2},  {0, 1},  {0,-1},
{0,-2}},
                                                                { {-2, 0},
{-1, 0},  {1, 0},  {2, 0}},

                                                                { {-2,-2},
{-1,-1},  {1, 1},  {2, 2}},
                                                                { {-2, 2},
{-1, 1},  {1,-1},  {2,-2}},  //{  {2, 2},  {1, 1}, {-1,-1}, {-2,-2}},

                                                                { {-1, 1},
{1,-1},  {0,-1}, {-1,-1}},
                                                                {  {1, 1},
{-1,-1},  {0,-1},  {1,-1}},
                                                                { {-1,-1},
{1, 1},  {0, 1}, {-1, 1}},
                                                                {  {1,-1},
{-1, 1},  {0, 1},  {1, 1}},
```

```
                                                                { {-2,-2},
{-1,-1}, {-1, 1}, {-2, 2}},
                                                                { {-2, 2},
{-1, 1},  {1, 1},  {2, 2}},
                                                                {  {2,-2},
{1,-1},  {1, 1},  {2, 2}},
                                                                { {-2,-2},
{-1,-1},  {1,-1},  {2,-2}}
                                                            };
*/
      static final int [][][] masks2D = {
            {  {0, 8},  {0, 7},  {0, 6},  {0, 5},  {0, 4},  {0, 3},  {0,
2},  {0, 1},  {0,-1},  {0,-2},  {0,-3},  {0,-4},  {0,-5},  {0,-6},  {0,-
7},  {0,-8}},
            { {-8, 0}, {-7, 0}, {-6, 0}, {-5, 0}, {-4, 0}, {-3, 0}, {-2,
0}, {-1, 0},  {1, 0},  {2, 0},  {3, 0},  {4, 0},  {5, 0},  {6, 0},  {7,
0},  {8, 0}},

            { {-8,-8}, {-7,-7}, {-6,-6}, {-5,-5}, {-4,-4}, {-3,-3}, {-
2,-2}, {-1,-1},  {1, 1},  {2, 2},  {3, 3},  {4, 4},  {5, 5},  {6, 6},
{7, 7},  {8, 8}},
            { {-8, 8}, {-7, 7}, {-6, 6}, {-5, 5}, {-4, 4}, {-3, 3}, {-2,
2}, {-1, 1},  {1,-1},  {2,-2},  {3,-3},  {4,-4},  {5,-5},  {6,-6},  {7,-
7},  {8,-8}},

            { {-7, 7}, {-6, 6}, {-5, 5}, {-4, 4}, {-3, 3}, {-2, 2}, {-1,
1},  {1,-1},  {0,-1}, {-1,-1}, {-2,-1}, {-3,-1}, {-4,-1}, {-5,-1}, {-6,-
1}, {-7,-1}},
            {  {7, 7},  {6, 6},  {5, 5},  {4, 4},  {3, 3},  {2, 2},  {1,
1}, {-1,-1},  {0,-1},  {1,-1},  {2,-1},  {3,-1},  {4,-1},  {5,-1},  {6,-
1},  {7,-1}},
            { {-7,-7}, {-6,-6}, {-5,-5}, {-4,-4}, {-3,-3}, {-2,-2}, {-
1,-1},  {1, 1},  {0, 1}, {-1, 1}, {-2, 1}, {-3, 1}, {-4, 1}, {-5, 1}, {-
6, 1}, {-7, 1}},
            {  {7,-7},  {6,-6},  {5,-5},  {4,-4},  {3,-3},  {2,-2},
{1,-1}, {-1, 1},  {0, 1},  {1, 1},  {2, 1},  {3, 1},  {4, 1},  {5, 1},
{6, 1},  {7, 1}},

            { {-8,-8}, {-7,-7}, {-6,-6}, {-5,-5}, {-4,-4}, {-3,-3}, {-
2,-2}, {-1,-1}, {-1, 1}, {-2, 2}, {-3, 3}, {-4, 4}, {-5, 5}, {-6, 6}, {-
7, 7}, {-8, 8}},
            { {-8, 8}, {-7, 7}, {-6, 6}, {-5, 5}, {-4, 4}, {-3, 3}, {-2,
2}, {-1, 1},  {1, 1},  {2, 2},  {3, 3},  {4, 4},  {5, 5},  {6, 6},  {7,
7},  {8, 8}},
            {  {8,-8},  {7,-7},  {6,-6},  {5,-5},  {4,-4},  {3,-3},
{2,-2},  {1,-1},  {1, 1},  {2, 2},  {3, 3},  {4, 4},  {5, 5},  {6, 6},
{7, 7},  {8, 8}},
            { {-8,-8}, {-7,-7}, {-6,-6}, {-5,-5}, {-4,-4}, {-3,-3}, {-
2,-2}, {-1,-1},  {1,-1},  {2,-2},  {3,-3},  {4,-4},  {5,-5},  {6,-6},
{7,-7},  {8,-8}}
};

      static final int WEST = 0;
      static final int NORTH = 1;
      static final int EAST = 2;
      static final int SOUTH = 3;
      static final int BEFORE = 4;
      static final int AFTER = 5;

