INSTITUTO TECNOLÓGICO DE AERONÁUTICA

Daniel Lélis Baggio

Intravascular Ultra Sound Image Segmentation Algorithms

Trabalho de Graduação
2006

Computação

621.397.3:616.12

Daniel Lélis Baggio

Intravascular Ultra Sound Image Segmentation Algorithms

Advisor
Prof. Dr. Nei Soma (ITA)

Co-advisor
Prof. Dr. Sergio Furuie (InCor)

Department of Computer Science

São José dos Campos
Centro Técnico Aeroespacial
Instituto Tecnológico de Aeronáutica

2006

Cataloging-in Publication Data
Division of Central Library of ITA/CTA

Baggio, Daniel Lélis
Intravascular Ultra Sound Image Segmentation Algorithms / Daniel Lélis Baggio.
São José dos Campos, 2006.
75f.

Undergraduate Final Project- Division of Computer Science – Aeronautical Institute of Technology, 2006. Advisor: Prof. Dr. Nei Soma. Co-advisor: Prof. Dr. Sergio Furuie.

1. Segmentation. 2. Livewire. 3. Active Contour. I. Aerospace Technical Center. Aeronautics Institute of Technology. Division of Computer Science. II. Title.

BIBLIOGRAPHIC REFERENCE

BAGGIO, Daniel Lélis. **Intravascular Ultra Sound Image Segmentation Algorithms**. 2006. 75f. Undergraduate Final Project – Technological Institute of Aeronautics, São José dos Campos.

CESSION OF RIGHTS

AUTHOR NAME: Daniel Lélis Baggio
PUBLICATION TITLE: Intravascular Ultra Sound Image Segmentation Algorithms.
TYPE OF PUBLICATION/YEAR: Trabalho / 2006

Daniel Lélis Baggio
Rua Lima Machado, 239
CEP 11.310-310 – São Vicente–SP

I dedicate this thesis to God and to my beloved grandmother, Anália Lélis da Silva, who's now resting in peace. She was the woman who taught me how to pray, the happy voice that answered me on the phone, the lasting hug when I would say good bye. I won't cry for missing her. Instead, I'll let her warm my heart with the love that comes from her prayers, which are now echoing from heaven.

Acknowledgments

I'd firstly like to thank my parents, Jorge Francisco Baggio and Andrea Beatriz Lélis da Silva Baggio, for being present in my whole life and for all support, advices, principles and love they've given me. I'd also like to thank my blood brother, Rafael Lélis Baggio, for all the laughs, knowledge, company and fun we've had in all these years. A special thanks for my aunt Sônia Lélis da Silva, for the great support she's given me, particularly in high-school. Another heart thank for my grandmother Anália Lélis da Silva, for all her prayers and love throughout my life. A big thank for my whole family.

I'd also like to thank all of my roomates, from 210 and the ones from 238. We had quite a great time together during these five years of battles. A special thank for Daniel Motta – and family – for being such a great friend when I needed and Humberto Silva Naves, for his crazy way of facing life, all the knowledge and friendship. I'd also like to thank all friends from Comp, with whom we've shared many experiences. I'd also thank *Naves Bros.* and *Kong Bros Jr.* for all the fun, challenges and knowledge gathered, specially for all teachings given by Carlos Stein.

To Marcos Botelho, my advisor throughout these years, for all the advices and great time we had. To IBM, for all the challenges and oportunities, as well as all the great persons I've met there, specially Katia Zaneti, who faced the military constraints, Mauro Assano and Fabio Gandour.

To all students from CASD for such a hunger for knowledge and all special friends I've made there.

To Tim McInerney for the JESS system as well as all explanations given through the work.

To Wayne Rasband, for ImageJ and all the support given, as well as to everybody in ImageJ mailing list.

A big thank to *Cida*, my English teacher for 7 long years as well as *D. Lenita*, my Portuguese teacher during 4 important years of my life.

To Instituto Tecnológico de Aeronáutica and Prof. Nei Soma, for all the advices and support given. To Instituto do Coração, Dr. Sérgio Furuie, Fernando Sales, Mônica Matsumoto and Vivian Jonke without whom this work would have never been accomplished. A special thanks to everybody from the Research and Development unit from InCor.

And at last, but not least, to God, for all I am.

"Ubuntu, or,
I am what I am because of who we all are"
— humanist ideology from Africa
and also a Linux-based operating system

Resumo

Esta tese de graduação trata da segmentação de imagens de ultra-som intravascular através dos algoritmos *LiveWire* e *Snakes*. Como o tipo de ultra-som intravascular utilizado é o de coronárias, há um detalhamento inicial do contexto médico no qual o problema está inserido.

Explica-se, então, como funciona o método *LiveWire*, através do qual o caminho de menor custo entre dois pixels da imagem é calculado. Discute-se uma implementação utilizando-se árvores binárias e a estrutura heap. Esta discussão é seguida pela avaliação dos custos a serem adotados na aresta, entre eles o módulo e a direção do gradiente, o laplaciano da imagem e uma função não-linear.

Aborda-se, então, o algoritmo *Snakes*, o qual simula um conjunto de vértices submetido a dois campos de energia, um interno e outro externo. Há também uma discussão sobre a quantidade de iterações que o algoritmo deve executar, bem como os valores utilizados para cada um dos parâmetros.

Discute-se, então, como imagens simuladas, utilizadas para validação, foram geradas. Além disso, descreve-se o tipo de modelo para ruído Speckle usado, bem como o filtro ao qual as imagens foram submetidas.

Há uma breve discussão de como os algoritmos foram implementados, particularmente discutindo a plataforma ImageJ e seus artifícios de extensibilidade.

Segue-se, então, a descrição dos resultados, com cada um dos tipos simulados de imagem: sem ruído, com ruído Speckle e com ruído Speckle filtrado. Os principais resultados encontrados foram que *LiveWire* tem uma qualidade de segmentação melhor e praticamente não é afetado por ruído, enquanto a técnica de *Snakes* é mais rápida e fracamente afetada por pontos iniciais posicionados incorretamente a partir da segmentação longitudinal.

Ocorre então o desfecho, com as conclusões encontradas, destacando-se a velocidade com que os métodos baseados em *Snakes* são executados, seguidos de seus bons resultados. Foram encontrados melhores resultados com a técnica *LiveWire*, no entanto, o tempo de processamento foi muito maior, o que pode indicar que o uso de tal método não seja desejável. Devido aos bons resultados com imagens simuladas, a plataforma desenvolvida mostra-se então de promissor uso no meio médico, caso seja submetida à validação por especialistas.

Abstract

This graduation thesis deals with intravascular ultrasound (IVUS) image segmentation through *LiveWire* and *Snakes* algorithms. As the type of ultrasound studied is the one from coronaries, there's an initial detailing of the medical environment in which the problem is inserted.

The introduction is followed by an explanation of how *LiveWire* technique works, which is by calculating the shortest cost path between two pixels of the image. There's a discussion about the implementation, using binary trees and a heap data structure. This is followed by the evaluation of which costs should be adopted, which were: modulus and direction of the gradient, the laplacian and a non-linear function.

Then, *Snakes* algorithm is described, which simulates a set of vertices submitted to one internal field of energy and another external one. There's also a discussion about the number of iterations to be adopted as well as the values used as parameters.

It is explained, so, how simulated images, used for evaluation, were generated. Besides, a model for *Speckle* noise is described as well as the filter to which the images were submitted.

There's a brief discussion on how the algorithms were implemented, with a special focus on *ImageJ* platform and it's extensibility features.

The results are, then, described, with each of the simulated image types: noiseless, with *Speckle* noise and filtered. The main results found are that *LiveWire* has better segmentation quality and is slightly affected by noise, while *Snakes* is faster and is practically not affected by wrong initial points retrieved from longitudinal segmentation.

There's, then, the conclusion, describing what was found, focusing the high velocity of *Snakes* technique, followed by good results. Although better results were found

for *LiveWire*, the processing time was about 500 times higher, what might indicate this method is not desirable. Due to the good results found with simulated images, the developed platform promises to work efficiently in a medical environment, after submitted to validation by specialists.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation and background

1.1.1 Image Segmentation

Throughout the history of science and engineering, image data has taken a high degree of importance, aiding scientists from several fields of knowledge to capture details previously unavailable. In medical imaging, for instance, tomographic scanners routinely produce two-, three-, and four-dimensional digital image data, showing human internal structures. It also helps when some physiologic function from the system structure is being studied. There are some regions of interest in these images – for example, the anatomical organs – whose properties (shape, dimension and functional process) need to be visualized and analyzed (3D..., 1991) to fulfill the main objectives of imaging the phenomenon. The most fundamental requirements for successful visualization and analysis of such regions of interest is a method for the repeatable, accurate, and efficient extraction of object information from given multidimensional images. This process is commonly referred to as *Image Segmentation* (FALCAO *et al.*, 1998).

Using segmentation may help a physician to identify important details of images. In figure 1.1, the lumen region of an intravascular ultrasound image is segmented. Lots of properties may be retrieved from it, like the lumen's area, its shape and pixel colors, which might be very useful for diagnostics.

Medical image segmentation can be accomplished in a number of ways (LOBREGT; VIERGEVER, 1995):

1. *Completely Manual*: An operator will need to sit in front of the screen and use a pointing device like a mouse or graphics tablet to segment the region of interest.

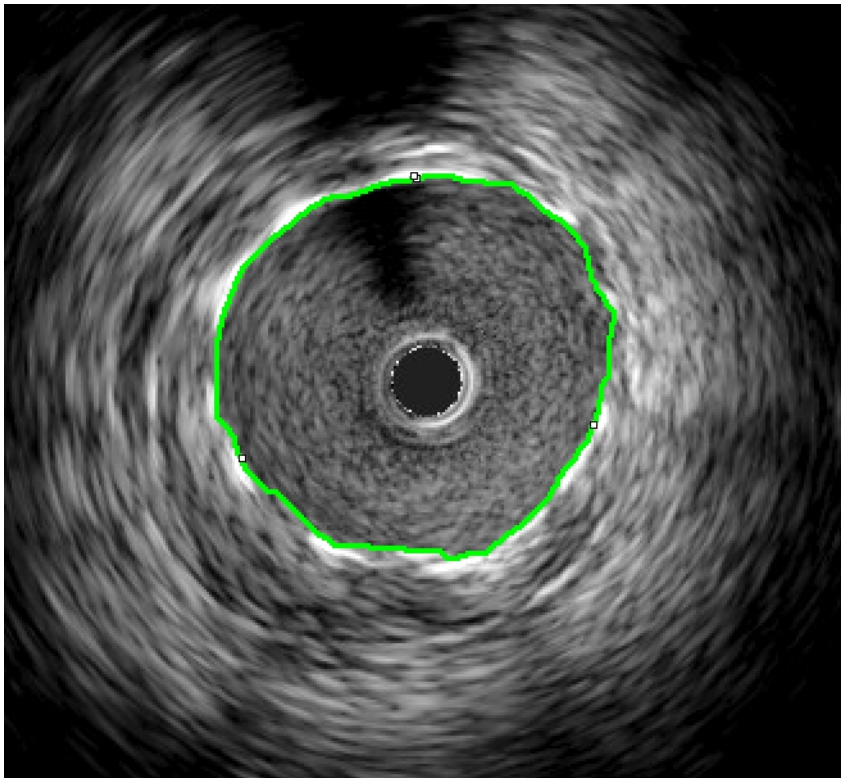


FIGURE 1.1 – Example of IVUS image segmentation

This requires a considerable amount of time as well as some expert knowledge of the operator about the clinical problem and some skill to use the devices. The main drawbacks of manual segmentation are the long time consuming procedure, specially for numerous frames, as happens in 3D slices, and that it suffers from a very low degree of reproducibility (EILBERT; MCEACHRON, 1990).

2. *Fully Automatic*: There are no sophisticated enough techniques available for many typical applications. The use of these techniques is therefore restricted to simple contours of specific objects, like endocardial delimitation of the left ventricle.

3. *Automated First Guess, Followed by Manual Editing*: The first guess generally uses simple techniques, like thresholding or region growing. In the manual phase, the operator uses contextual knowledge about anatomy and pathology to modify the generated contour. Besides being very time consuming, this approach leads to results that are not reproducible.

4. *Manual Rough Delimitation, Followed by Automated Contour Definition*: The user interaction is now limited since only a rough delineation, or some points around the contour, are required. The following automatic contour definition process refines the

countour, possibly based on user configured parameters. Some variation in initial contours, for instance, drawn by different operators, will still allow a result with very few differences, leading to a high degree of reproducibility.

For such desirable features described in 4, this thesis follows the same approach.

Some rigorous evaluation of computer medical imaging processing has revealed excellent perspectives for the future (ZONNEVELD; FUKUTA, 1994), but image segmentation has been pointed as one of the main problems in medical clinical applications for its requirement of a specialist, at least during verification of segmentation results. Although automatic methods have been used in several specific applications, in a dedicated way (CAN *et al.*, 1999) (COHEN; TURNER, 1994) (GERIG *et al.*, 1991), they fail when presented to new situations, due to different protocol acquisition, modalities or region of interest types. In such situations, interactive methods are used.

The process of segmentation may be divided in two parts (FALCÃO, 1997): *recognition* and *delineation*. Recognizing the region of interest consists in a rough determination of its place and differing it from the similar entities in the image. Delineating the object consists in defining precisely the extension of the object in the image. Human operators are generally capable of recognizing the object better than computers. The main difficult for this, is that it's rather difficult to make the global knowledge about the region of interest in computable local operations. It has been one of the major problems of automatic segmentation techniques. On the other hand, computers accomplish the task of delineation better than humans (more repetitive, faster and more exact). An efficient strategy is, therefore, explore the synergy between computers and algorithms during the segmentation. The method described in this thesis takes full use of this approach.

1.1.2 Intravascular Ultrasound

The main application for this thesis focus segmentation of coronary intravascular ultrasound images, which might help assess coronary atherosclerosis, one of the main causes of morbi-mortality in Brazil and in the world (SALES, 2006). In its critical state, the atherosclerotic plaque determines the narrowing of lumen resulting in angina[1] symptoms.

[1] cardiac chest pain

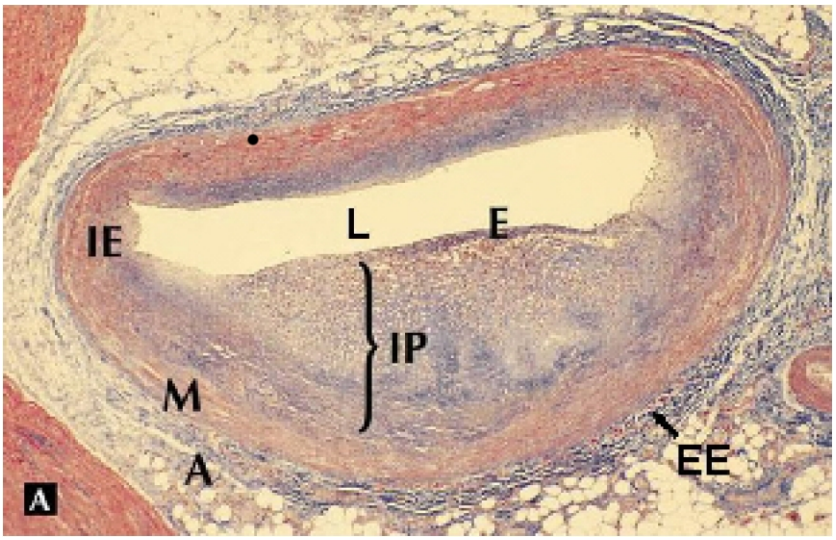


FIGURE 1.2 – Histological slice of coronary artery

In some individuals, atherosclerotic plaque might form coronary thrombosis [2] of a coronary artery, causing myocardial infarction (a heart attack). In this context, it is extremely desirable that academic studies detect which are the factors related to the causes of a chronic, stable plaque becoming a region subjected to coronary thrombosis.

Atherosclerosis – hardening of the artery – results from a deposition of tough, rigid collagen inside the vessel wall generating the necessity of observing it. Histological studies shows that the atheromatous plaque is developed inside the intima – a region of the vessel wall located between the endothelium, the media and adventitia – as shown in figure 1.2.

Region L is called lumen and the blood flows through it. The narrow cell layer that surrounds the lumen is the internal elastic membrane, which has a thicker cell layer around it, called intima. As stated above, this is where the atheromatous plaque grows, causing expressive increase in local volume. The same figure 1.2 shows clearly the intimal plaque (IP). Intima is involved externally by the media (M), which is the cell layer highlighted in red after treatment in the histological slice. The adventitia (A) is the external artery region, and it is separated from media by the presence of the external elastic membrane (EE), represented by the dark blue cell layer in figure 1.2. As such histological slices can not be made in live beings it is important to have a tool that allows seeing the arterial wall *in vivo*.

It is also stated in (SALES, 2006) that coronary angiography is the gold-standard

[2] the formation of a blood coagulation inside a blood vessel, obstructing the flow of blood through the circulatory system

procedure for coronary disease diagnosis. However, many reports have shown that there is frequently a difference between the real atherosclerotic damage and the corresponding found in angiography (MINTZ *et al.*, 1995), (BRIGUORI *et al.*, 2002). Arterial segments with up to 40–50% of its area occupied by atherosclerotic plaque keep its lumen dimensions relatively unchanged, what is a result from the artery "positive" (or "adaptive") remodeling (MINTZ *et al.*, 1995), (BRIGUORI *et al.*, 2002). It is noticed, therefore, that angiography presents limitations as a method of evaluating not only the volume but also morphological and composition features of the atherosclerotic plaque.

In order to solve this problem, some invasive assessment methods capable of quantitatively and qualitatively evaluating the atherosclerotic plaque have been introduced recently. Intravascular ultrasonography has been the chosen procedure to visualize coronary vascular wall *in vivo* (NISSEN; YOCK, 2001). Intravascular ultrasound allows evaluation and computation of the arterial dimensions, plaque volume and the presence of arterial remodeling, even in patients without lumen narrowing (MINTZ *et al.*, 1995). Evidences propose that greater atherosclerosis load are possibly linked to a high risk of coronary diseases. Equally, the presence of positive remodeling and bigger plaque area have been related more frequently in patients with acute coronary syndromes than in those with stable angina (HONG *et al.*, 2004) or in related plaques compared to non-related ones in patients with acute infarction (EHARA *et al.*, 2004).

According to (SALES, 2006), intra-coronary ultrasonography gives the hemodynamicist information about the real situation of the coronary network that were unavailable in coronary cineangiography (ABIZAID *et al.*, 1997). Based in the quality of such exam, there has been the emergence of a new doubt, whether making the intracoronary ultrasound was linked to an increase in the possibility of the patient developing an expressive cardiovascular event. At least two essays have been made in Europe to answer that doubt (LINKER; BATKOFF, 1996), (HAUSMANN *et al.*, 1995). The one that has made more exams of the two (HAUSMANN *et al.*, 1995), gathered 28 centers, totalling 2207 exams. From these, 2034 patients (92.16%) haven't showed up any event, 87 (3.9%) have presented events not related to the intracoronary ultrasound, 63 (2.9%) have presented coronary spasm [3], 9 (0.4%) have had complications related to the exam and in about 14 (0.6%), it

[3]Coronary artery spasm is a temporary, abrupt, and focal (restricted to one location) contraction of the muscles in the wall of an artery in the heart, which constricts the artery

is not known if the events were related to ultrasonography. These studies emphasized the security of this procedure, despite its invasive nature.

Ultrasound image formation is achieved through appropriate processing that takes place at one or more of a number of the following levels (BAMBER, 2002):

- Transmit pulse wave specification
- Scattering of sound by tissue
- Receive channel processing
- Beam forming
- Single line RF processing
- Multi-line RF processing
- Pre-processing
- Scan conversion
- Post-processing
- Image Display

Figure 1.3 shows the process of obtaining the image from ultrasound (SALES, 2006). The ultrasonic transducer [4] emits ultrasound pulses – generally between 1 and 40 MHz – which find a variety of structures on the arterial wall, suffering multiple reflections. The returning signal is then transformed in electrical signals – in the same process that it took to send the wave, but in reverse, since the transducer also converts sound into electrical energy. These signals are then processed before becoming an image. Firstly, they are digitalized and a series of filters are applied to them, finishing the procedure with some amplification to counterbalance the energy loss caused by the environment attenuation.

