DUBLIN CITY UNIVERSITY

SCHOOL OF ELECTRONIC ENGINEERING

Implementation of anisotropic diffusion for removal of noise from Chest CT data

Petru-Stefan Manescu

August 2009

MASTER OF ENGINEERING

IN

ELECTRONIC SYSTEMS

Supervised by Prof. Paul Whelan

Acknowledgements

I would like to thank my supervisor Prof. Paul Whelan for his guidance, enthusiasm and commitment to this project. I would also like to express my deep appreciation to Dr. Tarik Chowdhury and Dr. Ovidiu Ghita for who have been very supportive during this project and have given me precious advice.

Declaration

I hereby declare that, except where otherwise indicated, this document is entirely my own work and has not been submitted in whole or in part to any other university.

Signed: ..... Date: .....

Abstract

In this paper the implementation of anisotropic diffusion-based smoothing scheme and its application to medical 3D data is investigated. Chest CT datasets are characteristically defined by a low signal-to-noise ratio (SNR) which can increase the false positive rate of the Computer Assisted Diagnostic (CAD) system, and in turn the probability of missing small nodules from lung CT data sets. To limit these errors, anisotropic diffusion filtering implements an optimal, feature preserving smoothing strategy. Results of anisotropic diffusion filtering depend mainly on 2 parameters which can seriously affect the diffusion process: the diffusion parameter K and the artificial time parameter (number of iterations). In this paper, the diffusion parameter is calculated in order to maximize the signal-to-noise ratio from the image using robust statistics tools while the number of iterations is chosen so that the diffusion process stops when the small features of interest become altered. Additionally, computational time is reduced using two simple techniques. Results show significant improvements compared to median filtering.

Table Of Contents

Table of Figures

1. Introduction

When developing automated Computer Assisted Diagnostic (CAD) techniques the errors introduced by the image noise are not acceptable. Movements of the lung and of the body create noise in the chest CT data. The high level of noise which is due also to low-dose acquisition can increase the false positive rate of the CAD system, and in turn the probability of missing small nodules from lung CT data sets. Thus, to limit these errors, a solution is to filter the data in order to increase the signal-to-noise ratio (SNR). More importantly, the image filtering technique should be able to reduce the level of noise, but not at the expense of feature preservation [2].

Classical noise removal techniques, like Gaussian low-pass filters process the original fine-scale image, generating simplified coarse-scale images. Unfortunately, coarse-scale images generated by Gaussian filters present blurred edges that do not spatially match the original edges [3]. Anisotropic diffusion was originally developed by Perona and Malik [5] in order to implement an optimal, feature preserving smoothing strategy.

The behavior of the anisotropic diffusion depends on two parameters: the artificial time parameter t and the gradient thresholding parameter K. The appropriate choice of these parameters is essential to obtain a conveniently filtered image. Relations between the anisotropic diffusion and the robust statistics have been established in order to calculate automatically the diffusion parameter K, leading to the robust anisotropic diffusion (RAD) that preserves sharper boundaries than previous techniques [1].

Most of the research has been carried out on the automatic calculation of the diffusion parameter K, ignoring the time parameter because usually the diffusion is iterated until convergence. It is not the case with medical images used in the CAD systems where even the smallest features need to be preserved and where the computational time needs to be reduced as much as possible.

In this report, robust anisotropic diffusion is applied to medical 3D datasets and a simple stopping condition is developed, reducing thus the computational time and preservising all the features in the image. This report is organized as follows: Chapter Two introduces the concept of anisotropic diffusion filtering and presents the technical background, chapter three presents the filter design and the choice of the several parameters of anisotropic diffusion, chapter four describes the actual implementation of the filter while chapter five shows the results obtained. Chapter 6 concludes this paper mentioning a few directions in which further research can be done.

2. Technical background

Many mathematicians have been attracted by image processing and computer vision in recent years. This has been triggered by mathematically well-founded methods using e.g. wavelets or nonlinear partial differential equations [6]. In a survey on image smoothing techniques the approaches encountered may be classified under two broad headings, linear and non-linear. Standard linear smoothing techniques based on local averaging or Gaussian weighted spatial operators reduce the level of noise, but this is achieved at the expense of poor feature preservation. Consequently, the filtered data appears blurry as step intensity discontinuities such as edges are attenuated. To compensate for these undesirable effects, non-linear and adaptive techniques have been developed in order to achieve better feature preservation [2]. Diffusion algorithms remove noise from an image by modifying the image via a partial differential equation (PDE).

This chapter is organized as follows: Section One describes briefly the physical background of diffusion processes while Section Two gives an introduction to anisotropic diffusion. In Section Three the discrete formulation of anisotropic diffusion is presented and Section Four shows some examples and applications.

2.1 Physical background of diffusion processes

Diffusion can be explained intuitively as a physical process that equilibrates concentration differences without creating or destroying mass. This physical observation can be easily cast in a mathematical formulation. The equilibration property is expressed by Fick's law:

$$j = -D \cdot \nabla u \qquad (1)$$

This equation states that a concentration gradient ∇u causes a flux j which aims to compensate for this gradient. The relation between ∇u and j is described by the diffusion

tensor D, a positive definite symmetric matrix. The case where j and ∇u are parallel is called isotropic. Then we may replace the diffusion tensor by a positive scalar-valued diffusivity g. In the general anisotropic case, j and ∇u are not parallel. The observation that diffusion does only transport mass without destroying it or creating new mass is expressed by the continuity equation

$$\partial_t u = -\mathrm{div}\, j \qquad (2)$$

where t denotes the time. If we plug in Fick's law into the continuity equation we end up with the diffusion equation

$$\partial_t u = \mathrm{div}\,(D \cdot \nabla u). \qquad (3)$$

This equation appears in many physical transport processes. In the context of heat transfer it is called heat equation. In image processing we may identify the concentration with the grey value at a certain location. If the diffusion tensor is constant over the whole image domain, one speaks of homogeneous diffusion, and a space-dependent filtering is called inhomogeneous. Often the diffusion tensor is a function of the differential structure of the evolving image itself. Such a feedback leads to nonlinear diffusion filters. Diffusion which does not depend on the evolving image is called linear. Sometimes the computer vision literature deviates from the preceding notations: It can happen that homogeneous filtering is named isotropic, and inhomogeneous blurring is called anisotropic, even if it uses a scalar-valued diffusivity instead of a diffusion tensor [4].

2.2 Anisotropic diffusion

As mentioned before, diffusion algorithms use partial differential equations to remove noise from images. Isotropic diffusion is based on the heat equation given by $\partial I(x,y,t)/\partial t = \mathrm{div}(\nabla I)$, using the original (degraded/noisy) image as the initial condition, where $I(x,y,0)$ is an image in the continuous domain, (x,y) specifies spatial position, t is an artificial time parameter, and where ∇I is the image gradient. Modifying the image

according to this isotropic diffusion equation is equivalent to filtering the image with a Gaussian filter.