      /**
```

```
	 * Run method. (the main function)
	 * @see ij.plugin.PlugIn#run(java.lang.String)
	 */
	public void run(String arg) {
		/*
		 * Calls input dialog.
		 */
		if (!Dialogue())
			return;

		if (batchProcess == false) {
			/*
			 * Some input verifications
			 */
			ImagePlus imp = WindowManager.getCurrentImage();
		  if (imp==null) {
			IJ.noImage();
			return;
		  }
		  if (!(imp.getProcessor() instanceof ByteProcessor)) {
			IJ.showMessage("Filter Chitwong", "8-bit image or
stack required.");
			return;
		  }

			if (imp.getStackSize() < 2) {
				is3D = false;
			} else {
				is3D = true;
			}

			/* To check the processing time */
			Date t0 = new Date();

			/* Get stack info */
			ImageStack stack = imp.getStack();
			int stackSize = stack.getSize();
			width = stack.getWidth();
			height = stack.getHeight();

			/* Get stack arrays */
			//byte[][] tabstack = new byte[stackSize][width *
height];
			byte[] tabstack = new byte[width * height];
			for (int i = 1; i <= stackSize; i++) {
				for (int j = 0; j < width * height; j++) {
	//				tabstack[toIndex2D(i,j)] = ((byte[])
stack.getPixels(i))[j];
					tabstack[j] = ((byte[])
stack.getPixels(i))[j];
				}
			}

			//byte[][] res = new byte[1][1];
			byte[] res = new byte[1];
			for (int it = 0; it < iterations; it++) {
				res = filter_chitwong (tabstack, width, height,
stackSize, is3D);
```

```
                        if (showInterImages) {
                              ImageStack stak = new ImageStack(width,
height);
                              stak.addSlice("", res);
                              ImagePlus said = new
ImagePlus(String.valueOf(it), stak);
                              said.show();
                        }

                        /* Copy the output to input - for another
iteration. */
                        int ptr = 0;
                        for (int y = 0; y < height; y++) {
                              for (int x = 0; x < width; x++) {
                                    tabstack[ptr] = res[ptr];
                                    ptr++;
                              }
                        }
                  }

                  /* Get the result as a image stack and display it */
                  ImageStack resstack = new ImageStack(width, height);
                  resstack.addSlice("", res);
      //          for (int i = 0; i < stackSize; i++) {
      //                resstack.addSlice("", res[i]);
      //          }
                  ImagePlus saida = new ImagePlus("Image Filtered!
Chitwong", resstack);
                  saida.show();
                  WindowManager.setTempCurrentImage(saida);
                  IJ.save("D:\\temp\\c_" + imp.getTitle());

                  /* time to process */
                  Date t1 = new Date();
                  System.out.println("time:" + (t1.getTime() -
t0.getTime()) + " ms");
            } else {
                  String[] list = new File(dirSrc).list();
                  if (list==null)
                        return;
                  for (int i=0; i<list.length; i++) {
                        if (list[i] == "Thumbs.db")
                              continue;
                        ImagePlus img = IJ.openImage(dirSrc+list[i]);
                        if (img==null)
                              continue;

                        width = img.getWidth();
                        height = img.getHeight();
                        byte[] tabstack = new byte[width * height];
                        int ptr = 0;
                        for (int y = 0; y < height; y++) {
                              for (int x = 0; x < width; x++) {
                                    tabstack[ptr++] = (byte)
img.getProcessor().getPixel(x, y);
                              }
                        }

                        byte[] res = new byte[1];
```

```
                        ImageStack resstack = new ImageStack(width,
height);
                        for (int it = 0; it < iterations; it++) {

                              res = filter_chitwong (tabstack, width,
height, stackSize, is3D);
                              if (showInterImages) {
                                    resstack.deleteLastSlice();
                                    resstack.addSlice("", res);
                                    ImagePlus saida = new
ImagePlus(String.valueOf(it), resstack);
                                    saida.show();
                              }

                              /* Copy the output to input - for another
iteration. */
                              int pt = 0;
                              for (int y = 0; y < height; y++) {
                                    for (int x = 0; x < width; x++) {
                                          tabstack[pt] = res[pt];
                                          pt++;
                                    }
                              }
                        }

                        /* Get the result as a image stack and display
it */
                        ByteProcessor imgByte = new ByteProcessor(width,
height, res, img.getProcessor().getColorModel());
                        ImagePlus saida = new ImagePlus("Image
Filtered", imgByte);

                        WindowManager.setTempCurrentImage(saida);
                        IJ.save(dirDest + "//c_" + list[i]);
                  }
            IJ.showMessage("Filter Chitwong", "Finished!");
            }
      }