After the pre-processing phase, a window from the pulse's envelop is extracted. Its width is related to the maximum radial distance to be mapped, given that the velocity of radiation propagation is in a known range. Parts of the wave that were reflected in

[4] piezoelectric device that converts electrical energy into sound

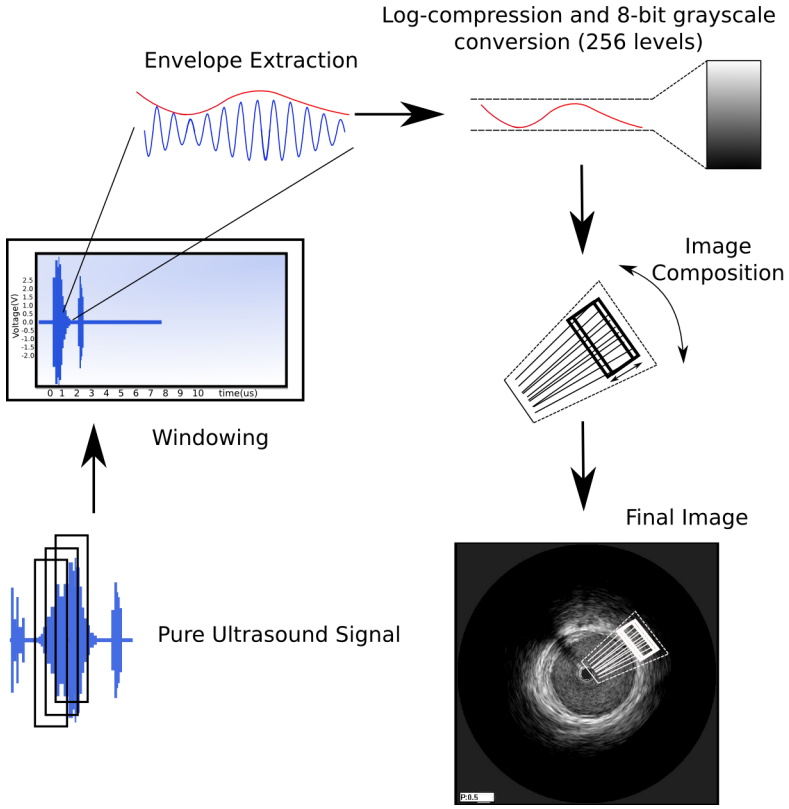


FIGURE 1.3 – Process of intravascular ultrasound image formation

smaller time delays correspond to arterial points that are nearer the transductor, while the ones that take more time to reflect are farther. The intensity of the reflected wave will correspond to the pixel's intensity, however, the range of values received by the transducer are in a wide voltage range, that goes from hundreds of micro volts up to unities of volts. In order to reduce the dynamic range of the envelope signal and enhance the comparatively weak backscattered echoes, logarithmic compression is routinely used in existing ultrasonic scanner (TSUI; WANG; HUANG, 2005). In this type of compression, the signal's envelope is submitted to a logarithmic function, limiting the amplitude range.

Following the logarithmic compression, the signal is discretized, nowadays, in 256 values, enough to be stored in 1 byte. One array of bytes is generated for each angular position of the transducer, from the center to the border of the radius. The same procedure is repeated for all of the desired directions in the plane, until a whole rotations is completed. According to (SALES, 2006) this type of sweeping is highly subjected to the type of catheter used, whether mechanic or electronic. For the first type, there's an engine attached to it, with a 1800 rotations per minute frequency, while the other counts on 64 transducer symmetrically positioned radially around the catheter.

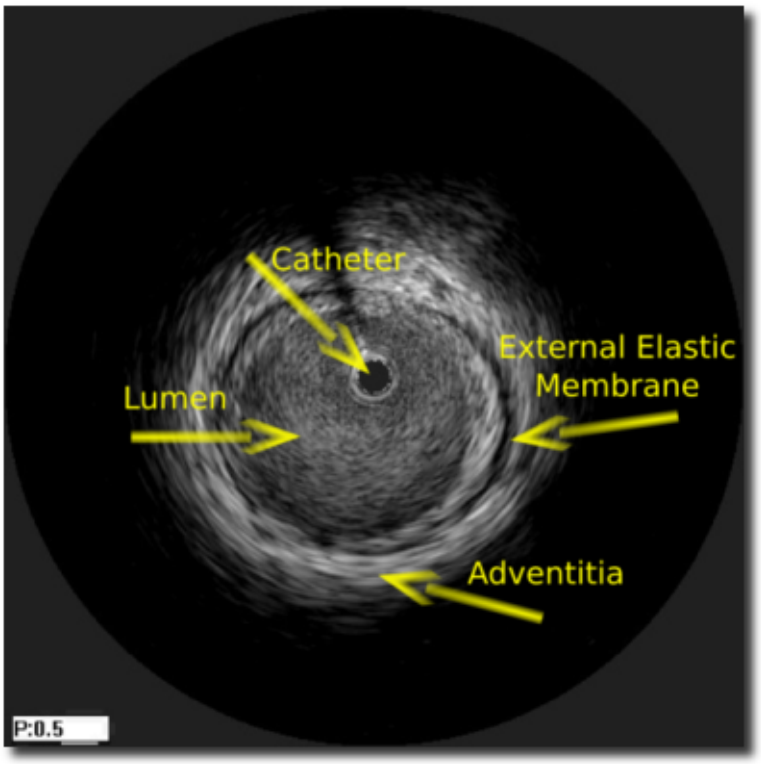


FIGURE 1.4 – Transversal coronary slice obtained from an intravascular ultrasound exam and its details

After composing the image, several details can be observed, as in figure 1.4. The center region – the catheter – as well as the lumen and adventitia regions are visibly segregated for a human observer. Although quite an informative image is formed, much of the details cannot be precisely measured until a proper segmentation is done, what is the main purpose of this thesis.

1.2 Goals

The goal of this thesis is to make the adventitia segmentation in three-dimensional coronary intravascular ultrasound simulated images, using *LiveWire* and *Snakes* techniques, in such way that both techniques can be compared, detecting the best features of each one of them.

2 Methodology

2.1 Live-Wire

Live-wire (MORTENSEN; BARRETT, 1995),(FALCAO *et al.*, 1998) is an image segmentation technique which allows regions of interest to be extracted quickly and accurately, using simple mouse clicks. It is based on the lowest cost path algorithm, by Dijkstra (DIJKSTRA, 1959) and may be applied to color or grayscale images. In case it is a grayscale one, the image is generally modeled as a rectangular matrix whose pixel values are integers ranging from 0 to 255. Each pixel of the matrix is a vertex of the graph – as seen in figure 2.1 – and has edges going to the 8 pixels around it: up, down, left, right, upper right, upper left, lower right and lower left. The edge costs are defined based on a cost function (MORTENSEN; BARRETT, 1995) and explained in more detail in section 2.1.2.

The user sets the starting point clicking on any image's pixel. Then, as he starts to move the mouse over other points, the smallest cost path is drawn from the starting point to the pixel where the mouse is over, changing itself when he moves away. If he wants to choose the path that is being displayed, he simply clicks the image again. The operator does it until the whole selection is made, finishing it through a right click, for instance.

2.1.1 Dijkstra's Algorithm Implementation

As the segmentation is a type of user interaction, it must be done as fast as possible. So, some improvements to the original $O(N^2)$ (DIJKSTRA, 1959) [1] implementation were added – in order to better understand the Big-O notation, one might look at appendix A. It must be noticed that, in this study, most of the pictures used for segmentation range from 400x400 pixels to 1,024x768, which equals to 160,000 and 786,432 pixels, almost one

[1] where N is the number of pixels

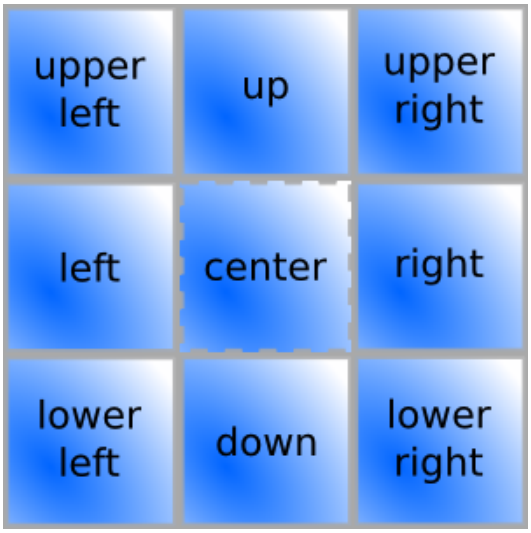


FIGURE 2.1 – 8 pixels around the center to be expanded

million pixels. As the segmentation process is supposed to have some real time response of 1 or 2 seconds, the $O(N^2)$ implementation is not desirable. Therefore, a faster approach has been studied.

Since the image is represented by a matrix, the pixel's neighbors can be found in $O(1)$ time. Then, the costs of going from one pixel to another must be calculated. This process depends highly on what sort of edge costs were chosen, so it will be discussed in subsection 2.1.2.

After this phase, it's needed to store the neighbor nodes as well as their path's cost, so that the one with lowest cost may be added to the final path. Some structures have been studied to perform this purpose. The one used in (MORTENSEN; BARRETT, 1998) is a bucket array, which performs Dijkstra's algorithm in $O(E + VC)$ (GOLDBERG; SILVERSTEIN, 1995) where the arc lengths are integers in the interval $[0, ..., C], C \geq 2$, E is the number of edges and V is the number of nodes in the network. The drawbacks of this approach, besides the necessity of using integer values, is that, if floating points are converted to integers, it is needed to multiply them by 100, or 1000 and truncate the values, which would result a high value of C, therefore, increasing the number of operations needed. Another structure studied, was the binary heap, which performs Dijkstra's algorithm in $O(Elog(V))$ (BARBEHENN, 1998). Since $E = 4V$ in the graphs used in this study, and V is at most 1,000,000 ($log_2(V) \approx 20$), this structure performs better for these types of graphs, hence, it was chosen. Another property of binary heaps that is desirable, is the fact that it does not require the edge costs to be integers.

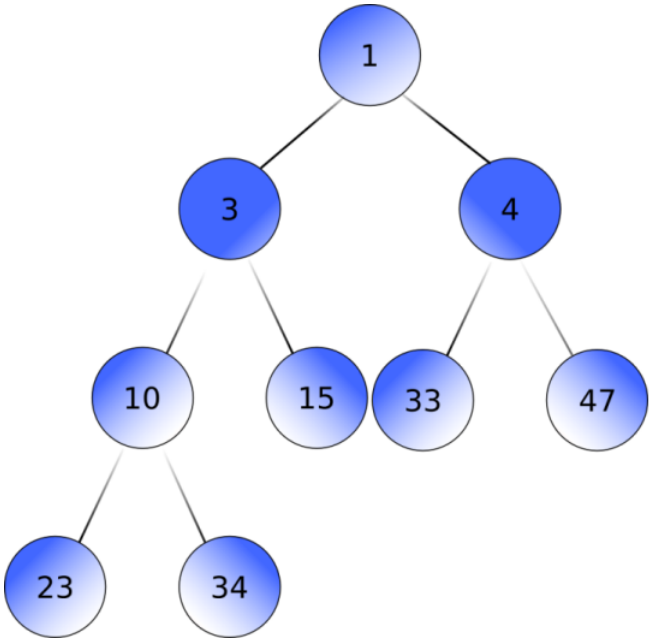


FIGURE 2.2 – Binary heap example

The binary heap is basically a tree which holds the lowest or highest element on its root, taking $O(1)$ time to be retrieved, a feature that is highly desirable. The disadvantage of this structure is that it takes $O(logN)$ to insert or delete elements, but it certainly decreases the Dijkstra's $O(V^2)$ time to $O(Elog(V))$ where V is the image's pixel number.

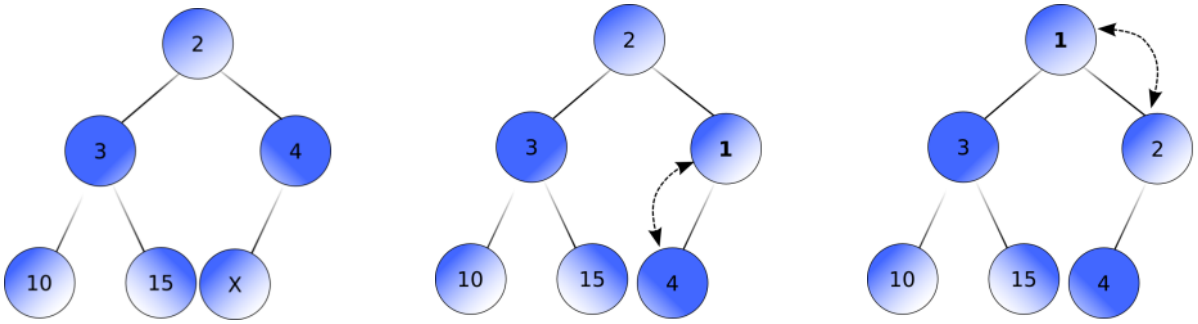
The original Dijkstra's algorithm pseudocode is best explained in (MORTENSEN; BARRETT, 1998), but the one used in this implementation is explained in section 2.1.3. Details of the heap structure used are in section 2.1.1.1.

2.1.1.1 Binary heap

Priority queues (INTRODUCTION..., 2001) are abstract data types that support three main operations:

- add an element to the queue with an associated priority;
- remove the element from the queue that has the highest priority, and return it;
- peek the element with the highest priority without removing it.

These operations are extremely useful for storing the node's total cost values when applying Dijkstra's algorithm.



(a) Add **1** to postion 'X' (b) Swap **1** and **4** (c) Swap **1** and **2**

FIGURE 2.3 – Add operation in a binary min-heap

Binary heaps are created through a binary tree, which is, in computer science, a tree data structure whose nodes have at most two children. They are typically called *left* and *right*. Binary heaps add two constraints to binary trees:

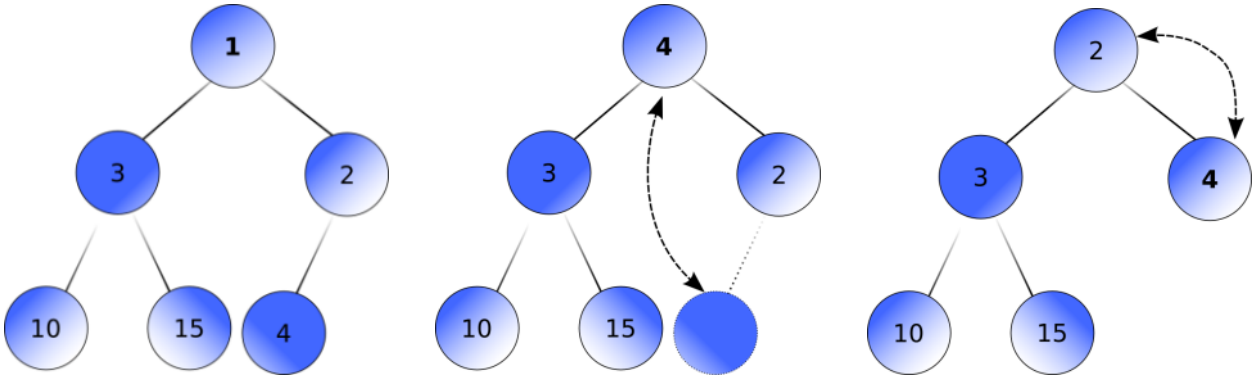
- *Shape property*: the tree is either a perfectly balanced tree (all leaves are at the same level), or, if the last level of the tree is not complete, the nodes are filled from left to right.
- *Heap property*: each node is *greater than* or equal to each of its children according to some comparison function, fixed for the entire data structure.

The *"greater than"* in *heap property* not necessarily refers to *"greater than"* in mathematical sense, but if it is used, the heaps are called *max-heaps*. As the comparison function used in Dijkstra's implementation is *"less than"*, the heap is called *min-heap* and they are readily applicable for priority queues. Figure 2.2 shows one example of such structure.

In order to add an element to the heap, so that the two properties listed above keep true, the following algorithm is applied and it takes $O(logN)$ time:

```
1. Add the element to the bottom level of the heap
2. Compare the element with its parent until it is correct
3. If the order is not correct swap the element with its
parent and return to previous step
```

Figure 2.3 shows how the element **1** would be added to a binary min-heap using the above procedure. Note that the maximum number of operations is log_2N – where N is



(a) Original tree (b) Replace root with the last element (c) Swap until valid

FIGURE 2.4 – Removing the root from a binary min-heap

the number of elements in the heap – because when a binary tree has N elements, it will have, at most, $log_2 N + 1$ levels.

Another important operation is deleting the root of the heap. This operation also takes $O(logN)$ time and is explained in figure 2.4. Firstly, the last element takes place of the root, being switched until there is no violation of the *heap property*. Notice that the minimum element is always in the root of the tree, hence, it can be accessed simply reading the tree's root in $O(1)$.

It must be observed that Live-Wire stores the cumulative cost as the value for sorting the binary heap, and it must also get track of what is the node's index in the image and whether it was used or not. All these properties are stored in a class named *PixelNode*. The implementation was made using Java's Priority Queue class (JAVA..., 2004). It is stated in the documentation that the class provides $O(logN)$ time for insertion and remove operations, while it takes constant time for peeking the root.

2.1.2 Edge Costs

A key feature for the Live-Wire to work properly is that the definition of the costs of going from one pixel to another is related to the type of image being used. Several articles (FALCÃO *et al.*, 1998), (MORTENSEN; BARRETT, 1995) describe ways to add costs to the edges. The ones chosen to be implemented in this thesis were the Laplacian Zero-Crossing (f_Z), Gradient Magnitude (f_G) and Gradient Direction (f_D). Another cost used, which is not described at (MORTENSEN; BARRETT, 1995) is the Exponential Cost (f_E). The

general formule for calculating the edge cost of going from node p to node q is defined in equation 2.1. All these costs will be described in detail in the following subsections.

$$l(\mathbf{p},\mathbf{q}) = \omega_Z \cdot f_Z(\mathbf{p},\mathbf{q}) + \omega_G \cdot f_G(\mathbf{p},\mathbf{q}) + \omega_D \cdot f_D(\mathbf{p},\mathbf{q}) + \omega_E \cdot f_E(\mathbf{p},\mathbf{q}) \tag{2.1}$$

Note that $\omega_Z, \omega_G, \omega_D$ and ω_E are the respective weights for each of the cost features.

2.1.2.1 Gradient Magnitude (f_G)

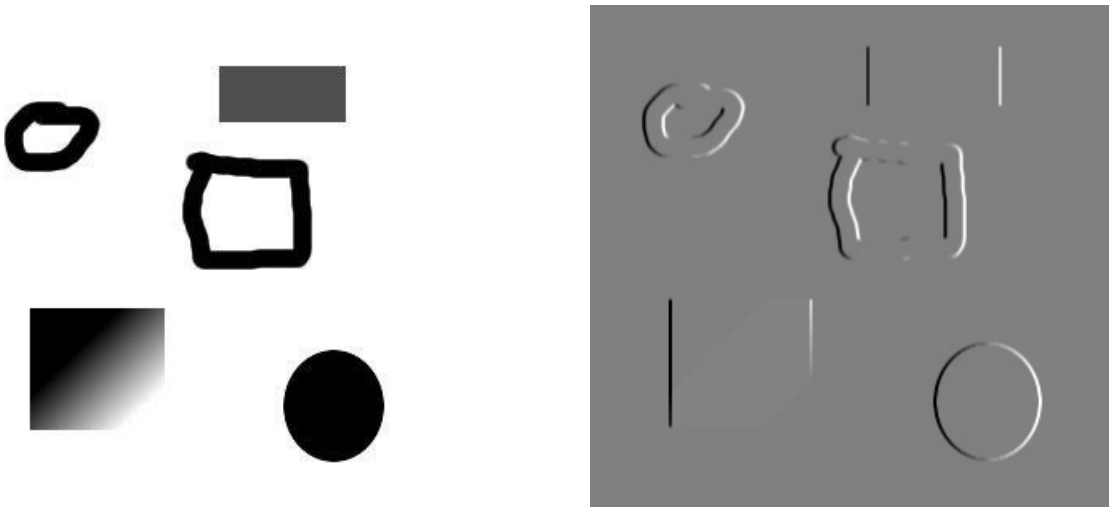
Since this section's cost and following's deal with gradient, a better explanation is required. As it is known from vector calculus, the gradient of a *scalar field* is a *vector field* which points in the direction of the greatest rate of increase of that field and whose magnitude is the greatest rate of change. One might ask himself: *"Where is the scalar field in an image?"* Since the algorithm is applied to grayscale images, the domain of the field is the two-dimensional discrete space formed by the matrix that represents the pixels of the image and the codomain of this function are the 8-bit values assumed to represent the pixels.

Mathematically, the gradient is represented by a column vector whose components are the partial derivatives of f(where f is a scalar field) as in equation 2.2. It must be noticed that, in this study, the slices of the images are two-dimensional, hence, the gradient has only two rows.

$$\bigtriangledown f = \left(\frac{\partial f}{\partial x_1}, ..., \frac{\partial f}{\partial x_n}\right)^T \tag{2.2}$$

In order to define partial derivatives in a discrete scalar field, the Sobel operator is used(ABDOU; PRATT, 1979). It is technically a discrete differentiation operator, which computes the gradient's approximation of the image's intensity function. To calculate the gradient in each point, two 3x3 kernels are convolved with it, one to detect horizontal changes and one for vertical.