Perona and Malik [5] replaced the classical isotropic diffusion equation with:

$$\frac{\partial I(x,y,t)}{\partial t} = div[g(\|\nabla I\|\nabla I)] \qquad (4)$$

Where ∇I is the gradient magnitude, and *g(.)* is an "edge-stopping" function. This function is chosen to satisfy *g(x) -> 0* when *x->∞* so that the diffusion is "stopped" across edges [1]. They suggested using one of the two edge-stopping functions below:

$$g_1(x) = \frac{1}{1+\frac{x^2}{K^2}}, g_2(x) = e^{-\frac{x^2}{2K^2}} \qquad (5)$$

The right choice of the edge-stopping function g can greatly affect the extent to which discontinuities are preserved. Black et al. [1] chose a different edge-stopping function which produces sharper boundaries than the previous ones, called Tukey's biweight:

$$g_3(x) = \begin{cases} [1 - \frac{x^2}{5K^2}], & \frac{x^2}{5} < K^2 \\ 0, & otherwise \end{cases} \qquad (6)$$

2.3 Discrete formulation

Perona and Malik discretized spatio-temporally their anisotropic diffusion equation as:

$$I(s, t+1) = I(s,t) + \frac{\lambda}{|\eta_s|}\sum_{p\in\eta_s} g(|\nabla I_{s,p}(t)|)\nabla I_{s,p}(t) \qquad (7)$$

where I(s, t) is a discretely sampled image, s denotes the pixel position in a discrete 2-D grid, $t \geq 0$ now denotes discrete time steps, the constant λ determines the rate of diffusion (usually, λ = 1), and η_s represents the set of spatial neighbors of pixel s. For 1-D signals,

usually two neighbors are considered: *left* and *right*, except at signal boundaries where only one neighbor must be considered. For 2-D images, usually 4-neighborhood is used: *north*, *south*, *west* and *east*, except at the image boundaries [3] whereas for 3-D images 6 neighbors are taken into consideration [2].

The gradient of an image measures how it is changing. It provides two pieces of information. The magnitude of the gradient tells us how quickly the image is changing, while the direction of the gradient tells us the direction in which the image is changing most rapidly. Because the gradient has a direction and a magnitude, it is natural to encode this information in a vector. The length of this vector provides the magnitude of the gradient, while its direction gives the gradient direction. Because the gradient may be different at every location, we represent it with a different vector at every image location. For a 2-D image the gradient vector is formed by combining the partial derivative of the image in the *x* direction and the *y* direction [7]. This can be written as:

$$\nabla I = \left(\frac{\partial I}{\partial x}, \frac{\partial I}{\partial x}\right)$$

Perona and Malik approximated the image gradient magnitude in a particular direction at iteration *t* as described in [3]:

$$\nabla I_{s,p}(t) = I(p,t) - I(s,t), p \in \eta_s \qquad (8)$$

2.4 Examples and applications

Anisotropic diffusion filtering is a very popular preprocessing technique used in various fields ranging from applications in medical image analysis to problems in computer aided quality control.

2.4.1 Denoising by edge-enhancing anisotropic diffusion

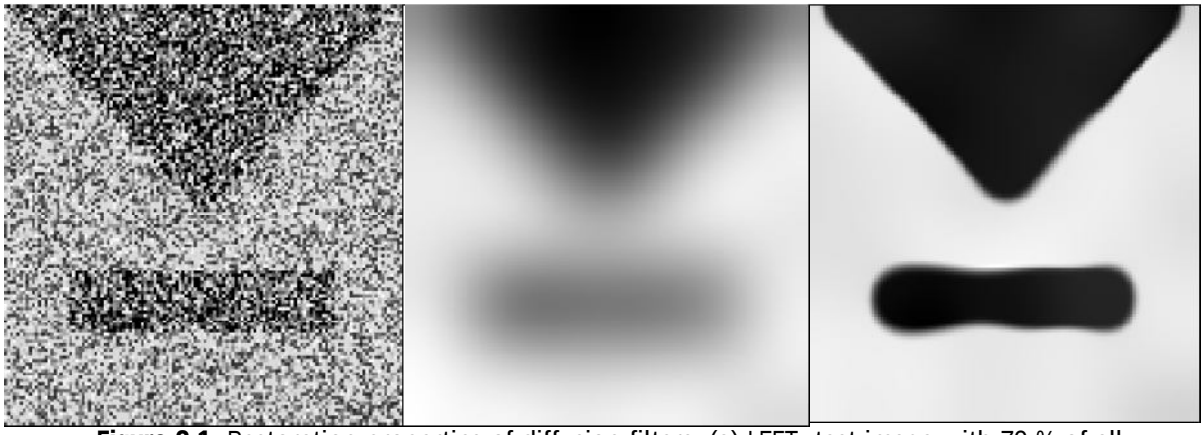


Figure 2.1. Restoration properties of diffusion filters. (a) LEFT: test image with 70 % of all pixels degraded. (b) CENTER: Linear diffusion, 80 iterations. (c) RIGHT: Nonlinear anisotropic diffusion, 80 iterations [4].

In Fig. 2.1(b) we observe that linear diffusion filtering is capable of removing all noise, but we have to pay a price: the image becomes completely blurred. Besides the fact that edges get smoothed so that they are harder to identify, the correspondence problem appears: edges become dislocated. Figure 2.1(c) demonstrates that nonlinear anisotropic filtering combines the good noise eliminating properties of linear diffusion with the stable edge structure of nonlinear isotropic filtering. Due to the permitted smoothing along edges, however, corners get rounded [4].

2.4.2 Applications in computer aided quality control

In this section, two applications of nonlinear diffusion filtering in computer aided quality control are studied: the grading of fabrics and wood surfaces.

The quality of a fabric is determined by two criteria, namely clouds and stripes. Clouds result from isotropic inhomogeneities of the density distribution, whereas stripes are an anisotropic phenomenon caused by adjacent fibres pointing in the same direction. Anisotropic diffusion filters are capable of visualizing both quality relevant features simultaneously (Fig. 5.7). For a suitable parameter choice, they perform isotropic smoothing at clouds and diffuse in an anisotropic way along fibres in order to enhance them [4].