      /**
       *  Dialog of the plugin.
       */
      private boolean Dialogue() {

            GenericDialog gd = new GenericDialog("Filter Chitwong");

            String [] tpDim = new String [2];;
            tpDim[0] = "2D";
            tpDim[1] = "3D";
            gd.addChoice("Dimensões", tpDim, tpDim[0]);
            gd.addNumericField("# of iterations:", iterations, 3);

            gd.addCheckbox("Show intermediate images?",
showInterImages);
            gd.addCheckbox("Batch process", batchProcess);

            gd.showDialog();
            if (gd.wasCanceled())
                  return false;
```

```
            if (gd.getNextChoice().equals(tpDim[0]))
                  is3D = false;
            else
                  is3D = true;

            iterations = (int)gd.getNextNumber();
            showInterImages = gd.getNextBoolean();
            batchProcess = gd.getNextBoolean();
            if (batchProcess) {
                  dirSrc = IJ.getDirectory("Select source folder...");
                  if (dirSrc == null)
                        return false;
                  dirDest = IJ.getDirectory("Select destination
folder...");
                  if (dirDest == null)
                        return false;
            }

            return (true);
      }

      /*
       * Return the absolute index for a given (x,y) pair.
       * Solve the bound problem copying the bound values (and not just
set to zero or 255).
       */
      private int toIndex2D(int x, int y) {
            /* If it is in the boundaries, use the bound value */
            if (x < 0)
                  x = 0;
            else if (x >= width)
                  x = width-1;

            if (y < 0)
                  y = 0;
            else if (y >= height)
                  y = height-1;

            return ((y*width) + x);
      }

      private int toIndex2D(int x, int y, int direction) {
            int destx = x, desty = y;

            switch (direction) {
            case WEST:
                  destx = x - 1;
                  break;
            case NORTH:
                  desty = y - 1;
                  break;
            case EAST:
                  destx = x + 1;
                  break;
            case SOUTH:
                  desty = y + 1;
                  break;
            }
```

```
		return (toIndex2D(destx, desty));
	}


	/*
	 * For 3D
	 */
	private int toIndex3D(int x, int y, int z) {
		/* If it is in the boundaries, use the bound value */
		if (x < 0)
			x = 0;
		else if (x >= width)
			x = width-1;

		if (y < 0)
			y = 0;
		else if (y >= height)
			y = height-1;

		if (z < 0)
			z = 0;
		else if (z >= stackSize)
			z = stackSize-1;

		return (z*(width*height) + (y*width) + x);
	}

	private int toIndex3D(int x, int y, int z, int direction) {
		int destx = x, desty = y, destz = z;

		switch (direction) {
		case WEST:
			destx = x - 1;
			break;
		case NORTH:
			desty = y - 1;
			break;
		case EAST:
			destx = x + 1;
			break;
		case SOUTH:
			desty = y + 1;
			break;
		case BEFORE:
			destz = z - 1;
			break;
		case AFTER:
			destz = z + 1;
			break;
		}
		return (toIndex3D(destx, desty, destz));
	}

	/*
	 * The filter
	 */
//	byte[][] filter_chitwong (byte [][] tabstack, int w, int h, int s,
boolean is3D) {
```

```
      byte[] filter_chitwong (byte [] tabstack, int w, int h, int s,
boolean is3D) {

//          byte[][] output = new byte[w*h][s];
            byte[] output = new byte[w*h];

            System.out.println("Masks " + masks2D.length + " " +
masks2D[0].length + " " + masks2D[0][0].length);

            /* Sliding window for all pixels */
            for (int i = 0; i < w; i++) {
                  for (int j = 0; j < h; j++) {
//                      for (int k = 0; k < s; k++) {
                              /* Gets the center value */
//                            byte center = tabstack[toIndex2D(i,
j)][k];
                              byte center = tabstack[toIndex2D(i, j)];
                              //System.out.println("Center " + center);

                              /* Calculates the variance for all masks
*/
                              int minor_variance = 0xFFFF;
                              int idx_minor_variance = -1;
                              for (int m = 0; m < masks2D.length; m++) {

                                    /* Variance between center and the
other mask values */
                                    int variance = 0;
                                    int total_variance = 0;
                                    for (int mV = 0; mV <
masks2D[m].length; mV++) {
//                                        byte value =
tabstack[toIndex2D(i+masks2D[m][mV][0], j+masks2D[m][mV][1])][k];
                                        byte value =
tabstack[toIndex2D(i+masks2D[m][mV][0], j+masks2D[m][mV][1])];
                                        int diff = center - value;
                                        variance += (diff*diff);
                                        //System.out.println("Value "
+ value + " diff " + diff + " variance " + variance);
                                    }
                                    total_variance = variance /
masks2D[m].length;
                                    if (total_variance < minor_variance)
{
                                          minor_variance =
total_variance;
                                          idx_minor_variance = m;
                                    }
                                    //System.out.println("m " + m + "
total_variance " + total_variance + " masks2D[m].length " +
masks2D[m].length + " minor_variance " + minor_variance + "
idx_minor_variance " + idx_minor_variance);
                              }
                              //System.out.println("idx_minor_variance =
" + idx_minor_variance + " " + minor_variance);