Consider two images G_x and G_y that will contain, respectively, the horizontal and vertical derivative approximations at each point. The Sobel operator is, consequently



(a) Original image (b) G_x from Sobel convolution

FIGURE 2.5 – Vertical edge detection applying Sobel

defined as the following convolutions with the target image T:

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * T \qquad \text{and} \qquad G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * T \tag{2.3}$$

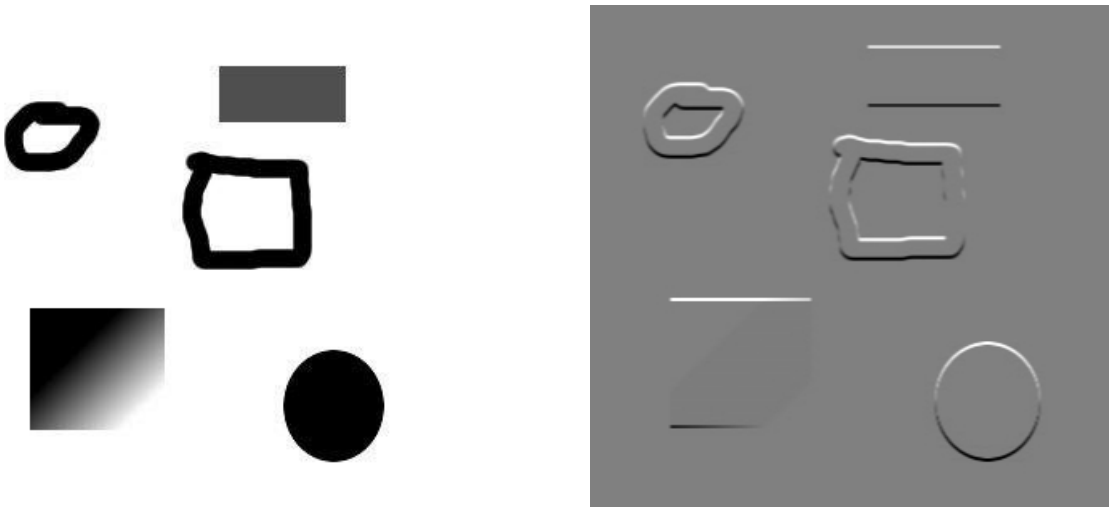
In order to obtain the gradient magnitude at each point of the image, the modulus of the resulting vector of G_x and G_y is taken:

$$G = \sqrt{G_x^2 + G_y^2} \tag{2.4}$$

The same vector approach is taken to obtain the Gradient Direction (θ):

$$\theta = \arctan\left(\frac{G_y}{G_x}\right) \tag{2.5}$$

It should be noticed that the convolution that generates G_x will only detect vertical edges, and, on the other hand, the one with G_y will only detect horizontal edges. To make it clearer, figure 2.5b shows the resulting convolution of the original image 2.5a, generating G_x. One should notice that what might look like an horizontal border on the lower left rectangle is due to the color gradient that was applied to it, which actually generates vertical borders. During the process, negative values may be found so, to display the



(a) Original image (b) G_y from Sobel convolution

FIGURE 2.6 – Horizontal edge detection applying Sobel

picture correctly, the values of G_x are shifted according to the following equation:

$$255 \cdot \left(\frac{G_{xi} - G_{xMAX}}{G_{xMAX} - G_{xMIN}} \right) \tag{2.6}$$

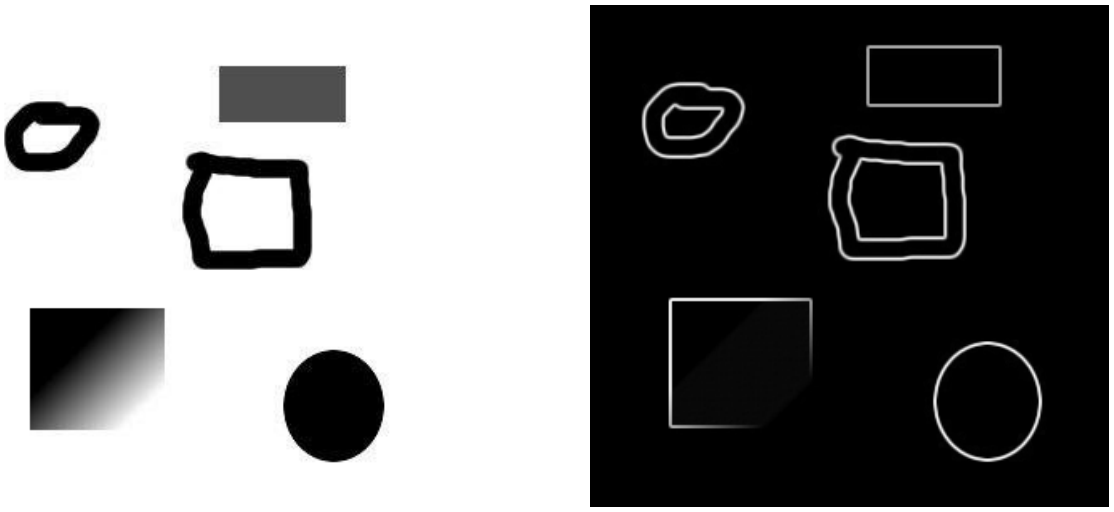
where G_{xi} is the current shifted pixel, G_{xMAX} is the maximum value found in G_x and G_{xMIN} is the minimum one. A similar approach, for horizontal edge detection is seen in figure 2.6. In these figures, values near zero are plotted as black pixels, while the ones near 255 are white.

After G_x and G_y are generated, the Gradient Magnitude (G) in each point of the image is ready to be calculated using equation 2.4. For a better visualization of G, a similar procedure as the one applied to G_x and G_y was taken and the results can be seen in figure 2.7. The edges of the figure are now visible and this is the main matrix used on f_G cost.

As defined in equation 2.1, $f_G(q)$ should have small values so that it would most probably be in the smallest cost path. The pixels belonging to this path must contain strong edge features, hence should be the white ones displayed in figure 2.7. The function $f_G(q)$ is defined as a normalized inverse linear ramp of the gradient magnitude G:

$$f_G(\mathbf{p}, \mathbf{q}) = \left(1 - \left(\frac{G(\mathbf{q}) - G_{min}}{G_{max} - G_{min}} \right) \right) . \frac{\| \mathbf{p} - \mathbf{q} \|}{\sqrt{2}} \tag{2.7}$$

Equation 2.7 should be interpreted as:



(a) Original image (b) G from Sobel convolution

FIGURE 2.7 – Resulting gradient for edge detection

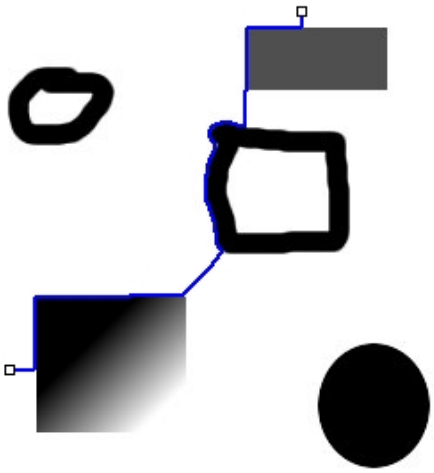


FIGURE 2.8 – Smallest cost path (shown in blue) between two points using only f_G cost

- The ramp is inverted so that the points with maximum gradient should be evaluated to zero, what makes them belong to the shortest paths;
- In case two adjacent points have the same value for $G(q)$, and one is in the diagonal and the other is not, the closer one must be chosen, so the distance scale basically tells: *"The closer, the better"*;
- $\frac{1}{\sqrt{2}}$ is used to normalize the distance, since the farthest pixel is the diagonal one.

Figure 2.8 shows the smallest cost path calculated through the livewire algorithm using only f_G for edge costs. Notice how the livewire follows the edges of the images.

2.1.2.2 Gradient Direction (f_D)

In section 2.1.2.1 it was described a cost that takes in consideration the *magnitude* of the gradient. This section describes another one that considers its *direction*. The f_D cost adds a smooth constraint to the Live-Wire, because it associates a higher cost for sharp changes in boundary direction.

Let $\mathbf{D'(p)}$ be defined as the unit vector normal to the gradient direction $\mathbf{D(p)}$ at pixel $\mathbf{p}$ (i.e. $\mathbf{D(p)} = \frac{1}{G}[G_x(\mathbf{p}), G_y(\mathbf{p})]$ and $\mathbf{D'(p)} = \frac{1}{G}[G_y(\mathbf{p}), -G_x(\mathbf{p})]$, where G is defined in equation 2.4). The gradient Direction cost is defined as:

$$f_D(\mathbf{p}, \mathbf{q}) = \frac{2}{3\pi} \cdot \{\arccos[d_p(\mathbf{p}, \mathbf{q})] + \arccos[d_q(\mathbf{p}, \mathbf{q})]\}, \tag{2.8}$$

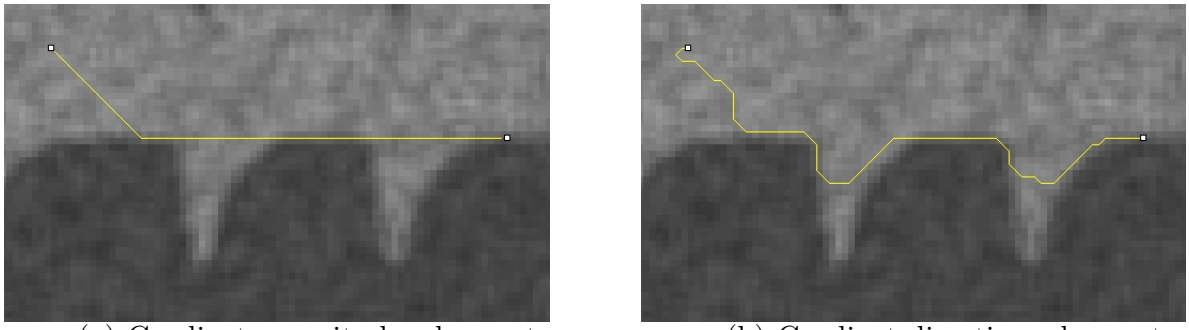
in which $d_p(\mathbf{p}, \mathbf{q}) = \mathbf{D'(p)} \cdot \mathbf{L\ (p,q)}$ and $d_q(\mathbf{p}, \mathbf{q}) = \mathbf{L\ (p,q)} \cdot \mathbf{D'(q)}$ are dot products and $\mathbf{L\ (p,q)}$ is defined as:

$$\mathbf{L(p,q)} = \frac{1}{\| \mathbf{p} - \mathbf{q} \|} \begin{cases} \mathbf{q} - \mathbf{p} & if\ \mathbf{D'(p)} \cdot (\mathbf{q} - \mathbf{p}) \geqslant 0, \\ \mathbf{p} - \mathbf{q} & if\ \mathbf{D'(p)} \cdot (\mathbf{q} - \mathbf{p}) < 0 \end{cases} \tag{2.9}$$

and it is the normalized bidirectional link or unit vector between pixels $\mathbf{p}$ and $\mathbf{q}$. Links are either horizontal, vertical or diagonal (relative to the position of $\mathbf{q}$ in $\mathbf{p}$'s neighborhood) and point such that the dot product of $\mathbf{D'(p)}$ and $\mathbf{L(p,q)}$ is positive, as noted in equation 2.9.

It can be seen from link definition (equation 2.9) that it associates a high cost to an edge or link between two pixels that have similar gradient directions but whose unit vectors normal to the gradient ($\mathbf{D'}$) are perpendicular – or near perpendicular – the link between them. Therefore, looking at equation 2.8, it is clear that the direction feature cost is low when the normal to the gradient direction of the two pixels are similar to each other and the link between them.

There are particular cases in which the direction cost sticks to the edges better than the magnitude cost, for instance in figure 2.9. It happens mostly when smooth curves will better fit the edges, consequently the weights of each feature will dramatically interfer in the quality of segmentation and may be dependent on the type of image in use.



(a) Gradient magnitude edge cost (b) Gradient direction edge cost

FIGURE 2.9 – Comparison of smallest cost paths using different edge costs

One detail of implementation to be observed is how the unit vector is calculated. One shall take care when applying the formula:

$$\widehat{u_x} = \frac{V_x}{\sqrt{V_x^2 + V_y^2}} \tag{2.10}$$

$$\widehat{u_y} = \frac{V_y}{\sqrt{V_x^2 + V_y^2}} \tag{2.11}$$

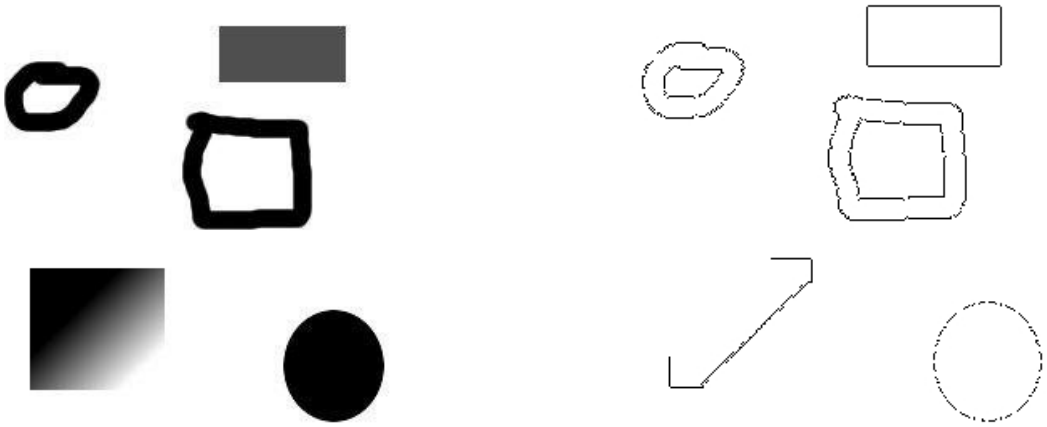
where $\widehat{u_x}$ and $\widehat{u_y}$ are the unit vector's x and y components, respectively; V_x and V_y are the vector's components. The problem of applying such equation without firstly checking if the gradient's magnitude is not zero is that the divisor will be nearly zero, generating a near infinite value for the modulus. To go around this problem, the programmer must firstly check if ($|V_x| < \varepsilon$) and ($|V_y| < \varepsilon$) attributing zero to any dot product that involves the unit vector whose gradient magnitude is near zero.

2.1.2.3 Laplacian Zero-Crossing (f_Z)

The Laplacian Zero-Crossing function f_Z is a binary function defined as:

$$f_Z(q) = \begin{cases} 0 & \text{if } I_L(\mathbf{q}) = 0, \\ 1 & \text{otherwise} \end{cases} \tag{2.12}$$

where $I_L(\mathbf{q})$ is the Laplacian of the image at pixel $\mathbf{q}$. According to (MARR; HILDRETH, 1980), the convolution of an image with a laplacian kernel approximates the second partial



(a) Original image (b) Laplacian zero-crossing pixels

FIGURE 2.10 – Edge detection using Laplacian zero-crossing cost

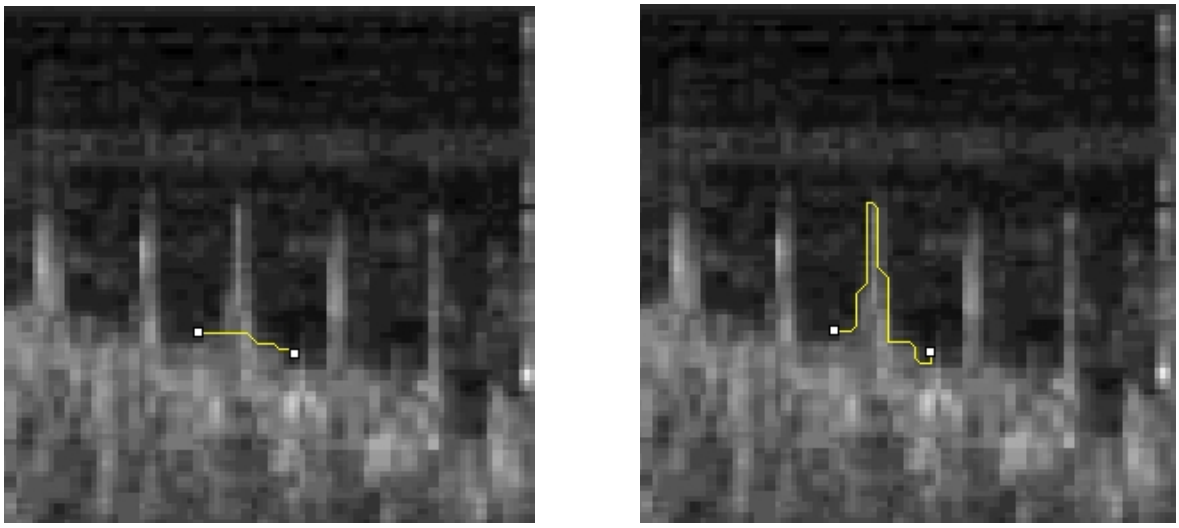
derivative of the image. This laplacian image zero-crossing corresponds to points of maximal (or minimal) gradient magnitude, thus, representing good edge properties. Intensity changes at a given scale are best detected by finding the zero values of $\bigtriangledown^2 G(x,y) * I(x,y)$ for image I, where $G(x,y)$ is a two-dimensional Gaussian distribution and $\bigtriangledown^2$ is the Laplacian. In order to calculate the Laplacian and Gaussian in a single convolution, the function Laplacian of Gaussian (LoG) was used (NISHIHARA; CROSSLEY, 1988):

$$LoG(x,y) = -\frac{1}{\pi\sigma^4}\left[1 - \frac{x^2+y^2}{2\sigma^2}\right] e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{2.13}$$

As very few zero-valued pixels are found applying Laplacian of Gaussian in a discrete image, the same procedure of (MORTENSEN; BARRETT, 1995) was used: if two neighboring pixels change from positive to negative, the one closest to zero is used to represent the zero-crossing. Figure 2.10 shows, in black, the pixels that assumed zero value using this approach.

2.1.2.4 Exponential Cost (f_E)

Another cost that seemed very useful for edge detection was the Exponential Cost (f_E). This cost comes from a suggestion of Prof. Sergio Furuie when it was beholden that the gradient magnitude (f_G) was not following the edges correctly in particular cases, for instance, in figure 2.11a. It is clear that the path that does not follow the edge, although



(a) 100% f_G weight (b) 100% f_E weight

FIGURE 2.11 – Comparison of different weights for smallest cost path

cheaper, is not desired. Nevertheless, this path is chosen because going all around the edge would have a higher cost.

The solution found was giving a higher cost for edges that would go to pixels with low gradient. f_E is defined as:

$$f_E(\mathbf{p},\mathbf{q}) = f_p(\mathbf{q}) \cdot f_G(\mathbf{p},\mathbf{q}) \tag{2.14}$$

$$f_p(\mathbf{q}) = e^{-p_w \frac{G(\mathbf{q})-G_{min}}{G_{max}-G_{min}}} \tag{2.15}$$

where $f_G(\mathbf{p},\mathbf{q})$ is defined in equation 2.7 and p_w is a *constant*, adjustable in execution time. When multiplying f_p by f_G, points with higher gradient will have smaller weights, and the ones with low gradient will have more weight, as it can be seen through figure 2.12 (in which $x = \frac{G(\mathbf{q})-G_{min}}{G_{max}-G_{min}}$), resulting in worse paths, hence not being chosen.

Figure 2.11a shows Live-Wire using total weights in magnitude cost, while exponential cost works better, as seen in figure 2.11b.

2.1.3 Dynamic Programming Graph Searching

As Live-wire is based in the search for an optimal path in a graph, Dijkstra's algorithm (DIJKSTRA, 1959) may be used, but a couple remarks must be watched:

1. There are no *a priori* nodes defined to finish the search;

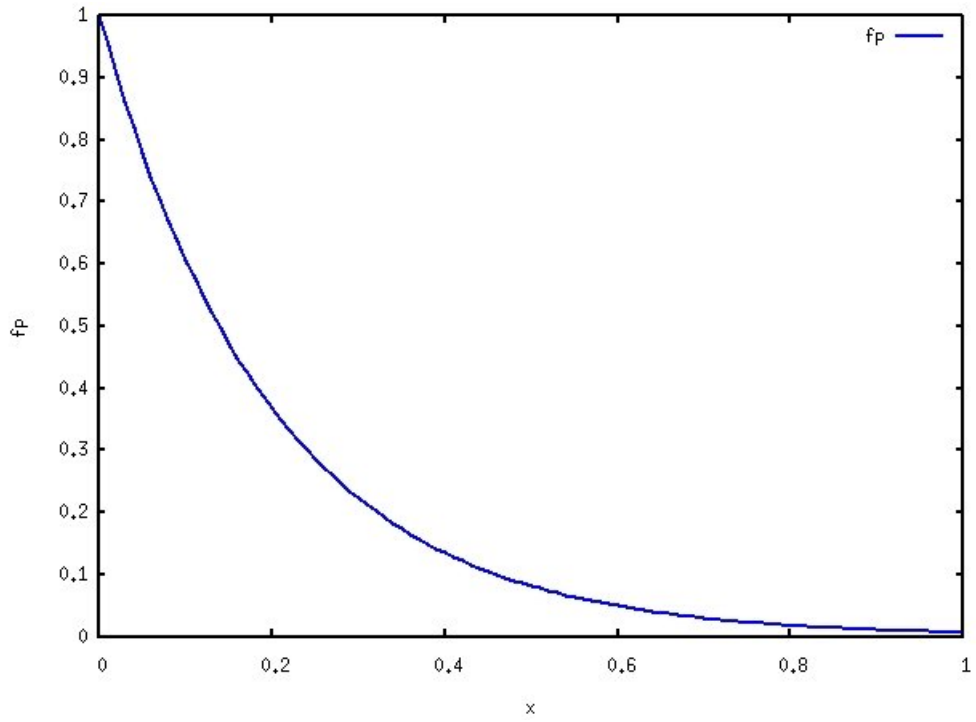


FIGURE 2.12 – Plot of f_p

2. A sparse[2] graph is used;

3. The next element to be expanded is retrieved with $O(1)$ for N nodes/pixels.

The **pseudo-code** for the algorithm used is as follows:

Algorithm: Live-Wire Optimized Graph Search

```
Input:
   s         Start or seed pixel
   e(q,r)    Local cost function for edge going from q to r

Data structures:
   Q         Heap queue of active pixels sorted by total cost (initially empty)
   N(q)      Neighborhood set of q(containing 8 neighbor pixels)
   v(q)      boolean to store if q was already visited and expanded
   g(q)      Total cost function from seed point to q

Output:
   p         Pointers from each pixel indicating the minimum cost path
```

[2]a graph with only a few edges

```
Algorithm:
  g(s)←0; L←s;           Initializes active list with zero cost seed pixel
  while L≠ ∅ do begin    While there are still points to expand
    q←min(Q)             Removes minimum cost pixel from active heap
    v(q)←TRUE            Marks q as visited
    for each r ∈N(q) such that not v(q) do begin
      cTMP←g(q) + e(q,r);       Computes total cost to neighbor
      p(cTMP)← q                Stores back pointer
      Q ← cTMP                  Inserts pixel in Heap
      while min(Q) is visited do begin  Removes minimum cost pixels
        min(Q) ← Q                      that have already been expanded
      end
    end
  end
```

2.1.4 Thread implementation

In order to achieve a faster user response, a thread implementation was used. When the user sets one point in the image a thread is started. This thread follows the algorithm described in section 2.1.3. If the point to where the mouse was moved has already been calculated, the path is drawn, else, it waits until the path is calculated.