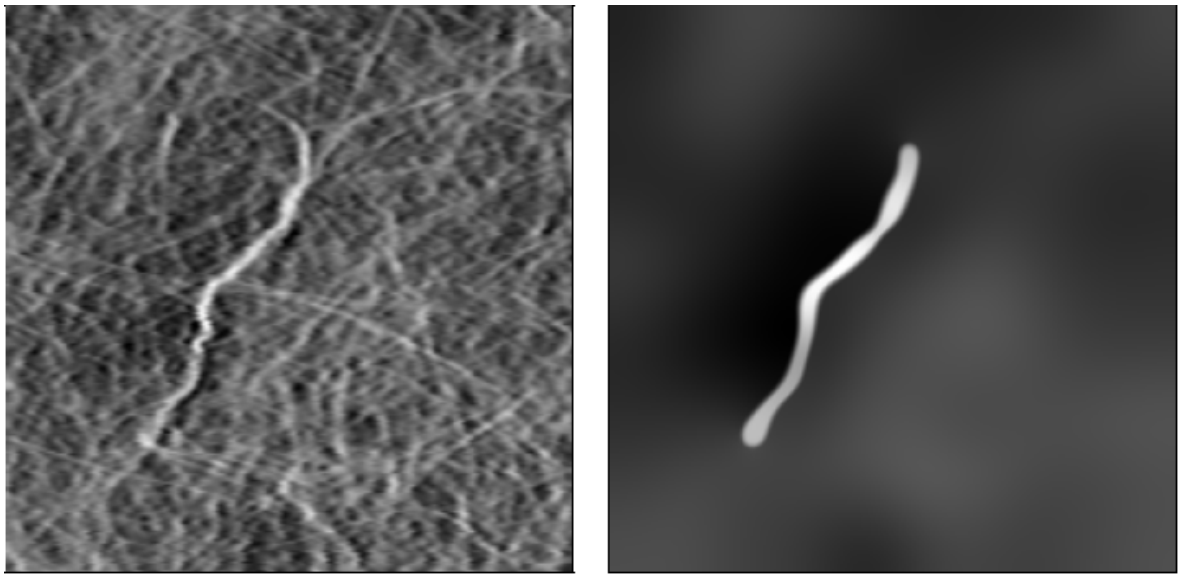


Figure 2.2 Preprocessing of a fabric image. (a) LEFT: Fabric. (b) RIGHT: Anisotropic diffusion, 240 iterations

For furniture production it is of importance to classify the quality of wood surfaces. If one aims to automize this evaluation, one has to process the image in such a way that quality relevant features become better visible und unimportant structures disappear.

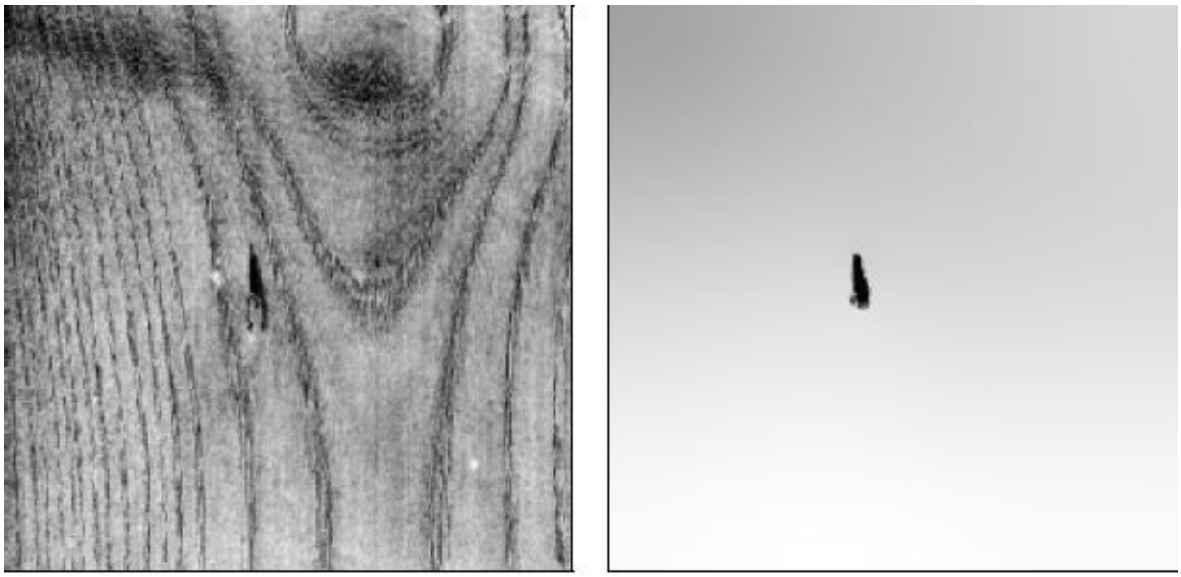


Figure 2.3 Defect detection in wood. (a) LEFT: Wood surface. (b) RIGHT: Isotropic nonlinear diffusion, 2000 iterations

2.4.3 Applications in image restoration

Another example which illustrates the usefulness of diffusion-based filtering is image restoration. Because edges are enhanced and interrupted lines are closed with anisotropic diffusion, old paintings or corrupted images can be easily reconstructed.

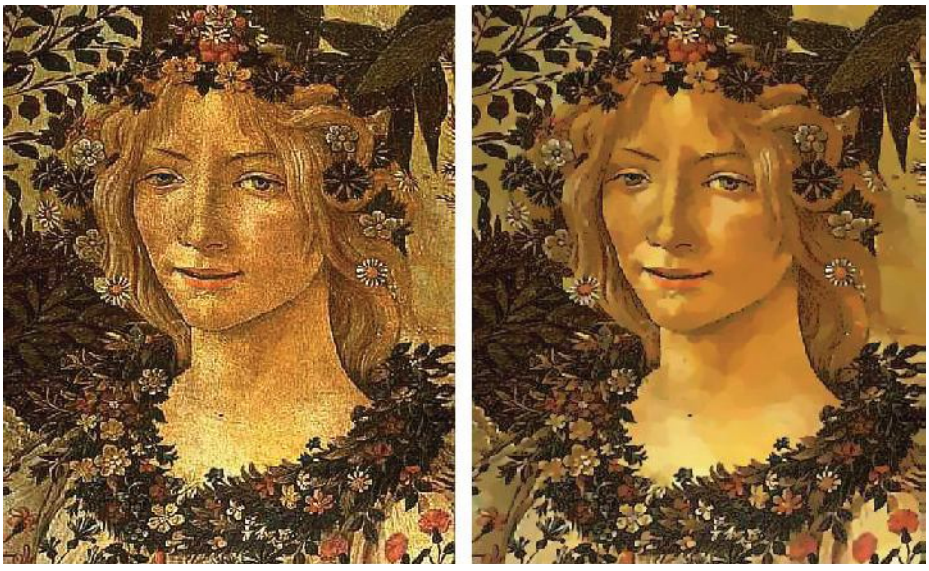


Figure 2.4 Reconstruction of Sandro Botticelli's painting Primavera (Uffizi Gallery, Florence). (a)LEFT: Original image. (b) RIGHT: Restored image using 'slowed anisotropic diffusion' [8]