                              /* Calculates the mean for the minor
variance mask */
                              int mean = 0;
```

```
                                for (int me = 0; me <
masks2D[idx_minor_variance].length; me++) {
//                                  byte value =
tabstack[toIndex2D(i+masks2D[idx_minor_variance][me][0],
j+masks2D[idx_minor_variance][me][1])][k];
                                    byte value =
tabstack[toIndex2D(i+masks2D[idx_minor_variance][me][0],
j+masks2D[idx_minor_variance][me][1])];
                                    mean += value;
                                }
                                mean += center;
                                mean /=
(masks2D[idx_minor_variance].length+1);

//                                output[toIndex2D(i, j)][k] = (byte)mean;
                                output[toIndex2D(i, j)] = (byte)mean;
//                          }
                        }
                    }

                    return output;
            }
}
```

### B.3 – Filtro LSMV

```
import ij.plugin.*;
import ij.*;
import ij.process.*;

//import filters.lsmv;
import java.awt.Point;
import java.awt.Rectangle;
import java.io.File;
import java.util.Date;

import ij.gui.GenericDialog;
import ij.gui.NewImage;
import ij.gui.Roi;

/**
 * Chitwong Filter
 * @author: Andre Fernando Lourenco de Souza
 * @version: 1.0
 * @created: Aug-2009
 * EPUSP - Polytechnic School of Sao Paulo - Brazil
 * LEB - Laboratorio de Engenharia Biomedica
 * andreflsouza@usp.br
*/
public class Filter_lsmv_2D implements PlugIn{
  //  ImagePlus imp;
  //  ImageProcessor ip;

      boolean batchProcess = true;
      String dirSrc;
      String dirDest;

      ///c; // b; // a;
      int leftX[] = {122/*91*/, 91, 81};
      //81; //91; //91;
      int leftY[] = {122/*196*/, 162, 214};
      //214; //162; //196;
      int larg[] = {10/*14*/, 15, 18};
      //18; //15; //14;
      int alt[] = {29/*16*/, 10, 11};
      //11; //10; //16;

      Roi roi_non_batch;

      /**
       *  Dialog of the plugin.
       */
      private boolean Dialogue() {

            GenericDialog gd = new GenericDialog("LSMV 2D");

//          gd.addNumericField("Rectangle Top Left X:", leftX, 1);
//          gd.addNumericField("Rectangle Top Left Y:", leftY, 1);
```

```
//          gd.addNumericField("Rectangle width:", larg, 1);
//          gd.addNumericField("Rectangle height:", alt, 1);
            gd.addCheckbox("Batch process", batchProcess);

            gd.showDialog();
            if (gd.wasCanceled())
                  return false;

//          leftX = (int)gd.getNextNumber();
//          leftY = (int)gd.getNextNumber();
//          larg = (int)gd.getNextNumber();
//          alt = (int)gd.getNextNumber();
            batchProcess = gd.getNextBoolean();
            if (batchProcess) {
                  dirSrc = IJ.getDirectory("Select source folder...");
                  if (dirSrc == null)
                        return false;
            }
            dirDest = IJ.getDirectory("Select destination folder...");
            if (dirDest == null)
                  return false;

            return (true);
      }

    /*
     * (non-Javadoc)
     * @see ij.plugin.PlugIn#run(java.lang.String)
     *  The main method.
     */
      public void run(String args) {
            /*
             * Calls input dialog.
             */
            if (!Dialogue())
                  return;

            /* To check the processing time */
            Date t0 = new Date();

            if (batchProcess == false) {
                        /*
                         * Some input verifications
                         */
                        ImagePlus imp = WindowManager.getCurrentImage();
                  if (imp==null) {
                        IJ.noImage();
                        return;
                  }
                  if (!(imp.getProcessor() instanceof ByteProcessor))
{
                        IJ.showMessage("LSMV filter", "8-bit image or
stack required.");
                        return;
                  }
```

```
                roi_non_batch = imp.getRoi();
                if (roi_non_batch == null) {
                  IJ.showMessage("FC segmentador", "A rectangle
selection is required (to be the seed area).");
                      return;
                  }

                      doLSMV(imp.getProcessor(), dirDest + "//l_" +
imp.getTitle(), batchProcess, -1);

            } else {
                      String[] list = new File(dirSrc).list();
                      if (list==null)
                            return;
                      for (int i=0; i<list.length; i++) {
                            if (list[i] == "Thumbs.db")
                                  continue;
                            ImagePlus imp =
IJ.openImage(dirSrc+list[i]);
                            if (imp==null)
                                  continue;