If another point is set, there is no need to keep the thread running – which would calculate unuseful points – so, a *flag* is set. In the main loop of the algorithm, it is checked whether the flag *Thread stopped* is set or not. In positive case, the loop is stopped and another thread is started, seeding a new point.

This approach gives the user a faster response, because points that are not desired do not need to be calculated, and the answer is given to the user as soon as it is ready.

2.2 Java Extensible Snake System (JESS)

While *Snakes* (Active Contour Models) are a powerful model based image segmentation technique, it hasn't been in commercial packages and has been restricted to the academic world. In order to solve this problem a *Java Extensible Snake System* (MCINERNEY; SHARIF; PASHOTANIZADEH, 2005) was created. It is open source, general, portable, and extensible. This system implements *Snakes* algorithms in a hierarchical fashion, with

a general *Snake* class, followed by several subclasses that add more functionality, including a subdivision-curve *Snake*.

A formal definition for a *Snake* is, according to (LIANG; MCINERNEY; TERZOPOULOS, 2006), a time-varying parametric contour $\mathbf{v}(s,t) = (x(s,t), y(s,t))^T$ in the image plane $(x,y) \in R^2$, where x and y are coordinate functions of parameter s and time t. The shape of a *Snake* subject to an image $I(x,y)$ is related to an energy functional $E(v) = I(v)+P(v)$. The first term is the internal deformation energy:

$$I(v) = \frac{1}{2}\int_0^L \alpha(s)\left|\frac{\partial v}{\partial s}\right|^2 + \beta(s)\left|\frac{\partial^2 v}{\partial s^2}\right|^2 ds \tag{2.16}$$

where $\alpha(s)$ controls the "tension" of contour and $\beta(s)$ regulates its "rigidity". $P(v)$ is related to the external energy:

$$P(v) = \int_0^L P_I(v)\, ds \tag{2.17}$$

This energy couples the *Snake* to image features via a scalar potential $P_I(x,y)$ generally computed from image gradient magnitude.

One of the models used for *Snakes*, as in (LOBREGT; VIERGEVER, 1995), is a set of connected vertexes. An initial contour is roughly defined and then it is automatically modified by an energy minimizing process. The energy is defined by an internal model that depends on local curvature, while the external one is derived from image features. A deformation process is then derived from the forces generated by these fields, which stops when a local minimum is reached.

Figure 2.13 shows the basic structure of a *Snake*. The vertexes are obtained by initial delineation and deformation is caused by a combination of forces which act on the vertexes. The vectors $\overrightarrow{a}$ symbolize the resulting acceleration due to the forces. Another vector that acts on the vertexes and is not represented in the figure is the velocity.

The internal forces of a model are related to the local contour curvature. The intention of these forces is to minimize local curvature, acting as a counterbalance to external forces.

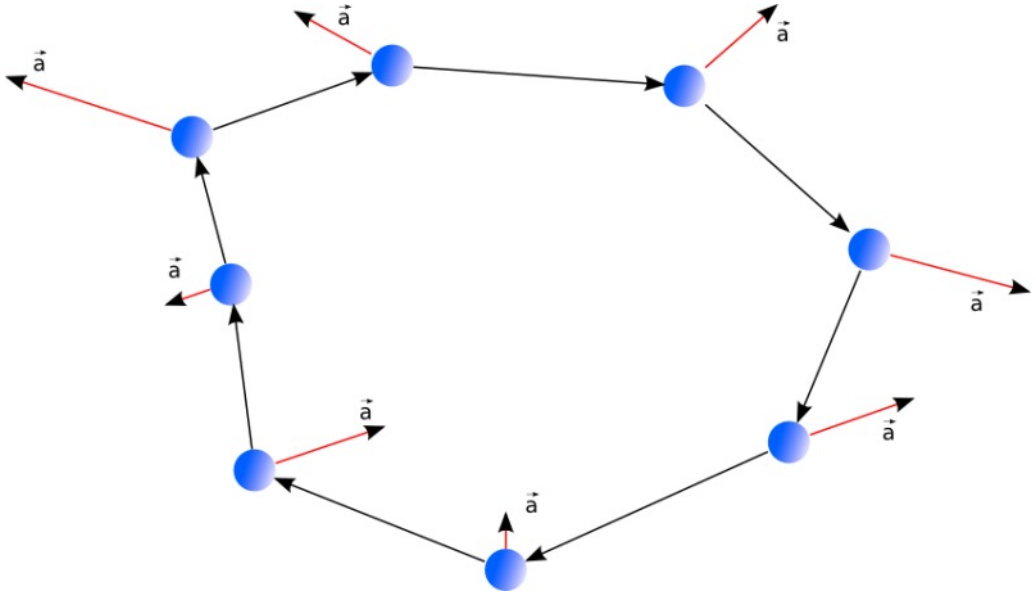


FIGURE 2.13 – Vertexes of a *Snake* showing acceleration vectors

This way, internal forces are defined by:

$$f_{in,i} = (c_i \cdot \widehat{r}_i) \otimes k_i \tag{2.18}$$

where c_i is the local curvature at vertex V_i, which is defined by the difference of two following unit direction vectors in consecutive vertexes. r_i is the local radial direction derived from the tangential unit vector, by a rotation over $\frac{\pi}{2}$ radians. k_i is a discrete filter.

Equation 2.18 allows internal forces $f_{in,i}$, which act on the vertexes V_i to have the same radial direction as the curvature vectors. Another feature of this equation is that it reduces local curvature without affecting areas of constant curvature. This is achieved if an appropriate symmetric filter k_i is chosen, with zero frequency component equal to zero (LOBREGT; VIERGEVER, 1995). A simple filter that has these features is:

$$k_i = \{..., 0, 0, -\frac{1}{2}, 1, -\frac{1}{2}, 0, 0, ...\} \tag{2.19}$$

where the value 1 is applied to position i and $-\frac{1}{2}$ to positions $i-1$ and $i+1$.

The image feature requested is obtained through the action of external forces. This is represented by an external potential energy distribution. In order to detect edges,

the gradient length may be used as an image feature. The deformation process is then implemented so that it will attempt to pull vertexes into a local minima of the energy distribution. Let E_{im} be the resulting distribution of potential energy. The force field that will pull towards the direction of lower energy is described by the following relation:

$$f_{im} = -\nabla E_{im} \tag{2.20}$$

If this force is applied to each vertex, considering no internal energy, in the end of the process, the contour model will connect to a local energy minima through the external energy distribution. The only problem with equation 2.20 is that the external force will also have a component along the path of the model, which would result in a force that moves the contour and form clusters in local minima, which is undesirable. Hence, only the radial component is taken, which is obtained by a cross product with r_i.

Since it may take too long for the contour model to come into a stable condition, or the model may remain oscillating between two states, a damping force is added. The general equation for the forces acting in each of the vertexes is given by the following equation:

$$f_i = \omega_{ex} f_{ex,r_i} + \omega_{in} f_{in,i} + f_{damp,i} \tag{2.21}$$

$$f_{damp,i} = \omega_{damp}\ v_i \tag{2.22}$$

where $f_{damp,i}$ is the damping force acting on vertex i and v_i is the velocity of vertex i. According to (LOBREGT; VIERGEVER, 1995) the equations for deforming are represented by:

$$\mathbf{p}_i(t+\Delta t) = \mathbf{p}_i(t) + \mathbf{v}_i(t)\Delta t \tag{2.23}$$

$$\mathbf{v}_i(t+\Delta t) = \mathbf{v}_i(t) + \mathbf{a}_i(t)\Delta t \tag{2.24}$$

$$\mathbf{a}_i(t+\Delta t) = \frac{1}{m_i}\mathbf{f}_i(t+\Delta t) \tag{2.25}$$

where $\mathbf{p}_i(t)$ is the position of vertex i at iteration t and m_i is a constant scaling factor.

Since the implementation was done using *JESS* framework, a few parameters must be explained. Figure 2.14 shows the class diagram from *JESS*. The class used in this study

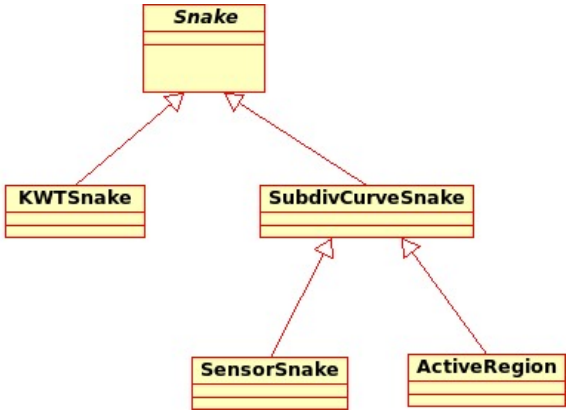


FIGURE 2.14 – Class diagram of JESS system

was a *SensorSnake* and it has the following parameters:

- *imageForceSF*: this parameter describes the scale factor for force vectors;
- *forceLimit*: if force magnitudes transpass this value, they are scaled to the *forceLimit*;
- *stretchSF*: it is the internal force scaling factor for stretching the *Snake*;
- *bendSF*: it is the internal force scaling factor for bending the *Snake*;
- *minLineSearch*: number of minimum pixels to look for imageEdge points;
- *maxLineSearch*: number of maximum pixels to look for imageEdge points;
- *minEdgeIntensity*: minimum intensity to be considered as edge;
- *maxEdgeIntensity*: maximum intensity to be considered as edge.

The following parameters were used:

```
imageForceSF     = 0.8;
forceLimit       = 100.0;
stretchSF        = 0.002;
bendSF           = 0.0;
minLineSearch    = -2 ;
```

```
maxLineSearch    = 10;
minEdgeIntensity = 35;
maxEdgeIntensity = 255;
```

Besides setting these parameters, the number of levels of subdivision was set to 5. This means that each segment will be recursively divided in two during 5 iterations, which gives a number of 32 segments for each initial one, resulting in a smooth curve.

2.3 Simulated Images

In order to have parameters to assure whether the segmentation process has worked properly or not, some simulated images have been used. They have been divided in three types: *Noiseless*, with *Speckle noise* and *Despeckle Filtered Noise*, the three of them described in the following subsections.

2.3.1 Noiseless Simulated Images

The noiseless simulated images cooperate to decrease the complexity of the problem so that the segmentation can be primarily validated in a controlled environment. In order to achieve such goals, the generated images have the following features:

- Same lumen, intima and adventitia color intensity as real IVUS images;
- There is a movement that resembles heart beating, which includes lumen and adventitia radius change as well as geometric center position variation;
- Images will have no noise.

These features have been simulated by another InCor researcher, named Mônica Matsumoto, in a Matlab© environment. All measures have been made to approach as much as possible the same sizes as in real IVUS images. The simulated results can be seen in figure 2.15, which clearly shows the lumen area approaching the catheter – small circle in the center of the figure –, what might happen during a heart beating pulse. Another feature of the image is the color difference of each area. The nine slices come from a series of 150 ones, suggesting a 5 second long intravascular exam, with 30 frames per second.

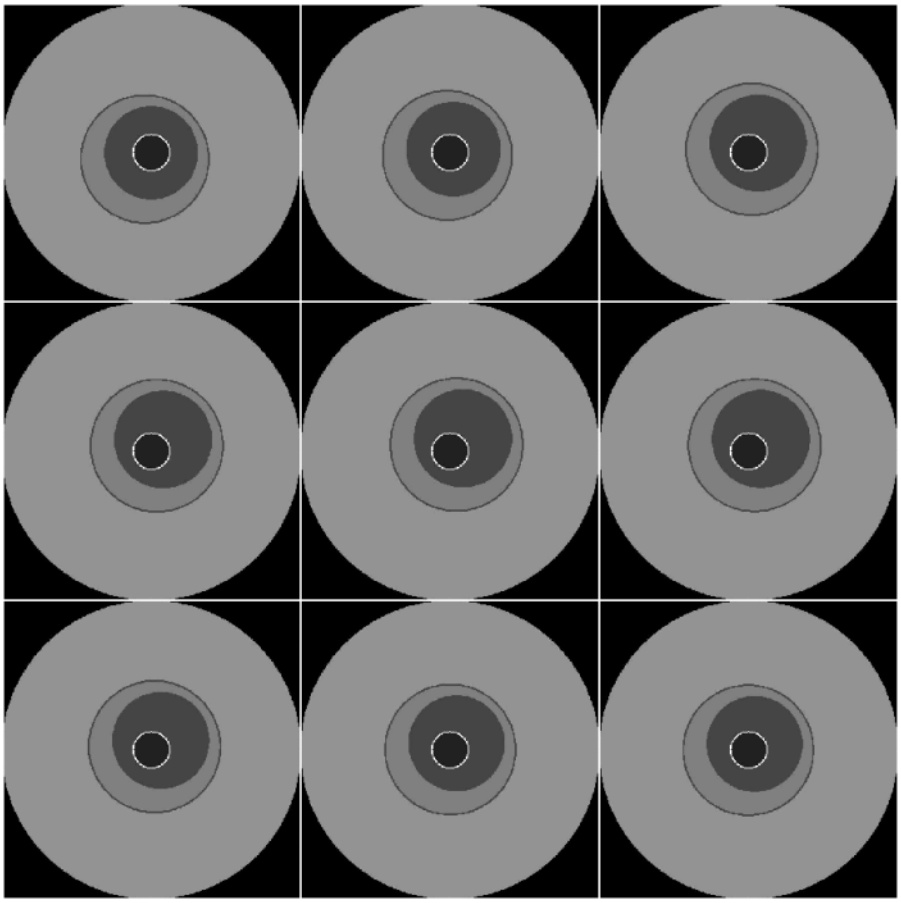


FIGURE 2.15 – 9 slices of a simulated IVUS image series, showing lumen approximation

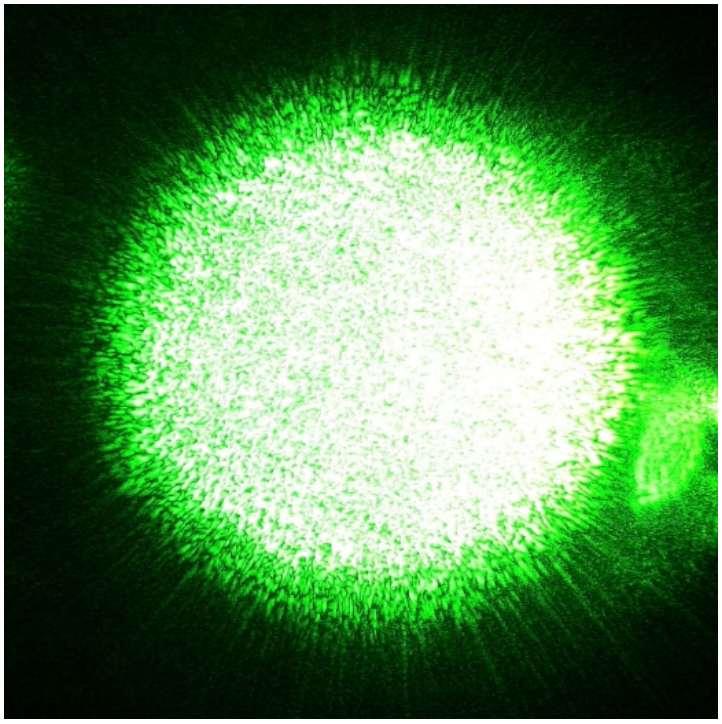


FIGURE 2.16 – Speckle pattern generated by a green laser pointer – licensed under Creative Commons Attribution 2.5 License from user jurvetson on flickr

2.3.2 Images with Speckle Noise

A definition of the Speckle pattern, taken from (TELECOMMUNICATIONS..., 1996) is "a field-intensity pattern produced by the mutual interference of partially coherent beams that are subject to minute temporal and spatial fluctuations". This pattern is, hence, formed by the interference of coherent wavefronts, what happens when watched points are subject to phase differences or intensity fluctuations. An example that shows the speckle noise is taken from the coherent laser beam reflection, as seen in figure 2.16, which demonstrates a laser speckle, on a digital camera image, from a green laser pointer – the speckle pattern is formed by the dark areas with bright islands. The salt-and-peppery appearance would not be observed if the object was illuminated with ordinary light. This kind of noise confuses edge detection and some procedures must be taken as described in section 2.3.3.

Speckle noise takes place not only in optical systems, but also in reflection of radio waves from rough surfaces such as the ground and ultrasonic imaging (PARK; SONG; PEARLMAN, 1999), what justifies the approach of adding this type of noise to the simulated intravascular ultrasound images.

According to (PARK; SONG; PEARLMAN, 1999) a general multiplicative speckle model may be applied so that the observed image pixel $y(i,j)$ on the image plane is represented by:

$$y(i,j) = x(i,j).n(i,j) \tag{2.26}$$

where $x(i,j)$ is the noise-free image and $n(i,j)$ represents the speckle noise. Matlab© models this noise as a random uniform variable with mean 1 and variance σ_n^2, using the equation:

$$J = I + n * I \tag{2.27}$$

where n is the random noise, J is the observed image and I is the noise-free one. The author of this thesis has developed an ImageJ Plugin (section 2.4) that takes the same approach as Matlab©. In order to generate the random variable, with uniform distribution between 0 and 1, the following equation was used (PRESS *et al.*, 2002):

$$I_{j+1} = aI_j + c \ (mod\ m) \tag{2.28}$$

where I_{j+1} is the next random number generated, and the *constants* $a = 1664525$, $c = 1013904223$ and $m = 2^{32}$ are also suggested in (PRESS *et al.*, 2002). As known from statistic sciences, the probability density function of the uniform distribution is:

$$f(x) = \begin{cases} \frac{1}{b-a} & for\ a \leq x \leq b \\ 0 & for\ x < a\ or\ x > b \end{cases} \tag{2.29}$$

Equation 2.29 is plotted in figure 2.17. The uniform distribution is generated from equation 2.28 through the following formula:

$$U = (b-a).I_{j+1} + a \tag{2.30}$$

where U is the uniform random variable generated. It is also known from statistics that uniform distribution has $mean = \frac{a+b}{2}$ and $variance = \frac{(b-a)^2}{12}$.

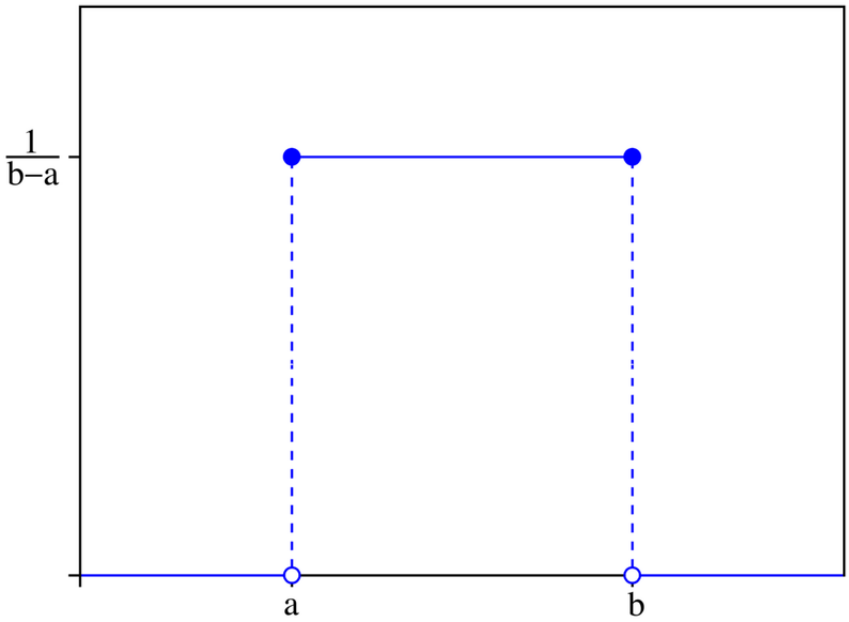


FIGURE 2.17 – Uniform probability density function

In order to increase the difficulty of detecting edges on the simulated images, speckle noise has been added and the result is shown in figure 2.18. Although this model makes no change to regions where the image intensity is zero – the black areas around the image – there is no inconvenience, since these areas are neither segmented nor analyzed, because they don't belong to the regions of interest in IVUS. The variance applied to figure 2.18 is 0.12.