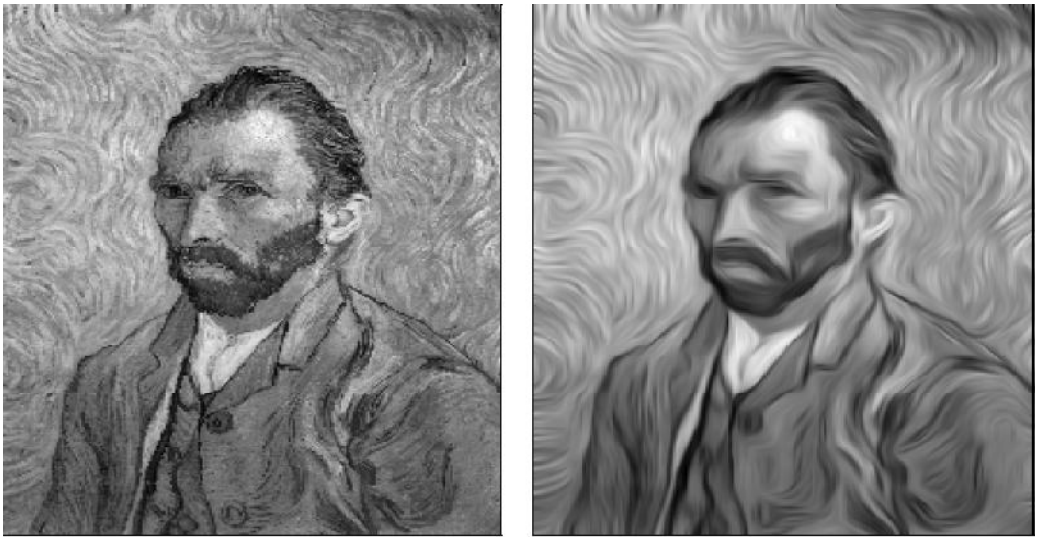


Figure 2.5 Image restoration using coherence-enhancing anisotropic diffusion. (a) Left: "Selfportrait" by van Gogh (Saint-Rémy, 1889;Paris, Musée d'Orsay). (b) Right: Filtered [4]

2.4.4 Applications in medical image analysis

Figure 2.5 gives an example for possible medical applications of nonlinear diffusion filtering as a preprocessing tool for segmentation. It depicts an MRI slice of the human head. For detecting Alzheimer's disease one is interested in determining the ratio between the ventricle areas, which are given by the two white longitudinal objects in the centre, and the entire head area. In order to make the diagnosis more objective and reliable, it is intended to automize this feature extraction step by a segmentation algorithm [4]. Figure 2.5 shows that one gets a better segmentation when processing the original image slightly by means of nonlinear diffusion filtering prior to segmenting it.

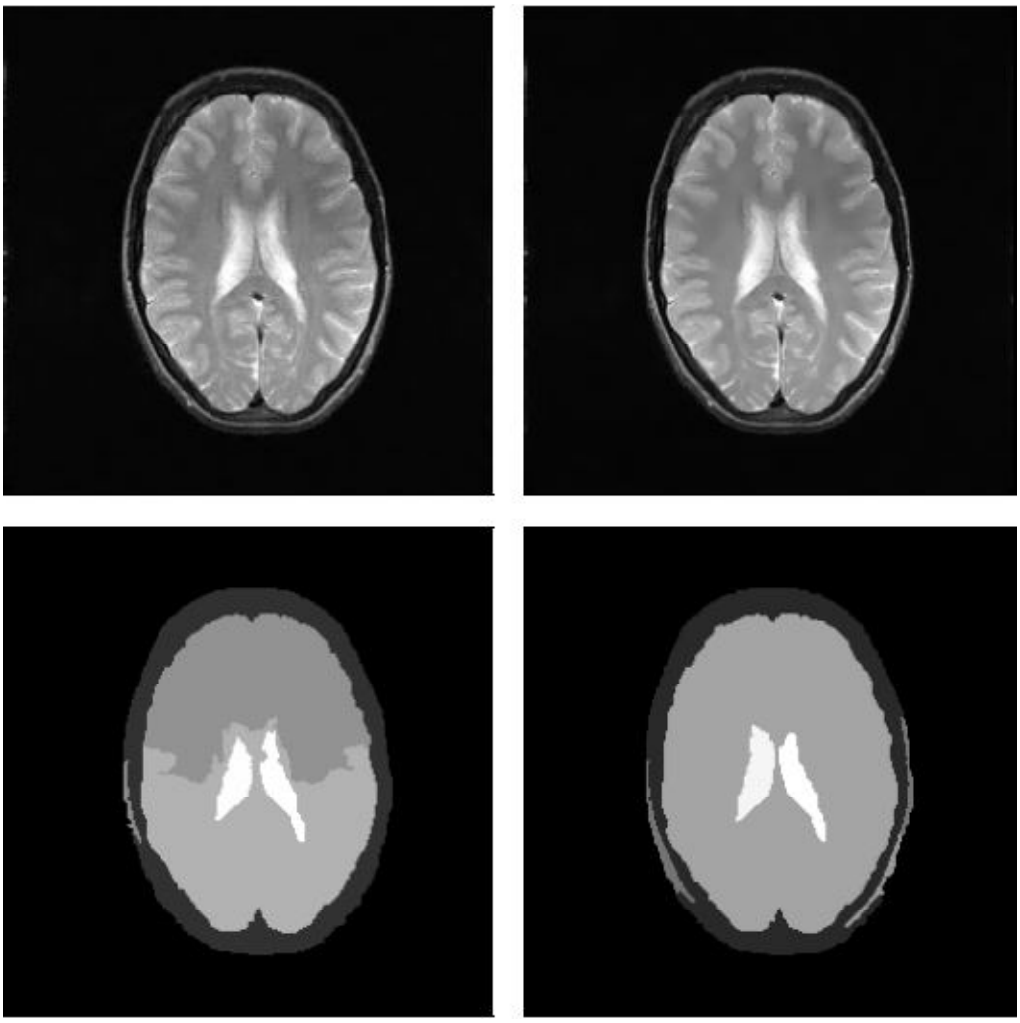


Figure 2.6 Preprocessing of an MRI slice. (a) Top Left: Head (b) Top Right: Diffusion-filtered, (c) Bottom Left: Segmented original image, (d) Bottom Right: Segmented filtered image. [4]

2.5 Summary

Diffusion algorithms remove noise from an image by modifying the image via a partial differential equation (PDE). Perona and Malik introduced the anisotropic diffusion equation making use of an edge-stopping function which stops the diffusion across the edges.