                            // nome do arquivo = "*P<indice do
padrao>_*.bmp"
                            int idxStr = list[i].indexOf("P") + 1;
                            int idxS =
Integer.parseInt(list[i].substring(idxStr, idxStr+1));
                            System.out.println("nome = " + list[i] + "
idxStr " + idxStr + " idxS " + idxS);
                            doLSMV(imp.getProcessor(), dirDest +
"//l_" + list[i], batchProcess, idxS);
                      }
            }

            /* time to process */
            Date t1 = new Date();
            System.out.println("time:" + (t1.getTime() - t0.getTime()) +
" ms");
        IJ.showMessage("Filter LSMV", "Finished!");
    }

    /*
     *
     */
    private void doLSMV(ImageProcessor ip, String fileName, boolean
isBatch, int idxSeed) {

      byte[] original = (byte[]) ip.getPixels();
        byte[] filtrada = new byte[original.length];

        lsmv teste = new lsmv(original,ip.getHeight(),ip.getWidth());

        Roi roi_s;
        if (idxSeed == -1)
            roi_s = roi_non_batch;
        else
```

```
            roi_s = new Roi (leftX[idxSeed], leftY[idxSeed],
larg[idxSeed], alt[idxSeed]);

        /*
         * Calculates the Noise from the given window.
         */
        teste.calculateGlobalVarNoise (roi_s.getBounds());

        ImagePlus nova = NewImage.createByteImage("Filtro Despeckle",

      ip.getWidth(), ip.getHeight(),
                                                                1,
NewImage.FILL_WHITE);
        ImageProcessor newImage = nova.getProcessor();

            ImagePlus novaK = NewImage.createByteImage("Constante
K",ip.getWidth(),ip.getHeight(),1,NewImage.FILL_WHITE);
            ImageProcessor newImageK = novaK.getProcessor();
        for (int i=0;i<ip.getHeight();i++){
            for (int j=0;j<ip.getWidth();j++){
                float k = teste.getK(i, j);
                int pixel = teste.getNewPixel(i, j);

                newImage.putPixel(j,i,pixel);
                newImageK.putPixel(j,i,(int)(255*k));
            }
        }

        if (!isBatch) {
            nova.show();
            nova.updateAndDraw();

            novaK.show();
            novaK.updateAndDraw();
        }

            WindowManager.setTempCurrentImage(nova);
            IJ.save(fileName);
    }
}

////////////////////////////////////////////////////////////
import java.awt.Rectangle;


public class lsmv {
    byte[] filtImage;
    byte[] original;
    int height;
    int width;
    int frames;
    float varNoise;
    float meanWin;
    float varWin;

    float[] constK;
```

```
    int WINDOW = 7;

    /****************** CONSTRUCTORS ******************/
    public lsmv(byte[] pixels, int heig, int wid) {
        varNoise = 0;
        this.height = heig;
        this.width = wid;
        original = new byte[heig * wid];
        System.arraycopy(pixels, 0, original, 0, pixels.length);
        filtImage = new byte[heig * wid];
        constK = new float[heig * wid];
    }

    /******************           ******************/
    /*
     * Perform the calculation.
     * K ~ 0 -> homogeneous area -> output = mean
     * K ~ 1 -> bound area (great variance) -> output = input (do
nothing)
     */
    public int getNewPixel(int x, int y) {
        int actualPixel = (int) (original[toIndex(x, y)] & 0xff);
        float gMean = this.getMean(x, y);

        return (int) (gMean + (this.getK(x, y) * (actualPixel -
gMean)));
    }

    /******************           ******************/
    /*
     * Calculates the Noise from the given window (global noise).
     */
    public float calculateGlobalVarNoise(Rectangle rec) {
      varNoise = 0;

      float mean = this.getGlobalMean(rec);
      float var = this.getGlobalVar(mean, rec);
      varNoise = var / mean;

      return varNoise;
    }

    private float getGlobalMean(Rectangle rec) {
        double sum = 0;
        double rep = 0;
        int pixel;

      for (int y = 0; y < rec.height; y++) {
            for (int x = 0; x < rec.width; x++) {
                pixel = (int) (original[toIndex(x+rec.x, y+rec.y)] &
0xff);
                sum += pixel;
                rep++;
            }
        }

        return ((float) (sum / rep));
    }
```

```
    private float getGlobalVar(float mean, Rectangle rec) {
        double sum = 0;
        double rep = 0;
        int pixel;

      for (int y = 0; y < rec.height; y++) {
            for (int x = 0; x < rec.width; x++) {
                pixel = (int) (original[toIndex(x+rec.x, y+rec.y)] &
0xff);
                sum += (pixel - mean) * (pixel - mean);
                rep++;
            }
        }

        return ((float) (sum / (rep/* - 1*/)));
    }

    /******************* LOCAL STATISTICS ******************/
    public float getK(int x, int y) {
        meanWin = this.getMean(x, y);
        varWin = this.getVar(meanWin, x, y);