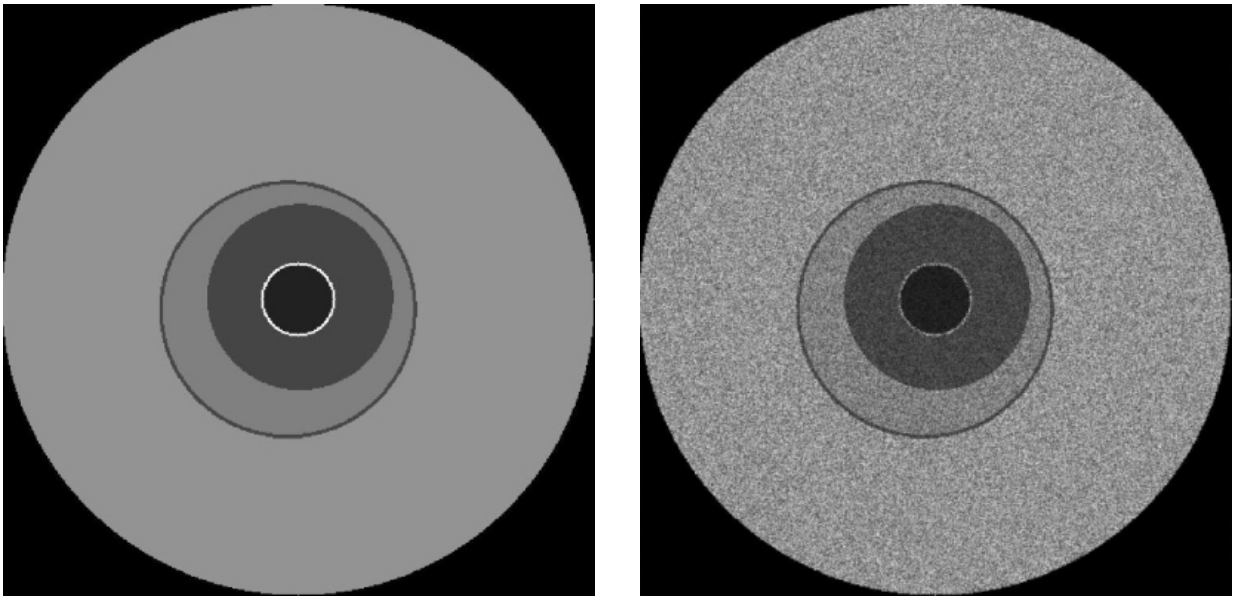
2.3.3 Despeckle Filtered Noise

In order to decrease the noise described in section 2.3.2, a filter that resembles Kalman's (LEE, 1980) has been implemented by another researcher from InCor, Fernando Sales. The general expression for the filter is given by the following equation:

$$f(i,j) = g(i,j) + k(i,j)\,[g(i,j) - \overline{g}] \tag{2.31}$$

$$k(i,j) = \frac{\sigma_g^2}{\overline{g}^2\sigma_n^2 + \sigma_g^2} \tag{2.32}$$

$$\sigma_n^2 = \frac{\sigma_p^2}{\overline{g_p}} \tag{2.33}$$



(a) Original simulated image (b) Same image with Speckle noise added

FIGURE 2.18 – Result of applying simulated speckle noise from model in equation 2.26

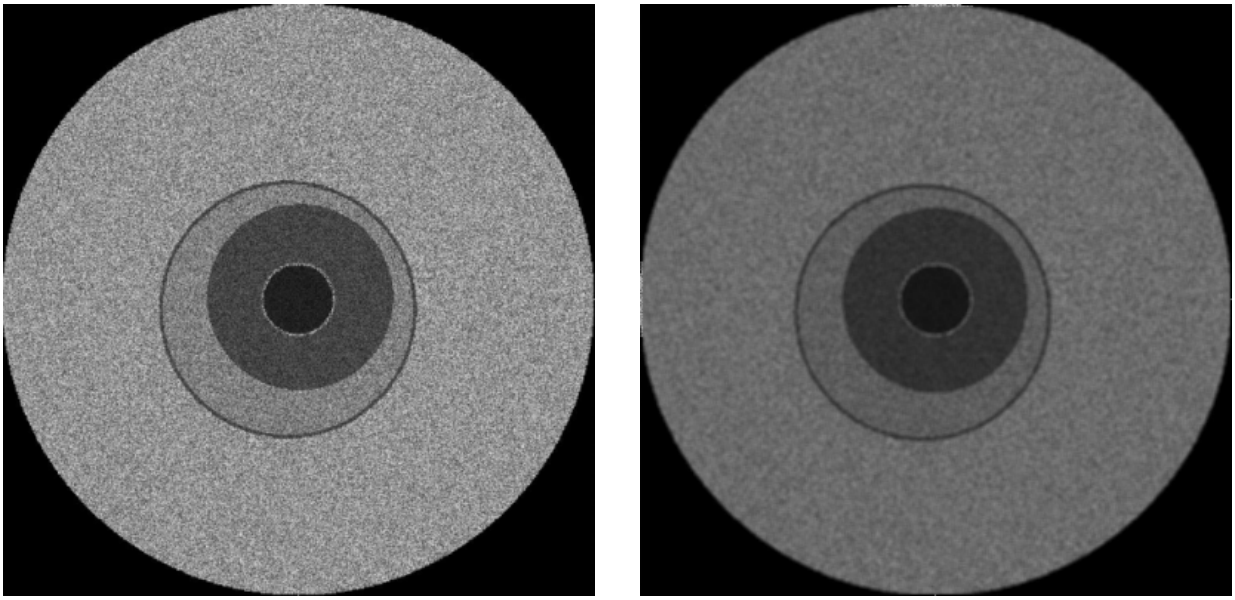
where $f(i,j)$ is the value of the pixel after applying the filter, $g(i,j)$ is the value before filtering, $\overline{g}$ and σ_g^2 are respectively the average intensity and the variance of the pixels watched in a square 7x7 kernel centered in the position (i,j), σ_n^2 is an estimation of the noise variance, computed in a homogeneous region of the image. Parameters $\overline{g_p}$ and σ_p^2 are, respectively, the average and variance of the pixels in a user defined window, used to estimate noise parameters.

Figure 2.19 shows the result of applying the filter described in equation 2.31 over a simulated speckle noise image, as the one in figure 2.18.

2.4 ImageJ

ImageJ (*http://rsb.info.nih.gov/ij/*) is an easily extensible Java-based image processing program developed at the National Institutes of Health, which is an agency of the United States Department of Health and Human Services responsible for biomedical research, located in Bethesda, Maryland. The main developer is Wayne Rasband, who has helped the author of this thesis along the plugin's development.

ImageJ has been selected as the main platform over which this thesis has been developed because of the fact that it is a public domain tool and has other important features,



(a) Simulated image with speckle noise (b) Filtered image

FIGURE 2.19 – Result of applying speckle filter from equation 2.31

like:

- *Cross platform*: ImageJ runs on Linux, Mac OS 9, Mac OS X, Windows and the Sharp Zaurus PDA;
- *Open source*: All Java source code is freely available and in the public domain, without license requiring. This fact enormously helps plugin development, since the code is documented and can be easily browsed;
- *User community*: More than 1400 users subscribe to the ImageJ mailing list, which provides fast responses for developers;
- *Plugins*: The functionality of ImageJ is highly increased through the development of plugins, as described in section 2.5;
- *File Formats*: It opens and saves GIF, JPEG, BMP, ASCII, TIFF and opens DICOM, FITS, PNG and PGM. One format that is extremely used in medical environment is the DICOM format. It stands for "Digital Imaging and Communications in Medicine" and is a set of standards for handling, storing, printing and transmitting information in medical imaging. An interesting feature available in DICOM files is the pixel length given in millimeters, what gives the real size of the images, an important feature to evaluate exams;

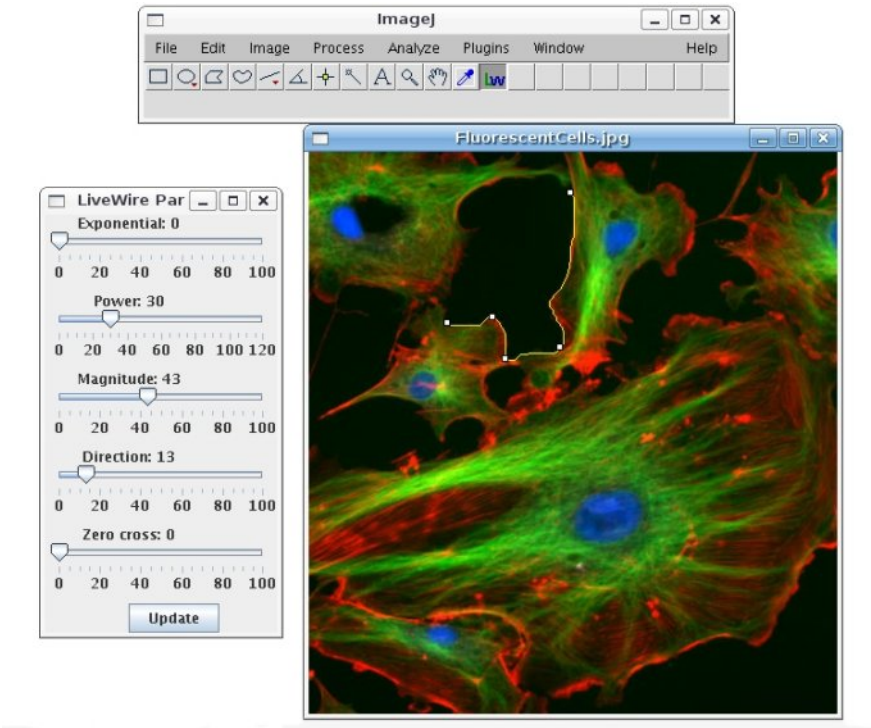


FIGURE 2.20 – Screenshot of ImageJ running on Linux with LiveWire plugin installed, during a cell segmentation

- *Selection Framework*: ImageJ gives a complete framework for handling regions of interest, which includes creating and moving rectangular, elliptical or irregular area selections. Edit, filter, draw, fill, clear, filter and measure selections, as well as saving them on separate files, which are crucial features for image segmentation.

- *Stacks*: It works with stacks, which is how three-dimensional exams are displayed to the user, in a slice-by-slice fashion. Most of the coronary evaluations are assessed this way.

Besides all these features, ImageJ is extensively documented. The version used in this thesis is 1.37s, released in September, 19th, 2006. A screenshot of the program running can be seen in figure 2.20

2.5 Developed ImageJ Plugins

The main feature explored in ImageJ was its easy extensibility through plugins. Under developer resources (*http://rsb.info.nih.gov/ij/developer/index.html*) a good tutorial(BAILER, 2006) on how to write plugins for ImageJ is found.

2.5.1 Writing a simple ImageJ Plugin

Writing a plugin is basically implementing a PlugInFilter interface. The methods that must be coded in this interface are:

`void run(ImageProcessor ip)`

This method runs the plugin, all code written here is executed by ImageJ. The ImageProcessor is an abstract superclass of the image processors for certain image types. The methods provided by the ImageProcesor will actually work on the image data. Depending on the type of image used, one of the following ImageProcessor subclasses is used: *ByteProcessor* – used for 8 bit grayscale and color images – *ShortProcessor* – 16 bit grayscale – *ColorProcessor* – 32 bit integer images – and *FloatProcessor* – used for 32 bit floating point images.

`int setup(java.lang.String arg, ImagePlus imp)`

This is used to set up the plugin. The String *arg* passes arguments so that the plugin may act differently based on what it receives. The ImagePlus class is the representation of an image in ImageJ, based on *ImageProcessor*. This method returns an integer that represents which type of images it can handle. This value is typically *DOES_8G*, which states that the plugin can handle 8 bit grayscale images.

A simple example of a plugin is given after this paragraph, so that one can have a general idea of how the image is managed. This plugin sample shows how to act on an 8 bit grayscale image to get its negative. The code is:

```
import ij.*;
import ij.process.*;
import ij.gui.*;
import java.awt.*;
import ij.plugin.filter.*;

public class Simple_ implements PlugInFilter {
  ImagePlus imp;

  public int setup(String arg, ImagePlus imp) {
    this.imp = imp;
    return DOES_8G;
  }
```

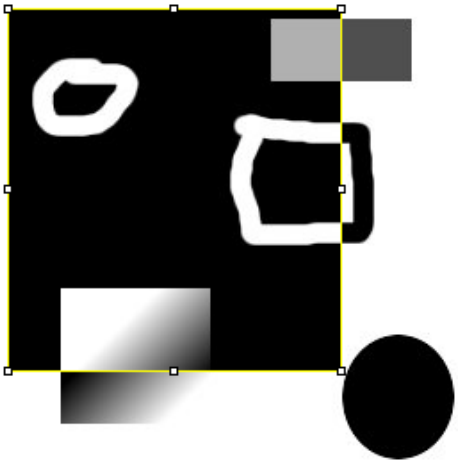


FIGURE 2.21 – Result of the sample plugin applied to invert the pixels that belong to the selected region of interest

```
  public void run(ImageProcessor ip) {
    Rectangle r = ip.getRoi();
    byte[] pixels = (byte[])ip.getPixels();
    int width = ip.getWidth();
    int offset, i;
    for (int y=r.y; y<(r.y+r.height); y++) {
      offset = y*width;
      for (int x=r.x; x<(r.x+r.width); x++) {
        i = offset + x;
        pixels[i] = (byte)(255-pixels[i]);
      }
    }
  }
}
```

The ImagePlus passed as a parameter in setup is firstly stored in the class, for later retrieval – not necessary in the sample plugin. The *setup* also states that the plugin can handle 8 bit grayscale images. Inside *run* method, the selected region of interest is acquired as a rectangle. Following it, a byte array is retrieved from the method *getPixels* of the ImageProcessor class. The two main loops are only used to go over all the pixels on the region of interest, in a row-by-row manner. All the code to handle the points is done in the line `pixels[i] = (byte)(255-pixels[i]);`, which complements each pixel value, generating the negative image, seen in figure 2.21.

2.5.2 Compiling and Installing ImageJ Plugins

In order for ImageJ to load the plugins it is needed that the source java code is compiled and installed in the right place. Supposing the code above is saved in a file called *Simple_.java*, the following command line handles the compiling procedure:

```
javac -cp ij.jar Simple_.java
```

In this context, *javac* is a command line call to the Java compiler (*http://java.sun.com/*). The option *-cp* states that the classpath[3] is given by ij.jar file, which contains all ImageJ compiled classes. The program requires the names of the plugins to include the character '_', because only class files with at least one underscore appear automatically in the "Plugins" menu of ImageJ.

Installing the plugins is a simple procedure, made by copying the *.class* files – generated through the *javac* command – to the *plugins* folder, under ImageJ installation directory. Following all the above steps makes the plugin ready to work in the next initialization of ImageJ. As mentioned, they are found under the "Plugins" menu.

2.5.3 LiveWire Plugin

In order to implement all that has been described in section 2.1, this plugin has been developed by the author of this thesis. Figure 2.22 shows the plugin in more detail. As soon as it is loaded a new toolbutton is added to the main toolbar. It shows the letters "Lw", standing for livewire. The fact of adding a toolbutton is useful because the operator might need to use the other tools, as the zoom utility for instance, while segmentation is done.

The plugin also loads a frame, the one seen on the right, entitled *"LiveWire Parameters"* which consists of five sliders, varying from 0 to 100 – or 0 to 120 on the *Power* slide. Each one of them represents the respective costs of equation 2.1, which means that ω_Z is represented by *Zero cost*, ω_G by *Magnitude*, ω_D by *Direction* and ω_E by *Exponential*. The *Power* slide represents the constant p_w in equation 2.15.

The button update transmits all the readjusted parameters to the cost function, which

[3]Specifies a list of directories, JAR archives, and ZIP archives to search for class files. Class path entries are separated by colons (:)

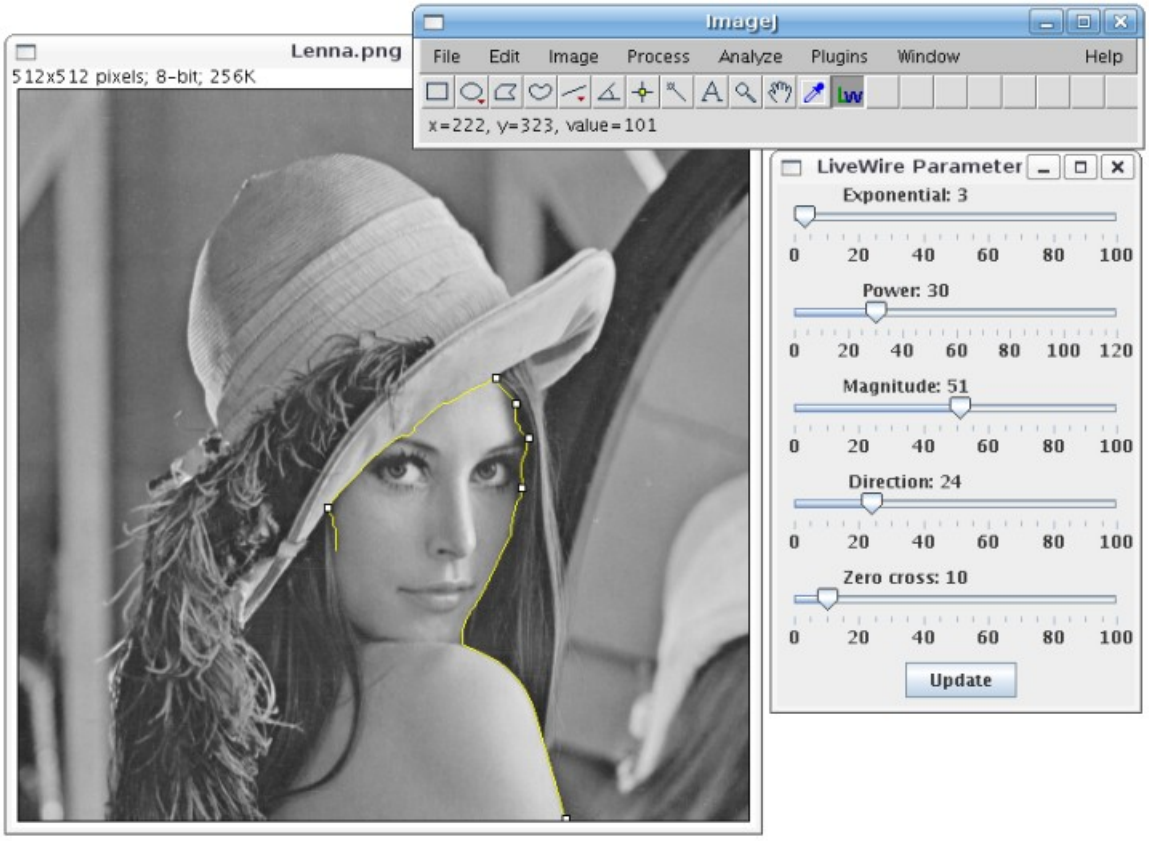


FIGURE 2.22 – Segmentation of traditional Lenna's picture, using Livewire ImageJ Plugin

changes the livewire behavior in realtime. This is extremely useful when parameters are being searched to best fit the type of image. It is observed that some images, with sharp edges, are better segmented with high costs to the exponential weight, while others require more directional weight to smooth the selection.

Along the segmentation, it is possible to see small white squares in figure 2.22. These are the anchor points, which means that whenever the operator clicks the image, the livewire is selected and he is able to continue the segmentation assuming the selection stands steady. These points may be moved after the segmentation is finished, a feature that is useful to adjust points that have not been correctly positioned. When a middle anchor point is moved, both livewires that pass through it are recalculated.

Figure 2.23 shows a class diagram, with the public methods of the main classes of LiveWire plugin. Since no selection implemented in ImageJ satisfied all requirements needed by the LiveWire plugin, a new type of Roi – region of interest – class has been created, named ERoi (Extended Roi). This is a subclass of ImageJ's PolygonRoi, which holds the position of the anchor points and takes care of its drawings, so that the white

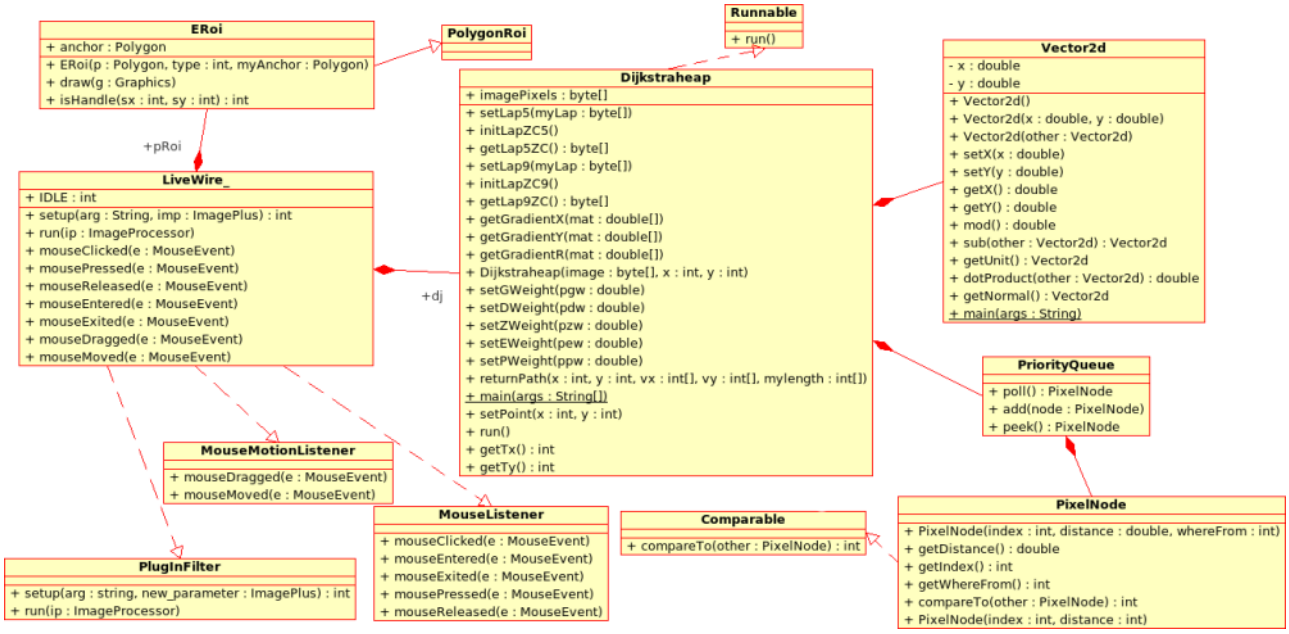


FIGURE 2.23 – Class diagram of LiveWire Plugin, showing public methods of main classes

rectangles are drawn. It also checks if the point clicked belongs to the anchor, what happens if it was nearer than four pixels of distance.