Anisotropic diffusion has many uses ranging from problems in computer aided quality control to applications in medical image analysis. In the following chapters we will be focusing on designing and implementing anisotropic diffusion for medical 3D images. This preprocessing filtering method has better segmentation results compared to the image smoothed by Gaussian filter or the original image without smoothing by any linear low-pass filter.

Results of anisotropic diffusion filtering depend on a few parameters which can seriously affect the diffusion process: the nature of the edge-stopping function, the value of the image gradient, the diffusion parameter K, the contrast parameter λ as well as the artificial time parameter (number of iterations). In the next chapter, a choice of these parameters is presented within the filter design.

3. Filter design

In the preceding sections anisotropic diffusion filtering has been introduced from a general point of view and some applications for 2D images have been presented. In this chapter the filter design and the diffusion parameters are examined in detail. The first section describes the extension to 3D of the classical anisotropic diffusion equation developed by Perona and Malik. The second section shows how robust statistics can be related to the diffusion process in order to estimate the diffusion parameter, while the third section deals with finding a simple stopping condition for the diffusion process.

3.1 Non-linear 3D diffusion filtering

In Perona and Malik's paper [5] filtering is formulated as a diffusive process where smoothing is performed at intra regions and suppressed at region boundaries. This section describes the extension to 3D of the anisotropic diffusion algorithm described in the previous chapter. The smoothing strategy described in Eq. (5) can be translated into an iterative discrete formulation for 3D data as follows:

$$I_{x,y,z}^{t+1} = I_{x,y,z}^{t} + \lambda \sum_{R=1}^{6} [g(\nabla_R I)\nabla_R I]^t \qquad (9)$$

where λ is a contrast parameter that takes a value in the range $0 < \lambda < 0.16$ and $g(x)$ represents the edge-stopping function [2]. ∇ is the gradient operator than can be defined according to Eq.(8) in a six voxel connected neighborhood as suggested in [2] as follows:

$$\nabla_1 I_{x,y,z} = I_{x-1,y,z} - I_{x,y,z}, \nabla_2 I_{x,y,z} = I_{x+1,y,z} - I_{x,y,z}$$

$$\nabla_3 I_{x,y,z} = I_{x,y-1,z} - I_{x,y,z}, \nabla_4 I_{x,y,z} = I_{x,y+1,z} - I_{x,y,z},$$

$$\nabla_5 I_{x,y,z} = I_{x,y,z-1} - I_{x,y,z}, \nabla_6 I_{x,y,z} = I_{x,y,z+1} - I_{x,y,z} \qquad (10)$$

3.2 Diffusion parameter

The diffusion function *g(x)* should be bounded in the interval (0->1) and should have the highest value when the *input x* has the value zero. These requirements translate to minimal smoothing around boundaries where the gradient has high values [2]. In practice, a large number of functions can be chosen to satisfy these requirements, the most popular having been presented in the previous chapter Eq. (5) and (6). The choice of *g(.)* can affect the extent to which discontinuities are preserved. Black et al. [1] state that diffusing with the Tukey norm Eq. (6) produces sharper boundaries than diffusing with the Lorentzian norm Eq. (5). Also, the choice of the function affects the "stopping" behavior of the diffusion; given a piecewise constant image where all discontinuities are above a threshold, the Tukey function will leave the image unchanged whereas the standard functions will not [1]. However, because of computational reasons, the results presented in this paper were obtained using the simplest of the 3 functions presented in chapter 2 – Eq.(5). Also, within the application, the user can chose the edge-stopping function he wishes to use.

More important than the choice of the edge-stopping function, is the choice of the diffusion parameter *K*. The parameter K controls the smoothing level which becomes more pronounced for high values of K. Black et al. [1] appealed to robust statistics in order to find an appropriate diffusion parameter. The assumption made is that the image has been corrupted by zero-mean Gaussian noise with small variance. Consider the image intensity differences, $I_p - I_s$, between pixel *s* and its neighboring pixels *p*. Within one of the piecewise constant image regions, these neighbor differences will be small, zero-mean, and normally distributed. On the other hand, the neighbor differences will not be normally distributed for an image region that includes a boundary (intensity discontinuity). Therefore, with respect to the assumption of Gaussian noise within each constant region, the neighbor difference $I_p - I_s$ can be viewed as an *outlier* because it does not conform to the statistical assumptions [1].

Before considering it to be an outlier, it has to be determined how large the image gradient can be. This is done using tools from robust statistics in order to estimate automatically the "robust scale", σ_e , of the image and the diffusion parameter as:

$$\boldsymbol{K = \sigma_e = 1.4826\, MAD\,(\nabla I) = \; 1.4826\, median_I(\|\nabla I - median_I(\|\nabla I\|)\|)} \quad (11)$$

where "MAD" denotes the median absolute deviation and the constant is derived from the fact that the MAD of a zero-mean normal distribution with unit variance is 0.6745 = 1/1.4826 [1]. The assumption made here is that the input image is a piecewise constant function that has been corrupted by zero-mean Gaussian noise with small variance. As this is not the case with real images, in order to obtain meaningful results, the user has to specify a small area of the image which verifies the assumption from above.

3.3 Artificial time parameter

In the previous section, we have seen that the choice of the diffusion function affects the "stopping" behavior of the diffusion when the discretized diffusion is iterated until convergence (t -> ∞). However this is computationally expensive. In addition, diffusion is a slow process, so finding an optimal stopping condition could lower the computational time improving thus the performance of the diffusion algorithm.

As mentioned before, diffusion-based filtering has been developed in order to implement an optimal, feature preserving smoothing strategy. However, an aggressive filtering (for high values of the diffusion parameter K) combined with a high enough number of iterations could lead to attenuating small details in the image. With medical images it is vital to preserve even the smallest features in order to be able to give the best diagnostic. In chest CT data, nodules can be very small (See Fig. 3.1) and running the diffusion algorithm for a large number of iterations could lead to altering the shape and the size of the nodules. So, finding an optimal number of iterations does not only lower the computational time but also can help preserving features of interest from the dataset.