        constK[toIndex(x, y)] = varWin / (varWin + (varNoise *
meanWin));  // loizou comparative - eq. 6

        return (constK[toIndex(x, y)]);
    }

    private float getMean(int x, int y) {
        double sum = 0;
        double rep = 0;
        int pixel;
        int aux = ((WINDOW - 1) / 2);

        for (int i = x - aux; i <= x + aux; i++) {
            for (int j = y - aux; j <= y + aux; j++) {
                if ((i >= 0) && (j >= 0) && (i <= height - 1) && (j <=
width - 1)) {
                    pixel = (int) (original[toIndex(i, j)] & 0xff);
                    sum += pixel;
                    rep++;
                }
            }
        }
        return ((float) (sum / rep));
    }

    private float getVar(float mean, int x, int y) {
        double sum = 0;
        double rep = 0;
        int pixel;
        int aux = ((WINDOW - 1) / 2);

        for (int i = x - aux; i <= x + aux; i++) {
            for (int j = y - aux; j <= y + aux; j++) {
                if ((i >= 0) && (j >= 0) && (i <= height - 1) && (j <=
width - 1)) {
                    pixel = (int) (original[toIndex(i, j)] & 0xff);
                    sum += (pixel - mean) * (pixel - mean);
```

```
                    rep++;
                }
            }
        }
        return ((float) (sum / (rep/* - 1*/)));
    }

    /******************** AUXILIAR ********************/
    //2D Image
    private int toIndex(int x, int y) {
        return ((x*width) + y);
    }
}
```

## B.4 – Avaliador de Imagens

```
import java.io.File;
import java.util.*;

import ij.*;
import ij.gui.*;
import ij.plugin.*;
import ij.process.*;

/**
 * Batch evaluator
 * @author: Andre Fernando Lourenco de Souza
 * @version: 1.0
 * @created: Aug-2009
 * EPUSP - Polytechnic School of Sao Paulo - Brazil
 * LEB - Laboratorio de Engenharia Biomedica
 * andreflsouza@usp.br
*/
public class Plugin_Batch_Evaluation implements PlugIn {

	private static String title1 = "";
	private static String title2 = "";

	private static int backgroundImg = -1; //0xFF; //0;
	private static int backgroundPadrao = 0;//0xFF; //0;

	int width;
	int height;

	public void run(String arg) {
		String dir1 = IJ.getDirectory("Select image standards
folder...");
		if (dir1 == null)
			return;

		String dir2 = IJ.getDirectory("Select image segmented
folder...");
		if (dir2 == null)
			return;

		String[] list = new File(dir1).list();
		if (list == null)
			return;

		String segName = "";
		for (int i = 0; i < list.length; i++) {
			if (list[i].equals("Thumbs.db"))
				continue;

			System.out.println("Padrao #" + i + " nome= " +
dir1+list[i]);
			ImagePlus imgSeg = null;
			ImagePlus imgPadrao = IJ.openImage(dir1+list[i]);

			/* Compare standard with LSMV */
```

```
                segName = "s_l_n_" + list[i];
                System.out.println("Segmented LSMV #" + i + " nome= "
+ dir2+segName);
                try {
                      imgSeg = IJ.openImage(dir2+segName);
                } catch (Exception e) {
                      System.out.println("Erro! File not found: " +
dir2+segName);
                      continue;
                }
                evaluate(imgPadrao, imgSeg, segName);

                /* Compare standard with Chitwong */
                segName = "s_c_n_" + list[i];
                System.out.println("Segmented Chitwong #" + i + "
nome= " + dir2+segName);
                try {
                      imgSeg = IJ.openImage(dir2+segName);
                } catch (Exception e) {
                      System.out.println("Erro! File not found: " +
dir2+segName);
                      continue;
                }
                evaluate(imgPadrao, imgSeg, segName);

                /* Compare standard with LSMV + Chitwong */
                segName = "s_c_l_n_" + list[i];
                System.out.println("Segmented LSMV + Chitwong #" + i +
" nome= " + dir2+segName);
                try {
                      imgSeg = IJ.openImage(dir2+segName);
                } catch (Exception e) {
                      System.out.println("Erro! File not found: " +
dir2+segName);
                      continue;
                }
                evaluate(imgPadrao, imgSeg, segName);

                System.out.println("");
          }
     }

     /*
      * main function.
      */
     void evaluate(ImagePlus imgPadrao, ImagePlus img2, String label) {
          width  = imgPadrao.getWidth();
          height = imgPadrao.getHeight();

          if (img2.getWidth() != width || img2.getHeight() != height)
{
          IJ.showMessage("Evaluation", "The images must have the same
size.");
          return;
          }

          byte[] bytesPadrao =
(byte[])imgPadrao.getProcessor().getPixels();
```

```
            byte[] bytesImg = (byte[])img2.getProcessor().getPixels();