*LiveWire_* class implements *PlugInFilter* interface, required to be run as an ImageJ plugin. It also implements two mouse related interfaces: *MouseMotionListener* and *MouseListener*. While the motion listener handles mouse dragged and moved events, the Mouse Listener catches clicks and releases. All these interfaces are extremely necessary, since the plugin is all related to user interaction.

The main plugin class has also one *Dijkstraheap*, which takes care of both Dijkstra implementation and the binary heap class. This type of association is useful if other algorithms are needed to be tested, which can be easily plugged to *LiveWire_* class. The most important method from *Dijkstraheap* is *returnPath*, which is used to retrieve the shortest path after applying Dijkstra's algorithm. It is called everytime the state of the LiveWire is *WIRE* and a mouseMoved event has occurred. It may happen that the path has not yet been calculated, in this case the length of the size is returned as zero and will only be updated on next mouseMoved event. Another important method from *Dijkstraheap* class is the *setPoint* which sets the initial point for Dijkstra's path. The other methods are related to parameter configuration and cost initialization. As *Dijkstraheap* can work in a thread manner, being stopped when there's no more need to calculate the points, it implements the *Runnable* interface.

Dijkstraheap also uses another class developed, the *Vector2d*. It basically handles common math vector operations, such as retrieving the *modulus, unit vector, dot product, subtraction* and *getting the normal vector*. Those are very useful during computation of the directional cost.

Another class used, from Java, was the *PriorityQueue*. It implements a binary heap with *poll, add* and *peek*. These operations have been discussed in section 2.1.1.1. The priority queue is made of several elements of the type *Pixel Node*, which implement the interface *Comparable*. This interface is used when the position on the heap is decided, according to the cost of the node. It also stores from which other node it came from in order to retrieve the whole path until it. The method *getIndex* returns the position of the pixel in the image, so that it can be displayed.

2.5.4 IVUS Plugin

Practically all IVUS processing is done through this plugin. It performs the following procedures:

- Makes four longitudinal slices;
- Marks upper and lower points for each selection;
- Makes a simple rectangle selection;
- Performs LiveWire and Snakes selection;
- Playbacks the selection;
- Makes measures.

Figure 2.24 shows a real exam and the four slices produced by IVUS plugin, which are: North-South, West-East, Northeast-Southwest and Northwest-Southeast, respectively. The window *assuto1_1.dcm* is loaded with a 44 MByte long DICOM stack of 201 images, which have 480 x 480 pixels and a real measure of 16.22 x 16.22 mm. The same window shows where the North-South longitudinal slice is made. All the slices pass through the center of the image, as seen in figure 2.25. The slice windows seen in figure

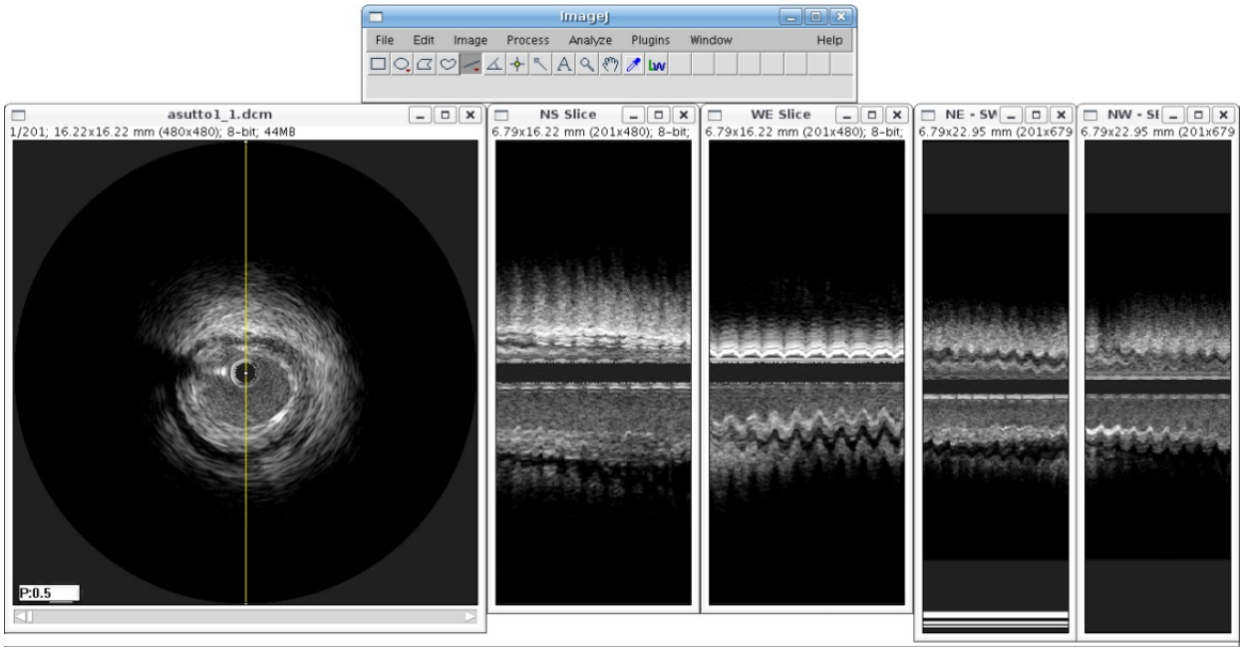
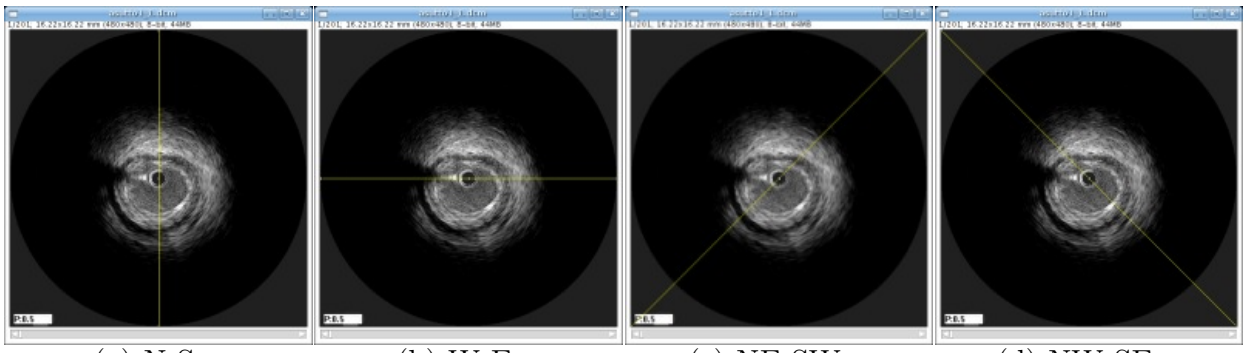


FIGURE 2.24 – Four longitudinal slices made by IVUS plugin on a real exam



(a) N-S (b) W-E (c) NE-SW (d) NW-SE

FIGURE 2.25 – Position of each of the four longitudinal slices

2.24 are generated drawing each of the pixels of the straight line shown in figure 2.25 for each of the images on the stack, hence, diagonal slices are longer.

Besides loading four new windows with each of the slices, the menu seen in figure 2.26 is also displayed. The eight first buttons allow the user to mark points for the selections along the three-dimensional axis of the exam. One example is shown in figure 2.27, in which two points are mapped from the North-South slice. It can be noticed that the upper point is shown as '1' in the main window, and the lower one is shown as '2'. These points are produced as soon as the operator clicks on the *"Mark NS upper points"* and *"Mark NS lower points"*. The delineation of the lumen, in figure 2.27 was previously made using the LiveWire plugin (section 2.5.3).

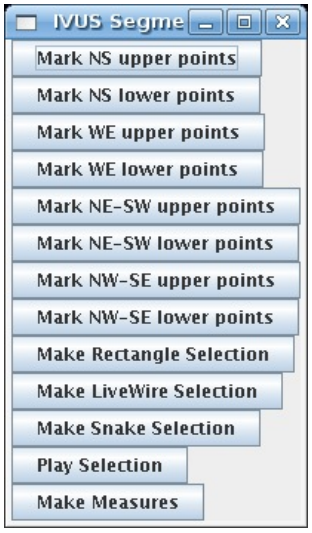


FIGURE 2.26 – IVUS Plugin Menu

It must be noticed that when the points are drawn on the slices, there must be a function to map them back to the original stack. This is accomplished either with simple equations – as using $\frac{width}{2}$ as x coordinate and the y position from the slice, for North-South cuts – or elementary trigonometric relations – as on the diagonal slices.

The button *Make Rectangle Selection*, in figure 2.26 makes a rough, fast rectangular selection, in order to verify if the operator is really making the desired segmentation. It works by drawing a rectangle that passes through the four points mapped on North-South and West-East slices.

Make LiveWire Selection and *Make Snake Selection* will be described, respectively in the following sections 2.5.4.1 and 2.5.4.2.

Another useful function is the *Play Selection*. It works as a playback for all the frames in the stacks, along with the generated selections – may it be LiveWire, Snakes or Rectangle. It uses built-in functions from ImageJ to display the frames in sequence, waiting a time interval of 50 milliseconds, which means 20 frames per second. It also loads the selections stored in the *Roi Manager*, that were previously generated during the process of making the selections.

The last button *Make Measures* is described in more detail in section 2.5.4.3.

The class diagram in figure 2.28 shows how IVUS plugin was implemented. Basically, it implements the PlugInFilter, so that it may be used as a plugin inside ImageJ. It also implements MouseListener and MouseMotionListener interfaces, in order to react to

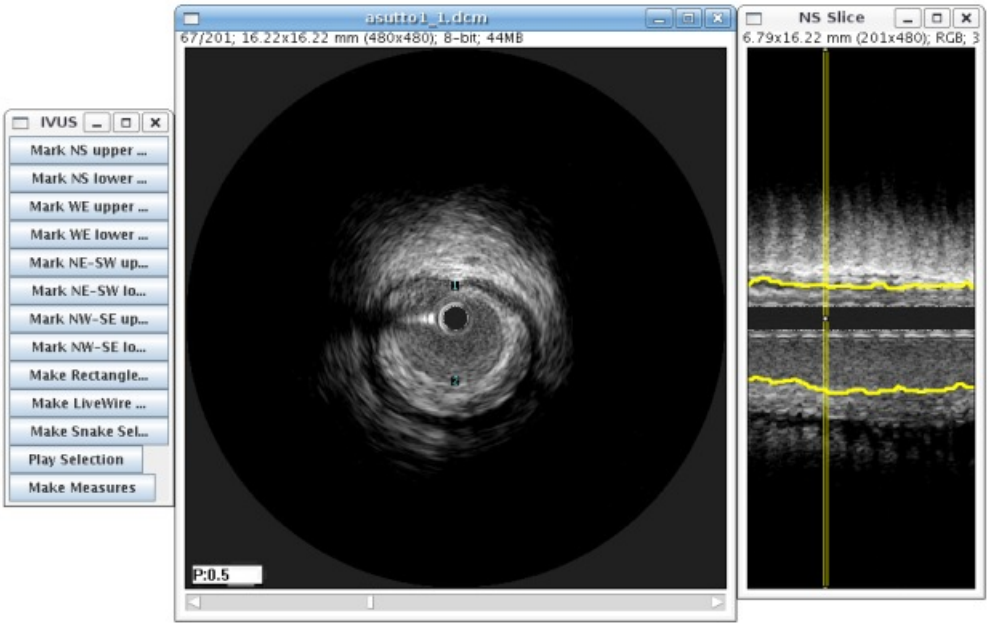


FIGURE 2.27 – Marking points in a North-South slice

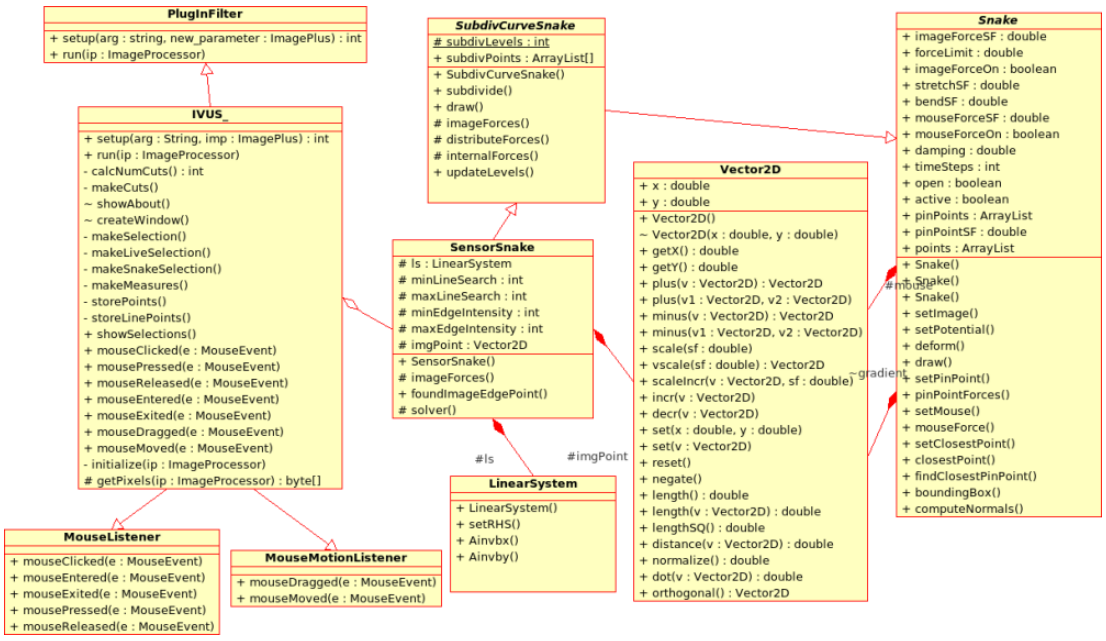


FIGURE 2.28 – Class diagram of IVUS plugin, showing main classes

mouse moves. To add *Snakes* functionality, it uses the class SensorSnake, obtained from JESS. This class is derived from SubdivCurveSnake, which is the type of *Snake* that add points to the initial delineation. The superclass *Snake* has typical *Snakes* methods, such as deform, and standard parameters.

2.5.4.1 Making LiveWire Selections

The process of making LiveWire selections is fully automated, after the segmentation of the slices has been already done. Since the activity of segmenting each one of the frames of an exam is quite tedious – it would take a few hours from the operator, depending on the number of frames – this process is left for the computer.

The implementation of the LiveWire plugin has taken in account that it could also be run as a Macro, which is, to be called from another plugin, with some parameters. This way, the following command:

```
IJ.run ("LiveWire", "x0= 10 y0= 15 x1= 200 y1= 230
magnitude=43 direction=13 exponential=0 power=10");
```

would call the LiveWire plugin to draw the shortest path from point $x_0 = 10$, $y_0 = 15$ up to $x_1 = 200$, $y_1 = 230$, with the parameters from equation 2.1, $\omega_G = 43$, $\omega_D = 13$, $\omega_E = 0$ and $p_w = 10$, from equation 2.15. This way, the eight points obtained from longitudinal segmentation are given as initial and end points to this macro. After each of the segments is finished, they are stored in a temporary selection and then, the complete selection is finally stored in the *Roi Manager* from ImageJ. This procedure can take much time, since each of the frames takes around one minute to be completely segmented. Although it's time consuming, there's no need for any operator being around, so, the procedure may be left running for some hours and afterwards the results will be stored.

The process is shown in figure 2.29. Points from 1 to 8 represent the mapped longitudinal slices in the displayed frame. Selection 9 is the livewire that was obtained running the macro from point 1 to point 2 and from point 2 to 3. This screenshot was taken in the middle of the process. It will continue until the whole structure is segmented, as well as all the frames.

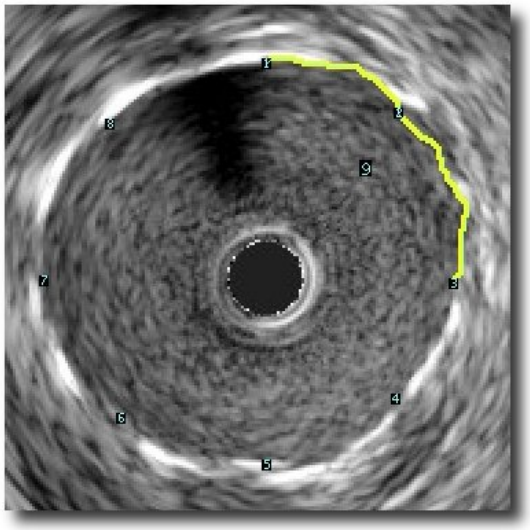


FIGURE 2.29 – The process of making a livewire selection from the points found through longitudinal slices

2.5.4.2 Making Snake Selections

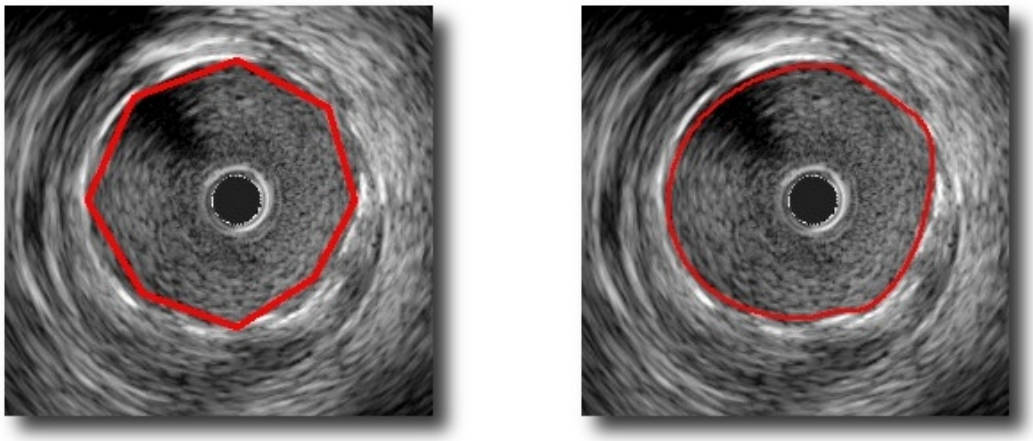
Another procedure to segment the whole three-dimensional structure is using the *Snake Selections*. Similarly as in *LiveWire Selections* the points acquired from longitudinal segmentations are also used as an initial condition for the segmentation. But this time, instead of linking the points with livewires, a polygon is drawn.

Figure 2.30a shows the generated polygon from eight points that were generated in the longitudinal segmentation. After the figure has been filtered, the resulting snake selection is shown in figure 2.30b. The process of generating snakes along the whole stack is quite fast if compared to the LiveWire approach, because it takes around 30 seconds to make the whole three-dimensional segmentation.

2.5.4.3 Making Measures

After the process of segmentation, it's extremely useful to make measures from the selections obtained. The main type of numerical data obtained in this thesis is the area of the selection. This type of measure takes for granted that the selections are closed, else it wouldn't be possible to calculate the area.

Making measures produce three new images for each image in the stack, as seen in figure 2.31. It is required that a selection of gold-standards is loaded previously on the



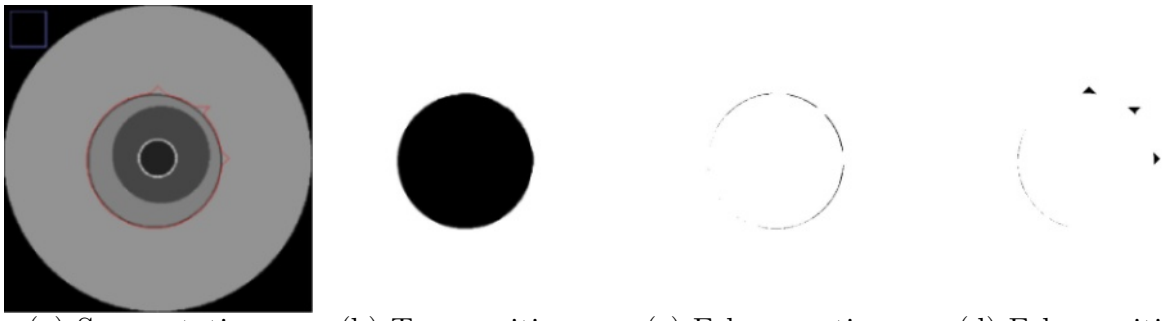
(a) Polygonal selection (b) Resulting snake selection

FIGURE 2.30 – Process of making snake selections from the polygon formed by longitudinal segmentation

Roi Manager. The first image generated is seen in figure 2.31b, which shows a mask of true positives – an *AND* operation between pixels of gold-standard and the selection. Another image generated is in figure 2.31d, which shows false positives – an *AND* operation between the complementary of the gold-standard and the selection. Finally, figure 2.31c shows false negatives – an *AND* operation between pixels from the gold-standard and the complementary of the selection.

Besides visually producing the results, there is also a file generated – *out.txt* – which shows:

- Number of the frame;
- Gold-standard area;
- True-positives area;
- False-positives area;
- False-negatives area;
- True-positives fraction;
- False-positives fraction;
- False-negatives fraction.



(a) Segmentation (b) True-positive (c) False-negative (d) False-positive

FIGURE 2.31 – Results of measures and gold-standard

These measures provide useful results for assessing intravascular ultrasound coronary exams.