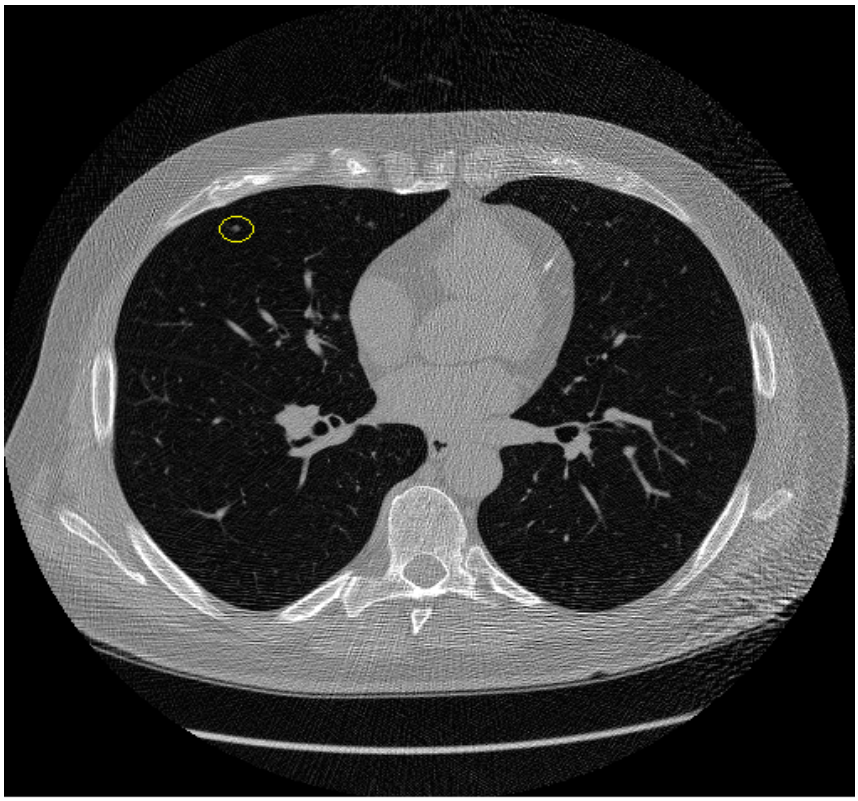


Figure 3.1 Nodule from a chest CT slice. The yellow circle indicates the area containing the small nodule

The idea behind finding the optimal stopping condition is very simple. Unfortunately, same as finding the diffusion parameter, it is not a fully automated solution. The user has to specify a feature that he wants preserved (a nodule for example). This requires some prior knowledge of the datasets. After specifying the feature to be preserved, its area is calculated at each iteration and then compared to the initial area. The diffusion stops when the area of the feature becomes smaller than the initial one. Choosing a small enough feature assures that the smoothing will not alter the shape and size of the features of interest.

Even if it is not very robust, this simple solution can give a good enough estimator for the number of iterations needed for the diffusion process for similar datasets. Results show that the optimal number of iterations is somewhere between 30 and 50 iterations for chest CT datasets.

3.4 Summary

The behavior of the anisotropic diffusion depends on two parameters: the artificial time parameter t and the gradient thresholding parameter K. The appropriate choice of these parameters is essential to obtain a conveniently filtered image. The diffusion parameter K is estimated using robust statistics tools in order to maximize the signal-to-noise ratio (SNR) in the image while the number of iterations is calculated so that the diffusion process stops before altering features of interest in the image which can be very small. The following chapter shows the implementation of the anisotropic diffusion filter with the specifications mentioned in this chapter.

4. Implementation

This chapter deals with the implementation of the diffusion filter for 3D datasets following the specification from the previous chapter. This is programmed in Java. The first section presents the graphical user interface which allows the user to visualize the datasets and to control the choice of the diffusion parameters. The second section shows how the actual diffusion process is implemented while the third section shows how this is done in an efficient way in order to save computational time.

4.1 Graphical User Interface

The graphical user interface is developed in Java and allows the user to visualize the chest CT datasets (slices) and to control the diffusion process. Most of it has been developed by Dr. Tarik Chowdhury, I have only modified it and adapted it to the anisotropic diffusion filtering.

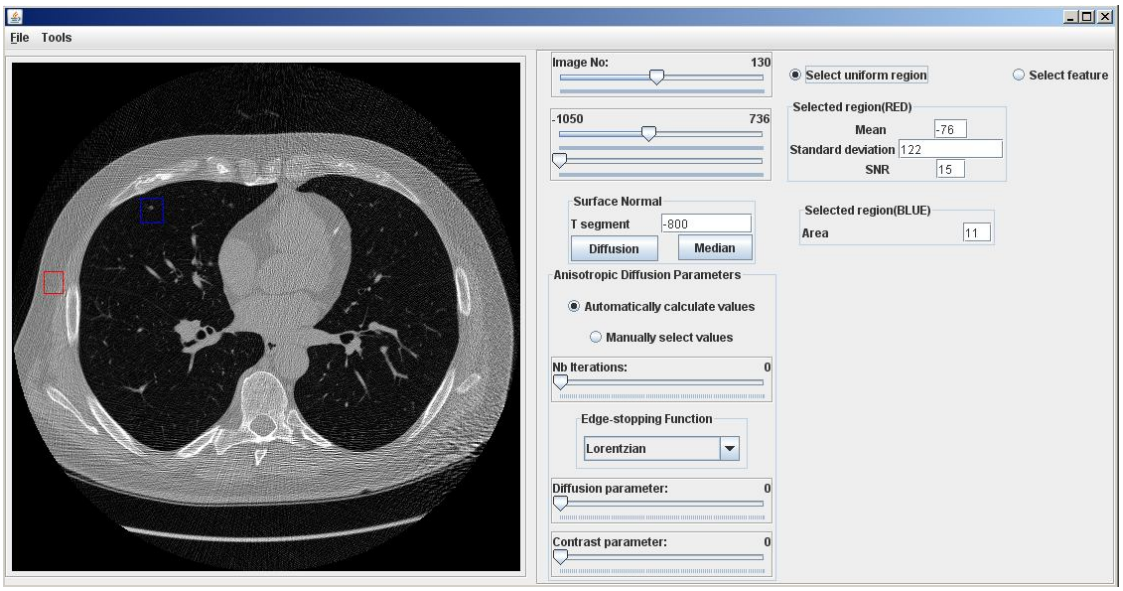


Figure 4.1 Graphical user interface

This interface is developed using Java Swing components within the *LungInterfaceP class*. This class contains also the main method. The slices are visualized using a label (on the left side). The first slider bar allows the user to choose the slice he wants to visualize from the dataset and the second one modifies the contrast of the images. The user can choose either to apply anisotropic diffusion filtering or median filtering to the dataset. This is done for comparison purposes.

The rest of the slider bars and radio buttons control the anisotropic diffusion parameters. The user can choose either to let the application automatically estimate the parameters as described in Chapter 3 or to manually select the parameters. If the "Manually select values" option is selected, the user has to indicate with the help of slider bars the number of iterations (artificial time parameter), the diffusion parameter K and the contrast parameter λ (Eq. (9)). The type of the edge-stopping function has to be specified either way.