            /*
             * Do the matching
             */
            int cnt_total_obj = 0;
            int cnt_total_bkg = 0;
            int cnt_true_positive = 0;
            int cnt_false_positive = 0;
            int cnt_true_negative = 0;
            int cnt_false_negative = 0;
            for (int i = 0; i < width; i++) {
                  for (int j = 0; j < height; j++) {
                        if (bytesPadrao[toIndex2D(i, j)] ==
backgroundPadrao) {
                              cnt_total_bkg++;
                              if (bytesImg[toIndex2D(i, j)] ==
backgroundImg) {
                                    cnt_true_negative++;

                              } else {
                                    cnt_false_positive++;

                              }
                        } else {
                              cnt_total_obj++;
                              if (bytesImg[toIndex2D(i, j)] ==
backgroundImg) {
                                    cnt_false_negative++;

                              } else {
                                    cnt_true_positive++;

                              }
                        }
                  }
            }

            /*
             * Show the results.
             */
        System.out.println (
        /*showMessage("Results - Evaluation",*/
                        label +
                        " True-positive: " + cnt_true_positive +
                        ", False-positive: " + cnt_false_positive +
                        ", Pixels Object: " + cnt_total_obj +
                        ", Pixels Bkg: " + cnt_total_bkg +
                        ", True-positive %: " +
((float)((float)cnt_true_positive/(float)cnt_total_obj)*100) +
                        ", False-positive %: " +
((float)((float)cnt_false_positive/cnt_total_bkg)*100)
                        );

        IJ.log (label + ", " +
((float)((float)cnt_true_positive/(float)cnt_total_obj)*100) +
                              "%, " +
((float)((float)cnt_false_positive/cnt_total_bkg)*100) +
                              "%"
```

```
                                );
}

	/*
	 * Auxiliary methods
	 */
	private int toIndex2D(int x, int y) {
		/* If it is in the boundaries, use the bound value */
		if (x < 0)
			x = 0;
		else if (x >= width)
			x = width-1;

		if (y < 0)
			y = 0;
		else if (y >= height)
			y = height-1;

		return ((y*width) + x);
	}

}
```

## APÊNDICE C – CÓDIGO MATLAB DO GERADOR DE *SPECKLE*

```
function [imagem_speckle]=speckle(imagem_ideal)

% Definição de constantes
c=1540;                 %velocidade do som no tecido (1540m/s)
f=50E6;                 %frequência do ultrassom 10MHz
k=2*pi*f/c;
alfa=1.2023;            %constante de atenuação do sangue
                        %0,18dB/MHz/cm(1,0423) ou 0,8dB/Mhz/cm(1,2023)
alfaSI=alfa/1E4;        %mudança das grandezas para SI
u=alfaSI*f;

% Tamanho da imagem
x_conv=300;
y_conv=300;

for x=1:x_conv
    h1(x)=exp(-(x*1E-4*u)^2); % * sin(k*x*20E6);%o x deve estar
multiplicado por E-4 pois cada pixel equivale a 100um
end
for y=1:y_conv
    h2(y)=exp(-abs((150-y)*1E-4*u)^2); %o x deve estar multiplicado por
E-4 pois cada pixel equivale a 100um
end

[x_size,y_size]=size(imagem_ideal);
meio=[imagem_ideal(:,y_size-
y_conv/2+1:y_size),imagem_ideal(:,:),imagem_ideal(:,1:y_conv/2)]; %meio
físico no qual se realiza o ultrassom

meioGauss=imnoise(meio, 'gauss' , 0 , 0.05);%adição ruído gaussiano 0.05
V=conv2(h1',h2,meioGauss);%convolução Speckle

%corte das regiões que não interessam
[x_V_size,y_V_size]=size(V);
V(:,y_V_size-round(y_conv/2*2)+1:y_V_size)=[];
V(:,1:round(y_conv/2*2)-1)=[];
V(x_V_size-x_conv+2:x_V_size,:)=[];
%fim:corte das regiões que não interessam