3 Results

Since two types of image have been evaluated, the results section will be divided in *Simulated Images* (3.1) and *Real Images* (3.2). Two regions of interest are tested: lumen and adventitia. For each of the images, the procedures described in sections 2.5.4.1 and 2.5.4.2 – respectively *LiveWire* and *Snakes* – were applied.

3.1 Simulated Images

In order to simulate as closely as possible real images, the procedures described in 2.3 were taken. The series of simulated images used have 149 frames, and the first 9 frames can be seen in figure 2.15. The gold-standard for these images has been generated as circles with varying radius and positions so that it could as closely fit the simulated adventitia region.

3.1.1 Noiseless Images

On the first step, noiseless images were considered. The procedure taken was firstly loading the images and then applying IVUS plugin. After that, for each of the slices, *LiveWire plugin* was loaded and the longitudinal segmentation was made. Parameters used are seen in table 3.1.

The time taken by the operator to make all the segmentations was 10 minutes and 30 seconds. Results for LiveWire segmentation are seen in table 3.2. LiveWire Make Selection procedure took 6 hours and 1 minute.

It took 3 minutes and 56 seconds to make the measures. Table 3.3 shows the results for segmentations using Snakes. The type of snakes used was *Sensor Snakes* and 100

TABLE 3.1 – Parameters used in the segmentation of Noiseless images

Slice	ω_Z	ω_G	ω_D	ω_E	p_w
N-S upper	0.00	0.43	0.30	0.00	30
N-S lower	0.00	0.43	0.00	0.00	30
W-E upper	0.00	0.43	0.13	0.00	30
W-E lower	0.00	0.43	0.00	0.00	30
NE-SW upper	0.00	0.43	0.13	0.00	30
NE-SW lower	0.00	0.43	0.00	0.00	30
NW-SE upper	0.00	0.43	0.13	0.00	30
NW-SE lower	0.00	0.43	0.00	0.00	30

TABLE 3.2 – Results using LiveWire segmentations for 149 simulated images without noise

	Gold Standard Area (pixels)	TPF Area (pixels)	FPF Area (pixels)	FNF Area (pixels)	TPF(%)	FPF(%)	FNF(%)
Mean	82359.49	80747.61	77.28	1611.88	98.04	0.09	1.96
Std. dev	1944.73	1991.35	175.35	539.21	0.66	0.21	0.66

deformations were made. The level of subdivisions was 5, which means that each segment was divided 32 times, resulting a polygon of 256 points.

TABLE 3.3 – Results using Snake segmentations for 149 simulated images without noise

	Gold Standard Area (pixels)	TPF Area (pixels)	FPF Area (pixels)	FNF Area (pixels)	TPF(%)	FPF(%)	FNF(%)
Mean	82359.49	75010.79	0	7348.7	91.07	0.00	8.93
Std. dev	1944.73	1956.3	0	552.2	0.68	0	0.68

3.1.2 Images with speckle noise

Segmentation time was 10 minutes and 22 seconds, and was made over the filtered longitudinal slices (same filter as the one that is being applied to the stack). The results of the segmentation with *LiveWire* for images with speckle noise are seen in 3.4. The time taken for processing the images with this technique was 8 hours and 6 minutes.

The results of the segmentation with *Snakes* for images with speckle noise are seen in 3.5. The reason for *Snakes* segmentation true positive results being so small are discussed

TABLE 3.4 – Segmentation results using *LiveWire* for 149 simulated images with speckle noise

	Gold Standard Area (pixels)	TPF Area (pixels)	FPF Area (pixels)	FNF Area (pixels)	TPF(%)	FPF(%)	FNF(%)
Mean	82359.49	79037.38	99.06	3322.11	96.01	0.12	3.99
Std. dev.	1944.73	2188.36	146.38	2617.06	3.09	0.18	3.09

in chapter 6. The time taken for processing the segmentation with *Snakes* was 1 minute and 15 seconds.

TABLE 3.5 – Segmentation results using Snakes for 149 simulated images with speckle noise

	Gold Standard Area (pixels)	TPF Area (pixels)	FPF Area (pixels)	FNF Area (pixels)	TPF(%)	FPF(%)	FNF(%)
Mean	82359.49	1986.46	10379.85	80373.03	2.37	12.51	97.63
Std. dev.	1944.73	12361.64	47598.14	12280.07	14.7	57.04	14.7

3.1.3 Filtered Images

The images analyzed on this section are the same from section 3.1.2 submitted to a filter, as described in section 2.3.3. Table 3.6 shows results obtained with *LiveWire*, while the time taken for this segmentation was 8 hours and 1 minute.

TABLE 3.6 – Segmentation results using *LiveWire* for 149 filtered simulated images

	Gold Standard Area (pixels)	TPF Area (pixels)	FPF Area (pixels)	FNF Area (pixels)	TPF(%)	FPF(%)	FNF(%)
Mean	82359.49	77481.93	182.6	4877.56	94.14	0.22	5.86
Std. dev.	1944.73	2922.61	198.18	3734.39	4.42	0.24	4.42

The results for segmentation using *Snakes* technique in filtered images took 1 minute and 15 seconds. These results are shown in table 3.7.

TABLE 3.7 – Segmentation results using *Snakes* for 149 filtered simulated images

	Gold Standard Area (pixels)	TPF Area (pixels)	FPF Area (pixels)	FNF Area (pixels)	TPF(%)	FPF(%)	FNF(%)
Mean	82359.49	76336.08	32811.51	6023.41	92.67	39.08	7.33
Std. dev.	1944.73	3178.23	98412.83	2095.49	2.55	117.29	2.55

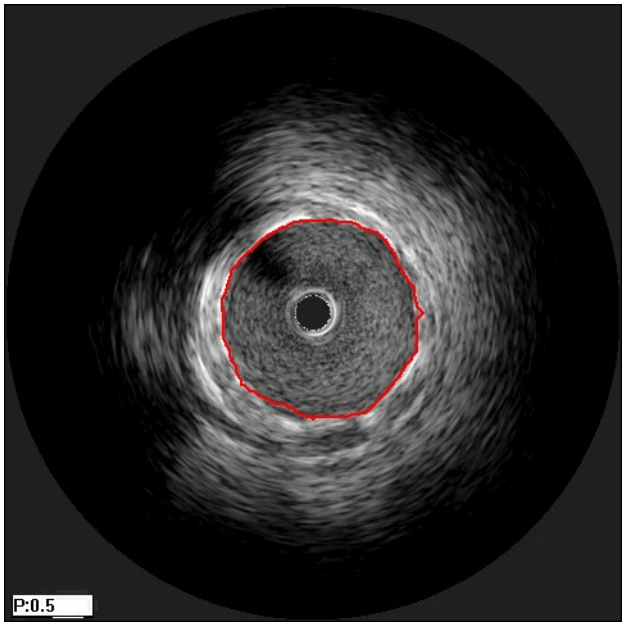


FIGURE 3.1 – Segmentation of real coronary lumen using *LiveWire* technique

3.2 Real Images

Although an effective way to measure how both methods would perform over real images would require a specialist, an attempt to use them was made. Figure 3.1 shows how *LiveWire* technique performed over the lumen segmentation of a real patient. This image shows only one frame of the 351 image stack. The procedure used in this segmentation was the same as the one applied in simulated images without noise, since *LiveWire* has suffered less from noise.

Figure 3.2 shows a North-South longitudinal slice segmentation of a real exam, used to generate points for *LiveWire* and *Snakes*. The time taken for the operator to do the segmentation of the four slices was about 8 minutes and very few anchor points were needed – around 40 points for the whole segmentation.

Another attempt to see how the technique works with real images was made using

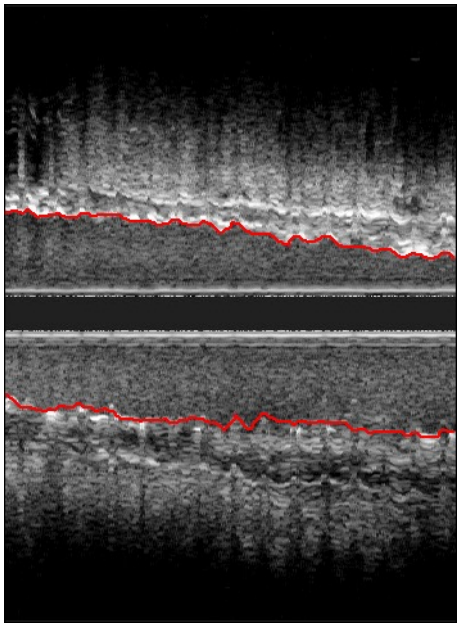


FIGURE 3.2 – Segmentation of North-South longitudinal slice using *IVUS* and *LiveWire* plugins

Snakes. In order to achieve better results, the same procedure as the one described in section 2.3.3 was used. Figure 3.3 shows the segmentation obtained after the filtering process. The selections were plotted over the original images, since there was no need to show background operations to the user.

3.3 Plugin Deployment

Since *LiveWire* plugin could be used in different areas and help the world community in *Segmentation*, this software was made publicly available through GNU General Public License, as seen in Appendix A. This license is widely used in free software and grants the recipients of the program the rights of free software definition and, to insure freedom is preserved, it uses copyleft. According to (THE...,), "Free software is a matter of the users' freedom to run, copy, distribute, study, change and improve the software. More precisely, it refers to four kinds of freedom, for the users of the software:

- The freedom to run the program, for any purpose (freedom 0);
- The freedom to study how the program works, and adapt it to their needs (freedom 1). Access to the source code is a precondition for this;

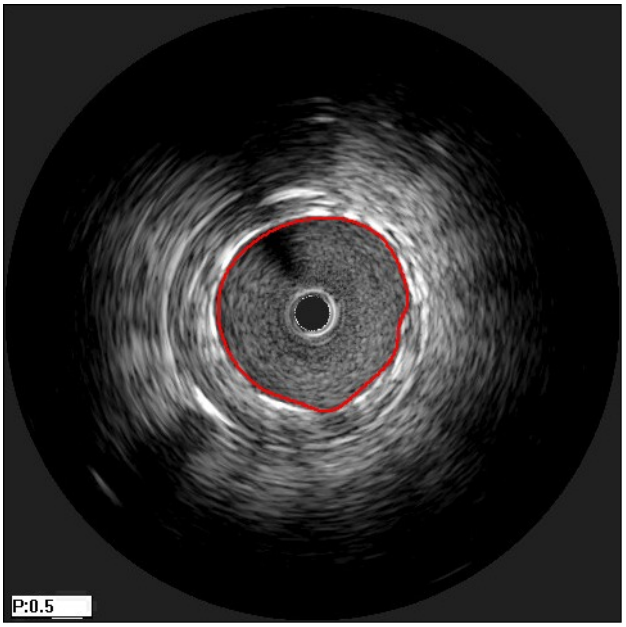


FIGURE 3.3 – Segmentation of real coronary lumen using *Snakes*

- The freedom to redistribute copies so they can help their neighbors (freedom 2);
- The freedom to improve the program, and release their improvements to the public, so that the whole community benefits (freedom 3). Access to the source code is a precondition for this."

In order to follow the principles of free software, the deployment method of choice was submitting a project to SourceForge.net. Hosted at address *http://sourceforge.net/*, it is a central location for software developers to control and manage open source software development and acts as a source code repository. The main features provided by Sourceforge are:

- CVS systems to allow simultaneous coding, as well as storing versions of the programs
- Mailing lists
- Web site for the project
- Screenshots, news and statistics about visitors
- Bug tracker and feature requests
- Download mirrors and statistics

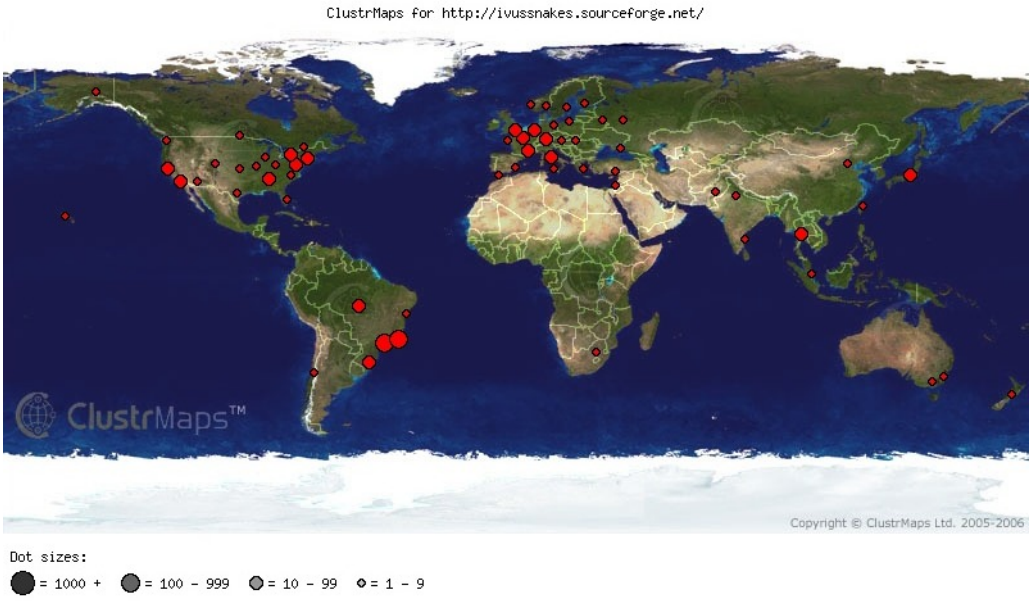


FIGURE 3.4 – Visitors of *http://ivussnakes.sourceforge.net*

Since all these features were available, the plugin *LiveWire* was distributed through the project link *http://sourceforge.net/projects/ivussnakes/*, while the official web site was *http://ivussnakes.sourceforge.net/*. Figure 3.4 shows where visitors of the site come from.

In download section of the site *http://sourceforge.net/projects/ivussnakes/*, all the source code of the plugins is available as well as in CVS repository, also accessible from the site.

4 Discussion

Since images in section 3.1 have been tested against a *gold-standard*, the analysis of those segmentations will be focused in this section.

4.1 Noiseless Images

It was observed in table 3.3 that *Snakes* true positives for noiseless images was 91.07%, while, as observed in table 3.2, *LiveWire*'s results were better, with 98.04%. It must be noticed throughout this section that the time taken by *LiveWire* is considerably larger, taking hours, while *Snakes* method takes just a few minutes, for hundreds of frames.

The fact that made *Snakes* perform worse was that its segmentations have converged to the inner part of the border, as shown by one of *Snakes*'s false negative frames, in figure 4.1.

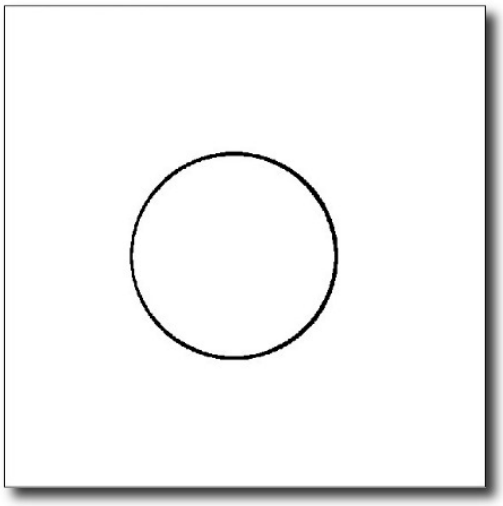


FIGURE 4.1 – False negative for one of the frames segmented using *Snakes* in noiseless images

Although *LiveWire* has shown better results, it should be noticed that *Snakes* has

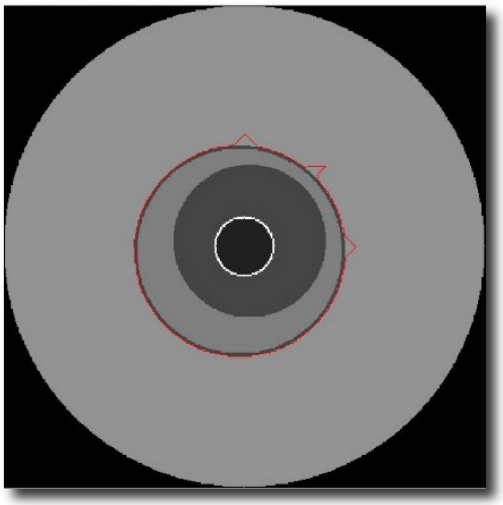


FIGURE 4.2 – False positive segmentation in one of the frames, using *LiveWire*

not shown a single pixel wrongly in false positives. On the other hand, *LiveWire* has 0.09% of false positives. It was due to *LiveWire*'s deficiency in dealing with wrong initial points generated from longitudinal segmentation. Figure 4.2 shows what happens when the initial point is too far from where it should be, what may happen for sharp transitions along longitudinal segmentation. It should be also noticed that *LiveWire* points have correctly sticked to the external part of the border.

4.2 Images with speckle noise

It is clear that simulated speckle noise has made segmentation more difficult. As seen in table 3.4, the percentage of true positives for *LiveWire* has decreased from 98.04% to 96.01%. The difference is due to the fact that some parts of the selection have sticked to the lumen region, as seen in figure 4.3. This fact justifies the decay.

While *LiveWire* had a small difference of only 2 percent between noiseless and speckle images, *Snakes* has shown a drastic change from 91.07% to 2.37% of true positives in images with speckle noise. The difference can be easily explained through figure 4.4, which shows one of the *Snakes* selections that has adhered to the external border region, due to the noise in the desired region. One might ask himself why aren't there 100% true positives, since the selection shown in figure 4.4 seem to involve the gold standard area. Actually what happens is that there are two curves collapsed in each other, so they look like only one. If their inner area was highlighted, only a thin layer inside the selection

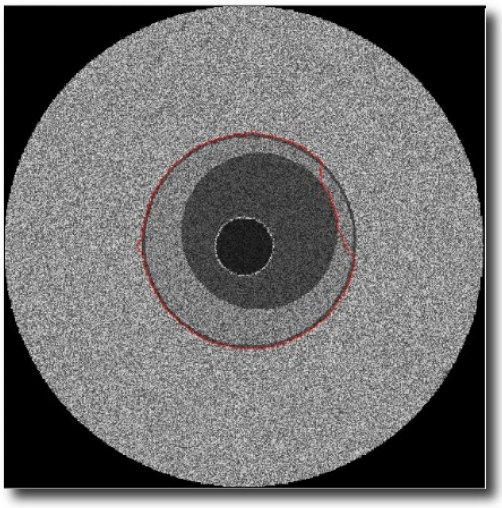


FIGURE 4.3 – *LiveWire* selection parts have adhered to lumen area in speckle noise images

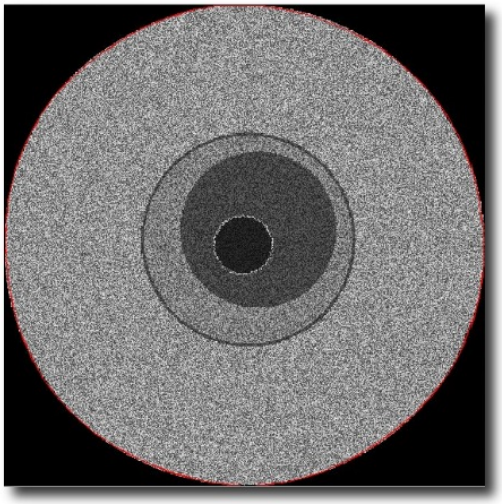
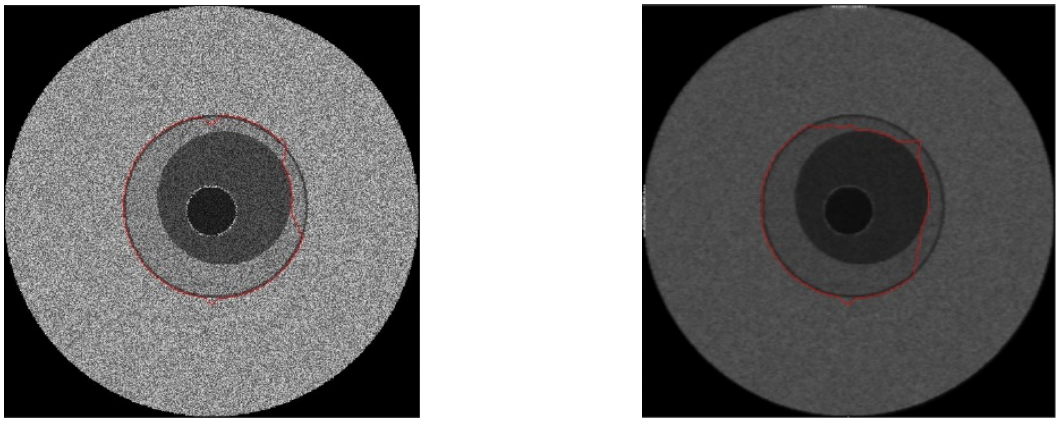


FIGURE 4.4 – *Snake* selections have adhered to external border due to noise

shown in figure 4.4 would be shown.