If the first option is selected ("Automatically calculate values") the user first has to specify a piecewise constant area from one of the slices (soft tissue, for example) in order to estimate the diffusion parameter K as described in the previous chapter. This is done by selecting the radio button labeled "Select uniform region". After having selected this button, the user can select a region from the current slice by dragging the mouse on the image. A red rectangle indicates the selected region. In the same way, the user can indicate a region containing a feature to be preserved when selecting the radio button labeled "Select feature". This time, a blue rectangle is drawn on the screen. Figure 4.2 shows an example of using the two selectors. See Appendix 1 for the enlarged graphical user interface.

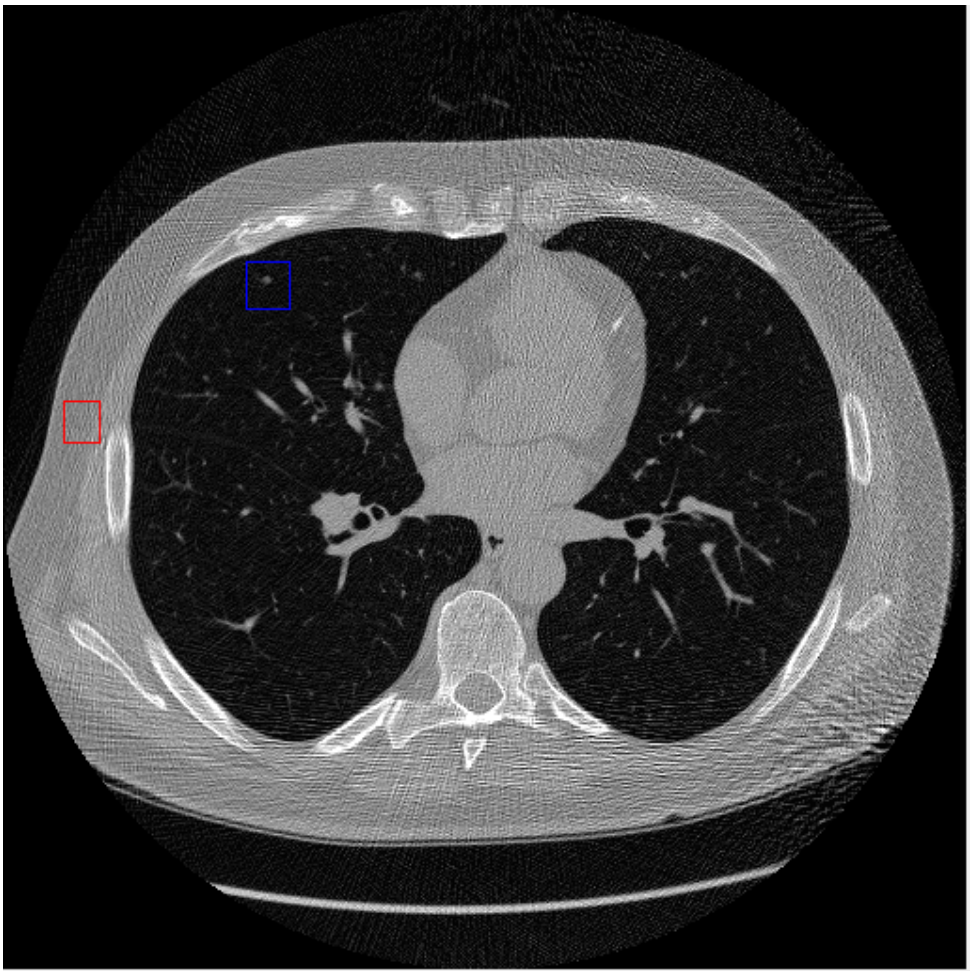


Figure 4.2 Selecting regions of interest in the image

The interface also displays details of the selected regions using text fields (See Appendix 1). For the constant piecewise region the signal-to-noise ratio is displayed whereas for the region containing the small feature, its area. This can be achieved by using a "selection

label" for visualizing the slices implemented in the *ImageSelectionPanel class* which extends the *jLabel class* (See Appendix 2 for the code). This class sends then the coordinates of the 2 rectangles to the *LungInterfaceP class.* At its turn, LungInterfaceP class sends the coordinates and the dimensions of the rectangles to the *TestLung class* which is the core of the application. It performs all the calculation necessary and the filtering. The TestLung class sends back to the LungInterfaceP class the results.

The estimation of the diffusion parameter K is implemented according to the specifications from chapter 3 within the TestLung class.

```
public double robustScaleFromROI(int X, int Y, int width, int height, int
slice){


      double[]ROIGradient = new double[width*height];
      int i = 0;
      for (int x=X;x<X+width;x++)
            for(int y=Y;y<Y+height;y++){
                  ROIGradient[i] = gradientMagnitude2D( x, y, slice);
                  i++;
            }

      Arrays.sort(ROIGradient);
      double median = ROIGradient[i/2];

      i=0;
      for (int x=X;x<X+width;x++)
            for(int y=Y;y<Y+height;y++){
                  ROIGradient[i] = Math.abs(gradientMagnitude2D( x, y,
slice) - median);
                  i++;
            }


      return 1.4826*ROIGradient[i/2];
    }
```

The image gradient magnitude is calculated according to [7] :

```
private double gradientMagnitude2D(int x, int y, int z){

      short dx = (short) (0.5*(getVoxelValueFast(x+1,y,z)-
getVoxelValueFast(x-1,y,z)));
      short dy = (short) (0.5*(getVoxelValueFast(x,y+1,z)-
getVoxelValueFast(x,y-1,z)));

      return Math.sqrt(dx*dx+dy*dy);
    }
```

The voxel values from one dataset are all stacked in a one-dimensional array. The getVoxelValueFast() function extracts the value of the voxel at the corresponding coordinates from that array.

In order to estimate the total number of iterations necessary, the user has to select a feature in the image to be preserved (a small nodule for example). Its area is calculated at each iteration and compared to the initial value. The selected region is at first transformed into a binary region by thresholding it and then the biggest blob is extracted from the region. This is done following a blob coloring algorithm [9]. Given a binary image **binary** = [binary(i);0≤ i ≤ N-1] define an array **regions** = [regions[i]; 0≤ i ≤ N-1] where region[i] is the region number of the binary-valued pixel binary(i), according to the following code. In the code denote the set of 2-D pixel coordinates **previous** = [(i-1,j) ; (i,j-1)] to denote the set of coordinates that are immediately adjacent (not along the diagonals) to i [9].

```
private int[] getRegions(int X, int Y, int width, int height, int slice){

            int[] regions = new int[width*height];
            short[] binary = new short[width*height];
            short[] previous = new short[2];
            int k;

            for (int i=0;i<width*height;i++)
                  regions[i]=0;

            binary = thresholdRegion(X, Y, width, height, slice);

            k=1;

            for (int i=0;i<width*height;i++){

                  //Create the array containing the previous pixels (x-
1,y) and (x,y-1)
                  if((i-1)%(width-1) >0)previous[0] = binary[i-1]; else
previous[0]=0;
                  if(i-width>=0)previous[1] = binary[i-width]; else
previous[1]=0;

                  if (binary[i] == 1){

                        if (previous[0]==0){
                              if(previous[1]==0){
                                    regions[i]= k;
                                    k++;
                              }else regions[i] = regions[i-width];
                        }else regions[i] = regions[i-1];
```

```
                }
            }
```