Vh=hilbert(V);%transformada de Hilbert
Va=V+1i*Vh;
[x_size,y_size]=size(Va);
for x=1:x_size
    for y=1:y_size
        imagem_speckle(x,y)=abs(Va(x,y));
    end
end
```

## REFERÊNCIAS BIBLIOGRÁFICAS

[1] NOBLE. J. A.; BOUKERROUI, D. **Ultrasound Image Segmentation: A Survey.** IEEE Trans. Med. Im., Vol. 25, No. 8, p. 987 – 1010, 2006.

[2] UNAL, G.; BUCHER, S.; CARLIER, S.; SLABAUGH, G.; FANG, T.; TANAKA, K. **Shape-Driven Segmentation of the Arterial Wall in Intravascular Ultrasound Images.** IEEE Transactions on Information Technology in Biomedicine, VOL. 12, NO. 3, MAY 2008.

[3] YAN, P.; SINUSAS, A.; SUNCAN, J. S. **LV Segmentation from 3D echocardiography using fuzzy features and a multilevel FFD model.** IEEE Xplore, 2008.

[4] UDUPA, J. K.; SAHA, P. K.; LOTUFO, R. A. **Relative Fuzzy Connectedness and Object Definition: Theory, Algorithms, and Applications in Image Segmentation.** IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 24, No. 11, November 2002.

[5] UDUPA, J. K.; SAHA, P. K. **Fuzzy Connectedness and Image Segmentation.** IEEE, Procedings of IEEE, Vol. 91, pp. 1649-1669, October 2003.

[6] FALCAO, A. X.; STOLFI, J.; LOTUFO, R. A. **The image foresting transform: theory, algorithms, and applications.** Pattern Analysis and Machine Intelligence, IEEE Transactions on Volume 26, Issue 1, Page(s):19 – 29, 2004.

[7] LOIZOU, C. P.; PATTICHIS, C. S.; PANTZIARIS, M.; TYLLIS, T.; NICOLAIDES, A. **Quality Evaluation of Ultrasound Imaging in the Carotid Artery based on normalization and Speckle Reduction Filtering.** Medicine and Biology Engineering Computational, Vol. 44, p. 414 – 426, 2006.

[8] LOIZOU, C. P.; PATTICHIS, C. S.; CHRISTODOULOS, C. I. **Comparative Evaluation of Speckle Filtering in Ultrasound Imaging of the Carotid Artery.** IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control, Vol. 52, No. 10, October 2005.

[9] YUE, Y.; CLARK, J. W. Jr.; KHOURY, D. S. **Speckle Tracking in Intracardiac Echocardiography for the Assessment of Myocardial Deformation.** IEEE Transactions on Biomedical Engineering, Vol. 56, No. 2, FEBRUARY 2009.

[10] YONGJIAN, Y.; ACTON, S. T. **Speckle Reducing Anisotropic Diffusion.** IEEE Transactions on Image Processing", Vol. 11, No. 11, November 2002.

[11] CHITWONG, S.; CHEEVASUVIT, F.; DEJHAN, K.; MITATHA, S.; NOJYOO, C.; PAUNGMA, T. **Segmentation on Edge Preserving Smoothing Image based on Graph Theory.** Geoscience and Remote Sensing Symposium, Proceedings, IGARSS 2000, IEEE 2000 International Volume 2, 24-28 July 2000 Page(s):621 - 623 vol.2.

[12] KASS, M.; WITKIN, A.; TERZOPOULOS, D. **Snakes: Active contour models.** International Journal of Computer Vision, 2:321-331, 1988.

[13] XU, J. P. **Generalized Gradient Vector Flow External Forces for Active Contours.** Signal Processing 71, pp 131-139, 1998.

[14] OSHER, S; FEDKIW, R. **Level Set Methods and Dynamic Implicit Surfaces.** Springer-Verlag, New York, 2002.

[15] ROERDINK, J. B. T. M.; MEIJSTER, A. **The Watershed Transform: Definitions, Algorithms and Parallelization Strategies.** IOS Press, Fundamenta Informaticae 41, 187{228, 2001.

[16] AUDIGIER, **R.;** LOTUFO, R. **Duality between the Watershed by Image Foresting Transform and the Fuzzy Connectedness Segmentation Approaches.** XIX Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAPI'06), 2006.

[17] PEDNEKAR, A. S.; KAKADIARIS, I. A. **Image Segmentation Based on Fuzzy Connectedness using Dynamic Weights**, 2006.

[18] YOO, T. S. **Insight into Images.** ITK, AK Peters, 2004.

[19] OTSU, N. **A Threshold Selection Method from Gray-Level Histograms.** IEEE Trans. Systems, Vol. SMC-9, No. 8, p.62, 1979.

[20] GONZALEZ R.; WOODS R. **Digital image processing.** 2nd edn. Prentice-Hall, Englewood Cliffs, pp 419–420, 2002.

[21] XU, M. H.; LIU, Y. Q.; HUANG, Q. L.; ZHANG, G. F. **An improved Dijkstra's shortest path algorithm for sparse network.** 2007.

[22] UDUPA, J. K. **Methodology for Evaluating Image Segmentation Algorithms.** 2002.

[23] SOUZA, A. F. L.; FURUIE, S. S. **Investigação de filtros adaptativos em imagens ecocardiográficas para segmentação de estruturas.** São Paulo: VI Simpósio de Instrumentação e Imagens Médicas, 2009.

[24] SOUZA, A. F. L.; LAGE, D. M.; FURUIE, S. S. **Coronary Segmentation from Echocardiography using Fuzzy Connectedness.** Rio de Janeiro: Sibgrapi, 2009.

[25] IMAGEJ website. **http://rsbweb.nih.gov/ij/.** Data acesso: 21 de agosto de 2010.