4.3 Filtered Images

One odd fact observed in filtered *LiveWire* segmentations, was that the percentage of true positives in images with speckle noise was higher (96.01%) than the filtered ones (94.14%). A good explanation for this fact is obtained from figure 4.5. On the left side, the same image as the one on the right side is shown during speckle segmentation, and there was practically no change because of the wrong initial point set in longitudinal segmentation, in the upper part. Figure 4.5b, however, shows that a smaller cost path



(a) *LiveWire* segmentation with noise (b) Filtered *LiveWire* segmentation

FIGURE 4.5 – Comparison of *LiveWire* segmentation with noise and filtered images

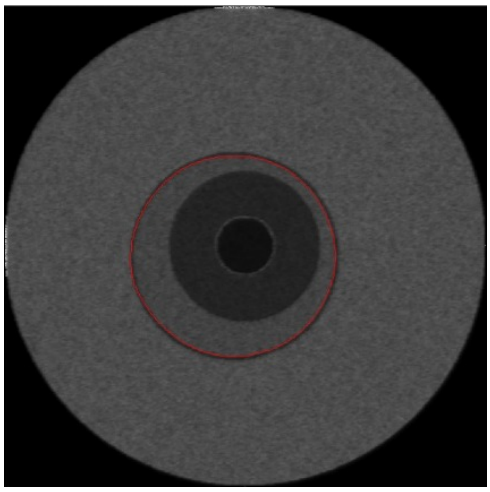


FIGURE 4.6 – Segmentation of filtered image obtained through *Snakes* technique

was found, hence, a smaller area was compared to the *gold-standard* what resulted in a worse segmentation for filtered images, using *LiveWire*. One way to go around this problem would be changing the weights used for the segmentation.

Snakes segmentation true positive percentage has increased significatively, from 2.37% to 92.67%. It is clear from figure 4.6 that the use of filtering has allowed *Snakes* selections to correctly adhere to the desired region of interest. Another fact that should be noticed is that *Snakes* don't suffer as much as *LiveWire* from incorrect position of initial points, retrieved from longitudinal segmentation.

5 Future work

During section 4, it was clear that both techniques have advantages in several different fields. While *Snakes* is fast and is not highly affected by wrong initial points, *LiveWire* has better segmentation results and works finely with noisy images. A good suggestion for future work would be implementing techniques that combine the best traits of them. An interesting idea was discussed in (LIANG; MCINERNEY; TERZOPOULOS, 2006), and states that the two techniques are in fact complementary and can advantageously be combined by introducing an effective hard constraint mechanism.

Another good point to work on would be the optimization of *LiveWire* technique. The implementation used in this thesis uses a better algorithm than the original one, but there are still better approaches as pointed by (FALCAO; UDUPA; MIYAZAWA, 2000) – algorithms of $O(m\ log\ log\ C)$ are available for finding the shortest path, where m is the number of edges in the graph and C is the edge's highest cost. Besides implementing faster algorithms, other points may be enhanced, like not plotting the results on the screen, since they can go to a backbuffer and only show the results.

Since only the default parameters were used on *Snakes*, the study of different ones could improve the results of this technique. An elegant way to specify them would be training a *Neural Network* on specific IVUS images.

6 Conclusion

The proposed goal of making the adventitia segmentation in three-dimensional coronary intravascular ultrasound simulated images, using LiveWire and Snakes techniques was successfully achieved. This chapter shows the main conclusions about both methods used during segmentation. Table 6.1 will be discussed in more detail throughout the chapter.

TABLE 6.1 – Table comparing *LiveWire* and *Snakes* techniques

	LiveWire	*Snakes*
Noise	slightly affected	highly affected
Segmentation quality	better (94.14%)	worse (92.67%)
Computational cost	high (8 hours)	low (1 minute)
Wrong initial points	highly affected	slightly affected

Chapter 4 has shown enough reasons to prove that *LiveWire* suffered less from noise than *Snakes*. This is particularly clearer in section 4.2, in which *Snakes* show true positive percentage of about 2 percent, while *LiveWire*'s is practically not affected.

In order to obtain better results with *Snakes*, the number of subdivision levels, as well as the type of *Snake* and its parameters should be tested with different values. An elegant way to train the parameters would be using *Neural Networks*.

Another point to be observed is that *LiveWire* presented better segmentation results in all types of images, specially in noisy ones, although there wasn't much difference in filtered images.

A bad aspect related to *LiveWire* is that its computational cost is too high. While segmentation of 150 frames using *Snakes* takes around one minute, *LiveWire* segmentation takes hours. This is extremely not desired if the number of frames is increased, which actually happens frequently. Nevertheless, the time taken in *LiveWire* segmentation is

directly proportional to the number of frames, which means that if the number of frames is doubled, the time is also doubled. Although many hours are taken during the process, it should be noticed that there is no need for the operator to be in front of the computer. The results might be processed during computer's idle time, when the specialist is doing something else, like another exam. If many computers are available in the segmentation site, lots of exams may be run in parallel. Another way to decrease the process time would be using distributed computation, like a grid or cluster. The algorithm of segmentation can be extremely parallelized, since each frame of segmentation does not depend on the previous. An easy way to do such distribution would be giving one frame for each computer, while the whole process would be coordinated by a central machine, retrieving the results to the operator. It should be noticed that no optimization was done to the implemented algorithm, since it would work as a proof of concept.

On the other hand, *Snakes* results were obtained through a faster algorithm. Since it still does not give better results than *LiveWire* it could be used when rough analysis were made, while the real analysis should be made by *LiveWire*.

An interesting point observed while using *Snakes* was that it is not much affected by wrong initial points, while *LiveWire* is extremely affected. A good approach would be using some algorithm as the one described in chapter 5.

All techniques developed through this thesis have given a strong background to the author. Using Mathematics, Software Engineering, Computer Graphics, Open Source Technologies, Content Management Systems – Drupal CMS was used to publish the website –, and many other new technologies, the author was able to give the world community a piece of software that is easy to use and extremely important in the field of Image Segmentation. The work counted with the participation of the Associate Professor Tim McInerney, from the Department of Computer Science of Ryerson University, in Canada, who helped very much with Java Extensible Snakes System development and support. Until the date of publication of this work, November, 23th, 2006, there has been more than 300 downloads of the software, which proves it is being used by the world community.

Although this thesis describes the validation process over simulated images, an effort has been made to verify how the segmentations would perform against the *gold-standard* of real images. When this process is finished, the software might be used to assess coronary

segmentation in real centers that study heart diseases, like InCor, and then it will be able to help save lives.

Bibliography

3D Imaging in Medicine. [S.l.]: CRC Press, 1991.

ABDOU, I.; PRATT, W. K. Quantitative design and evaluation of enhancement/thresholding edge detectors. In: **Proceedings of the IEEE**. [S.l.: s.n.], 1979. p. 753–763.

ABIZAID, A.; MINTZ, G. S.; PICHARD, A. D.; KENT, K. M.; SATLER, L. F.; POPMA, J. J.; LEON, M. B. Is intravascular ultrasound clinically useful or is it just a research tool? **Heart**, 1997.

BAILER, W. **Writing ImageJ Plugins – A Tutorial**. [S.l.], July 2006.

BAMBER, J. C. Image formation and image processing in ultrasound. Joint Department of Physics, Institute of Cancer Research and The Royal Marsden NHS Trust, Downs Road, Sutton, Surrey, SM2 5PT, UK. 2002.

BARBEHENN, M. A note on the complexity of dijkstra's algorithm for graphs with weighted vertices. **IEEE Transactions on Computers**, IEEE Computer Society, Los Alamitos, CA, USA, v. 47, n. 2, p. 263, 1998. ISSN 0018-9340.

BRIGUORI, C.; TOBIS, J.; NISHIDA, T.; VAGHETTI, M.; ALBIERO, R.; MARIO, C. D.; COLOMBO, A. Discrepancy between angiography and intravascular ultrasound when analysing small coronary arteries. **Eur Heart J**, v. 23, n. 3, p. 247–254, 2002.

CAN, A.; TURNER, J.; TANENBAUM, H.; ROYSAM, B. **Rapid automated tracing and feature extraction from live high-resolution retinal fundus images using direct exploratory algorithms**. 1999.

COHEN, B. R. A. R.; TURNER, J. N. Automated tracing and volume measurements of neurons from 3-d confocal fluorescence microscopy data. **Journal of Microscopy**, v. 173, n. 2, p. 103–114, February 1994.

DIJKSTRA, E. W. A note on two problems in connexion with graphs. **Numerische Mathematik**, v. 1, p. 269–270, 1959.

EHARA, S.; KOBAYASHI, Y.; YOSHIYAMA, M.; SHIMADA, K.; SHIMADA, Y.; FUKUDA, D.; NAKAMURA, Y.; YAMASHITA, H.; YAMAGISHI, H.; TAKEUCHI, K.; NARUKO, T.; HAZE, K.; BECKER, A. E.; YOSHIKAWA, J.; UEDA, M. Spotty calcification typifies the culprit plaque in patients with acute myocardial infarction: An intravascular ultrasound study. **Circulation**, v. 110, n. 22, p. 3424–3429, 2004.

EILBERT, C. G. J.; MCEACHRON, D. The variation in user drawn outlines on digital images: Effects on quantitative autoradiography. **Compur. Med. Imag. Graphics**, v. 14, n. 5, p. 331–339, 1990.

FALCAO, A.; UDUPA, J.; SAMARASEKERA, S.; SHARMA, S. User-steered image segmentation paradigms: Live wire and live lane. **Graphical Models and Image Processing**, 1998.

FALCAO, A.; UDUPA, J. K.; MIYAZAWA, F. K. An ultra-fast user-steered image segmentation paradigm: live wire on the fly. **IEEE Transactions on Medical Imaging**, v. 19, n. 1, p. 55–62, January 2000.

FALCÃO, A. X. **Paradigmas de Segmentação de Imagens Guiada pelo Usuaário: Live-Wire, Live-Lane e 3D-Live-Wire**. Tese (Doutorado) — Universidade Estadual de Campinas, 1997.

FALCÃO, A. X.; UDUPA, J. K.; SAMARASEKERA, S.; SHARMA, S.; HIRSCH, B. E.; LOTUFO, R. de A. User-steered image segmentation paradigms: live wire and live lane. **Graph. Models Image Process.**, Academic Press, Inc., Orlando, FL, USA, v. 60, n. 4, p. 233–260, 1998. ISSN 1077-3169.

GERIG, G.; MARTIN, J.; KIKINIS, R.; KÜBLER, O.; SHENTON, M. E.; JOLESZ, F. A. Automating segmentation of dual-echo mr head data. In: **IPMI '91: Proceedings of the 12th International Conference on Information Processing in Medical Imaging**. London, UK: Springer-Verlag, 1991. p. 175–187. ISBN 3-540-54246-9.

GOLDBERG, A. V.; SILVERSTEIN, C. Implementations of dijkstra's algorithm based on multi-level buckets. NEC Research Institute and Stanford University Computer Science Department. November 1995.

HAUSMANN, D.; ERBEL, R.; ALIBELLI-CHEMARIN, M.-J.; BOKSCH, W.; CARACCIOLO, E.; COHN, J. M.; CULP, S. C.; DANIEL, W. G.; SCHEERDER, I. D.; DIMARIO, C.; FERGUSON, I.; J., J.; FITZGERALD, P. J.; FRIEDRICH, G.; GE, J.; GORGE, G.; HANRATH, P.; HODGSON, J. M.; ISNER, J. M.; JAIN, S.; MAIER-RUDOLPH, W.; MOONEY, M.; MOSES, J. W.; MUDRA, H.; PINTO, F. J.; SMALLING, R. W.; TALLEY, J. D.; TOBIS, J. M.; WALTER, P. D.; WEIDINGER, F.; WERNER, G. S.; YEUNG, A. C.; YOCK, P. G. The safety of intracoronary ultrasound : A multicenter survey of 2207 examinations. **Circulation**, v. 91, n. 3, p. 623–630, 1995.

HONG, M.-K.; MINTZ, G. S.; LEE, C. W.; KIM, Y.-H.; LEE, S.-W.; SONG, J.-M.; HAN, K.-H.; KANG, D.-H.; SONG, J.-K.; KIM, J.-J.; PARK, S.-W.; PARK, S.-J. Comparison of coronary plaque rupture between stable angina and acute myocardial infarction: A three-vessel intravascular ultrasound study in 235 patients. **Circulation**, v. 110, n. 8, p. 928–933, 2004.

INTRODUCTION to Algorithms. [S.l.]: MIT Press and McGraw-Hill, 2001.

JAVA 2 Platform SE 5.0 Class PriorityQueue<E>. [S.l.], 2004.

LEE, J. S. Digital image enhancement and noise filtering by use of local statistics. **IEEE Trans Pattern Anal Mach Intell**, v. 2, n. 2, p. 165–168, 1980.

LIANG, J.; MCINERNEY, T.; TERZOPOULOS, D. United snakes. **Medical Image Analysis**, v. 10, n. 215–233, 2006.

LINKER, D.; BATKOFF, B. W. Safety of intracoronary ultrasound : Data from a multicenter european registry. **Catheterization and cardiovascular diagnosis**, v. 38, n. 3, p. 238–241, 1996.

LOBREGT, S.; VIERGEVER, M. A. A discrete dynamic contour model. **IEEE Transactions on Medical Imaging**, v. 14, n. 1, p. 12–24, March 1995.

MARR, D.; HILDRETH, E. Theory of edge detection. In: **Proceedings of the Royal Society of London**. [S.l.: s.n.], 1980. p. 187–217.

MCINERNEY, T.; SHARIF, M. R. A.; PASHOTANIZADEH, N. Jess: Java extensible snake system. **Medical Imaging 2005: Image Processing**, 2005.

MINTZ, G. S.; PAINTER, J. A.; PICHARD, A. D.; KENT, K. M.; SATLER, L. F.; POPMA, J. J.; CHUANG, Y. C.; BUCHER, T. A.; SOKOLOWICZ, L. E.; LEON, M. B. Atherosclerosis in angiographically "normal" coronary artery reference segments: an intravascular ultrasound study with clinical correlations. **J Am Coll Cardiol**, v. 25, n. 7, p. 1479–1485, 1995.

MORTENSEN, E. N.; BARRETT, W. A. Intelligent scissors for image composition. In: **SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques**. New York, NY, USA: ACM Press, 1995. p. 191–198. ISBN 0-89791-701-4.

MORTENSEN, E. N.; BARRETT, W. A. Interactive segmentation with intelligent scissors. **Graph. Models Image Process.**, v. 60, n. 5, p. 349–384, 1998.

NISHIHARA, H. K.; CROSSLEY, P. A. Measuring photolithographic overlay accuracy and critical dimensions by correlating binarized laplacian of gaussian convolutions. **Pattern Analysis and Machine Intelligence, IEEE Transactions on**, v. 10, n. 1, p. 17–30, 1988.

NISSEN, S. E.; YOCK, P. Intravascular ultrasound : Novel pathophysiological insights and current clinical applications. **Circulation**, v. 103, n. 4, p. 604–616, 2001.

PARK, J. M.; SONG, W. J.; PEARLMAN, W. A. Speckle filtering of sar images based on adaptive windowing. In: **Vision, Image and Signal Processing**. [S.l.: s.n.], 1999. v. 146, n. 4, p. 191–107.

PRESS, W. H.; TEUKOLSKY, S. A.; VETTERLING, W. T.; FLANNERY, B. P. **Numerical Recipes in C**. [S.l.]: Cambridge University Press, 2002.

SALES, F. J. R. Segmentação 2d do lúmen e adventícea da íntima da coronária em imagens de ivus usando contornos ativos do tipo snakes. Monography of the course: PTC5750 - Tópicos Avançados em Processamento de Imagens Médicas. 2006.

TELECOMMUNICATIONS: Glossary of Telecommunication Terms, Federal Standard 1037. [S.l.], 1996.

THE Free Software Definition. Http://www.gnu.org/philosophy/free-sw.html.

TSUI, P.-H.; WANG, S.-H.; HUANG, C.-C. The effect of logarithmic compression on estimation of the nakagami parameter for ultrasonic tissue characterization: a simulation study. **Physics in Medicine and Biology**, v. 50, p. 3235–3244, 2005.

ZONNEVELD, F. W.; FUKUTA, K. A decade of clinical three-dimensional imaging: A review. part i. historical development. **Investigative Radiology**, 1994.

Appendix A - GNU General Public License

The GNU General Public License

Version 2, June 1991

51 Franklin St, Fifth Floor, Boston, MA 02110-1301, USA

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU General Public License
Terms and Conditions For Copying, Distribution and Modification

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   (a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

(b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

(c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

(a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

(b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

(c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source

code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

   If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

   It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system,

which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

   Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

No Warranty

11. Because the program is licensed free of charge, there is no warranty for the program, to the extent permitted by applicable law. Except when otherwise stated in writing the copyright holders and/or other parties provide the program "as is" without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The entire risk as to the quality and performance of the program is with you. Should the program prove defective, you assume the cost of all necessary servicing, repair or correction.

12. In no event unless required by applicable law or agreed to in writing will any copyright holder, or any other party who may modify and/or redistribute the program as permitted above, be liable to you for damages, including any general, special, incidental or consequential damages arising out of the use or inability to use the

PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

End of Terms and Conditions

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

> <one line to give the program's name and a brief idea of what it does.>
> Copyright (C) <year> <name of author>
>
> This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.
>
> This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
>
> You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301, USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

> Gnomovision version 69, Copyright (C) <year> <name of author>
> Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.
> This is free software, and you are welcome to redistribute it under certain conditions; type 'show c' for details.

The hypothetical commands `show w` and `show c` should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w` and `show c`; they could even be mouse-clicks or menu items—whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

> Yoyodyne, Inc., hereby disclaims all copyright interest in the program
> 'Gnomovision' (which makes passes at compilers) written by James Hacker.
>
> <signature of Ty Coon>, 1 April 1989
> Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

Annex A - Big O Notation

Big-O notation, Landau notation or asymptotic notation, is used mainly by computer scientists to mathematically describe the asymptotic behavior of functions. It is intended to compare different algorithms in a rigorous way. The precise definition is that the symbol O is used to describe an upper bound for the magnitude in terms of another simpler function. This is especially useful in computer science to evaluate the complexity of algorithms.

The Big-O notation expresses the runtime of an algorithm as a function of a given input of size n. For example, $O(n)$ means that the algorithm grows linearly with the number of elements – doubling the input doubles the runtime. When using the Big-O notation, constants and factors with smaller exponents are hidden in the formula, so that only the highest element is shown. For instance, if the running time is proportional to $n^2 + n$ this would be represented by the following Big-O: $O(n^2)$. Table A.1 shows the most common types of function orders.

TABLE A.1 – Big-O order of most common types of functions

Notation	Name	Example
$O(1)$	constant	Determining if a number is even or odd (checking last bit)
$O(log\ n)$	logarithmic	Finding an item in a sorted list
$O(n)$	linear	Finding an item in an unsorted list (checking all values)
$O(n\ log\ n)$	linearithmic	Sorting a list with heapsort algorithm
$O(n^2)$	quadratic	Sorting a list with insertion sort
$O(c^n)$	exponential	Finding the exact solution to the travelling salesman problem

FOLHA DE REGISTRO DO DOCUMENTO

1. CLASSIFICAÇÃO/TIPO	2. DATA	3. DOCUMENTO N	4. N DE PÁGINAS
TC	21 de novembro de 2006	CTA/ITA - IEC/TC-009/2006	75

5. TÍTULO E SUBTÍTULO:

Intravascular Ultra Sound Image Segmentation Algorithms

6. AUTOR(ES):

Daniel Lélis Baggio

7. INSTITUIÇÃO(ÕES)/ÓRGÃO(S) INTERNO(S)/DIVISÃO(ÕES):

Instituto Tecnológico de Aeronáutica. Divisão de Engenharia da Computação – ITA/IEC

8. PALAVRAS-CHAVE SUGERIDAS PELO AUTOR:

Segmentation; LiveWire; Snakes; ultrasound; intravascular; coronary; ImageJ; Speckle

9. PALAVRAS-CHAVE RESULTANTES DE INDEXAÇÃO:

Image Processing ; ultrasound; cardiology; algorithms; parameter identification; Speckle patterns; filtering; Electronic Engineering; Medicine

10. APRESENTAÇÃO: ( ) **Nacional** (**X**) **Internacional**

ITA, São José dos Campos, 2006, 75 páginas

11. RESUMO:

This graduation thesis deals with intravascular ultrasound (IVUS) image segmentation through *LiveWire* and *Snakes* algorithms. As the type of ultrasound studied is the one from coronaries, there's an initial detailing of the medical environment in which the problem is inserted.

The introduction is followed by an explanation of how *LiveWire* technique works, which is by calculating the shortest cost path between two pixels of the image. There's a discussion about the implementation, using binary trees and a heap data structure. This is followed by the evaluation of which costs should be adopted, which were: modulus and direction of the gradient, the laplacian and a non-linear function.

Then, *Snakes* algorithm is described, which simulates a set of vertices submitted to one internal field of energy and another external one. There's also a discussion about the number of iterations to be adopted as well as the values used as parameters.

It is explained, so, how simulated images, used for evaluation, were generated. Besides, a model for *Speckle* noise is described as well as the filter to which the images were submitted.

There's a brief discussion on how the algorithms were implemented, with a special focus on *ImageJ* platform and it's extensibility features.

The results are, then, described, with each of the simulated image types: noiseless, with *Speckle* noise and filtered. The main results found are that *LiveWire* has better segmentation quality and is slightly affected by noise, while *Snakes* is faster and is practically not affected by wrong initial points retrieved from longitudinal segmentation.

There's, then, the conclusion, describing what was found, focusing the high velocity of *Snakes* technique, followed by good results. Although better results were found for *LiveWire*, the processing time was about 500 times higher, what might indicate this method is not desirable. Due to the good results found with simulated images, the developed platform promises to work efficiently in a medical environment, after submitted to validation by specialists.

12. GRAU DE SIGILO:

(**X**) **OSTENSIVO** ( ) **RESERVADO** ( ) **CONFIDENCIAL** ( ) **SECRETO**