At the end of this procedure, all white pixels in **binary** belonging to the same connected region are assigned to the same region number (color). The array **regions** contains the region number of every pixel in the selected area. Using this information, the total number of regions and the area of each region can easily be found [9] and thus the biggest blob can be extracted:

```
private int getBiggestBlob(int[] regions, int numberOfPixels){

            int[] area = new int[numberOfPixels];
            int maxArea;

            for(int i=0;i<numberOfPixels;i++)
                  area[i]=0;

            for(int i=0;i<numberOfPixels;i++)
                  if (regions[i]>0)
                        area[regions[i]]++;

            maxArea=0;
            for(int j=0;j<numberOfPixels;j++)
                  if(maxArea<area[j])
                        maxArea=area[j];

            return maxArea;
      }
```

4.2 Anisotropic diffusion filter implementation

Having calculated the diffusion parameter K and the area of the feature to be preserved, we can proceed to implementing the actual filtering. First, the image gradients from Eq. (10) are computed and stored in an array:

```
private short[] createGradientArray(int x, int y, int z){

      short[] gradient = new short[6];
      gradient[0] = (short) (getVoxelValueFast((x-1),y,z) -
getVoxelValueFast(x,y,z));
      gradient[1] = (short) (getVoxelValueFast((x+1),y,z) -
getVoxelValueFast(x,y,z));
      gradient[2] = (short) (getVoxelValueFast(x,(y-1),z) -
getVoxelValueFast(x,y,z));
```

```
      gradient[3] = (short) (getVoxelValueFast(x,(y+1),z) -
getVoxelValueFast(x,y,z));
      gradient[4] = (short) (getVoxelValueFast(x,y,(z-1)) -
getVoxelValueFast(x,y,z));
      gradient[5] = (short) (getVoxelValueFast(x,y,(z+1)) -
getVoxelValueFast(x,y,z));


      return gradient;

    }
```

A temporary one-dimensional array of the same length as the initial data array stores the diffused values of the voxels at each iteration:

```
short[] nData = new short[data.length];
```

The new values of the voxels are then calculated according to Eq. (9):

```
while (currArea==initArea){ // stopping condition
            ...
            for(int z=1; z<depth-1;z++){
                  for(int y=1;y<height-1;y++){
                        for(int x=1; x<width-1; x++){
                              //create the array;
                              ...
                              short[] gArray = createGradientArray(x,y,z);
                                    double div = 0;
                                    for(int i=0;i<6;i++)
                                          ...
                              div = div +
gArray[i]*diffusionFunction(gArray[i], K, functionType);
                                    nData[x+y*width+z*width*height] =
(short)(getVoxelValueFast(x,y,z)+ (div*Lambda));
                                    gArray = null;
                              }
                        }

            }
```

The new values of the voxels are then copied to the initial data array and the process is repeated until the stopping condition is reached.

```
for(int z=1; z<depth-1;z++){
                  for(int y=1;y<height-1;y++){
                        for(int x=1; x<width-1; x++){

      setVoxelValueFast(x,y,z,nData[x+y*width+z*width*height]);
                        }
                  }
            }
            nData=null;
```

Even if the diffusion process does not alter the feature chosen by the user, the process is ended when the number of iterations reaches 50.

4.3 Efficient implementation

Diffusion filtering is a very slow process compared to other filtering techniques and this can be a major drawback when dealing with medical datasets. This section deals with minimizing the computational time of the filter implementation described in the previous section.

In general CT chest datasets contain surrounding air voxels, body tissue/fat/blood vessels, air inside the lung and air inside small/large intestine. Surrounding air voxels number comprises 20-30% of the total voxels inside the CT chest dataset. Hence, to reduce the computational time, a surrounding air voxel removal method was applied initially and it was named as dataCrop(). This method marks the voxels representing air with a very high HU number, and thus they can be easily ignored by the diffusion algorithm.

Also, precious time can be saved by stopping the diffusion for smooth areas. If a region is smooth enough, the diffusion algorithm will not change the value of the voxels belonging to that area because the gradient in those areas will be close 0. So, another solution is to test if the gradient (calculated from Eq. (10)) for a voxel is not close to 0 and only then apply the diffusion algorithm. This means that, even if the algorithm runs for 50 iterations for example, not all the regions in the image undergo the diffusion process 50 times.

4.4 Summary

Anisotropic diffusion is implemented in Java. The graphical user interface allows the user to visualize the slices from the chest CT datasets and to calculate or to choose the parameters controlling the diffusion process. Computational time is reduced by ignoring the surrounding air voxels and by stopping the diffusion for smooth regions within the volume. The following chapter, in which experimental results are presented, also gives an estimate of the time saved with the efficient implementation discussed here.

5. Results and discussion

In Chapter two we have seen that anisotropic diffusion filtering has better segmentation results compared to images smoothed by Gaussian filter or original images without smoothing by any linear low-pass filter. Among non-linear techniques that have been developed in order to achieve better feature preservation, median filter is the simplest operator to remove impulse-like noise [2]. In this chapter, anisotropic diffusion filtering results for chest CT datasets are presented in comparison to median filtering. The assumption made is that the noise is additive with a normal distribution. Any other type of noise which do not follow this assumption, like motion artifacts for example, cannot be removed using diffusion filtering.

This chapter is organized as follows: section one shows results from the point of view of level of noise reduced by the filters. Section two presents segmentation results while section three gives an estimate of time saved by the implementation described in the previous chapter.

5.1 Level of noise

The most appropriate tool for estimating the level of noise reduced by the filter is signal-to-noise ratio (SNR). The *signal-to-noise ratio*, *SNR*, can have several definitions. The noise is characterized by its standard deviation, s_n. The characterization of the signal can differ. If the signal is known to lie between two boundaries, $a_{min} <= a <= a_{max}$, then the *SNR* is defined as [10]:

(12)

where s_n is the *sample standard deviation* representing the unbiased estimate of the standard deviation of the brightness within a region (R) with n pixels.

(13)

The average brightness of a region is defined as the *sample mean* of the pixel brightnesses within that region. The average, m_a, of the brightness over the A pixels within a region (R) is given by [10]:

(14)

Results show a higher signal-to-noise ratio of a region containing soft tissue (Figure 5.1) for anisotropic diffusion filtering than for median filtering.

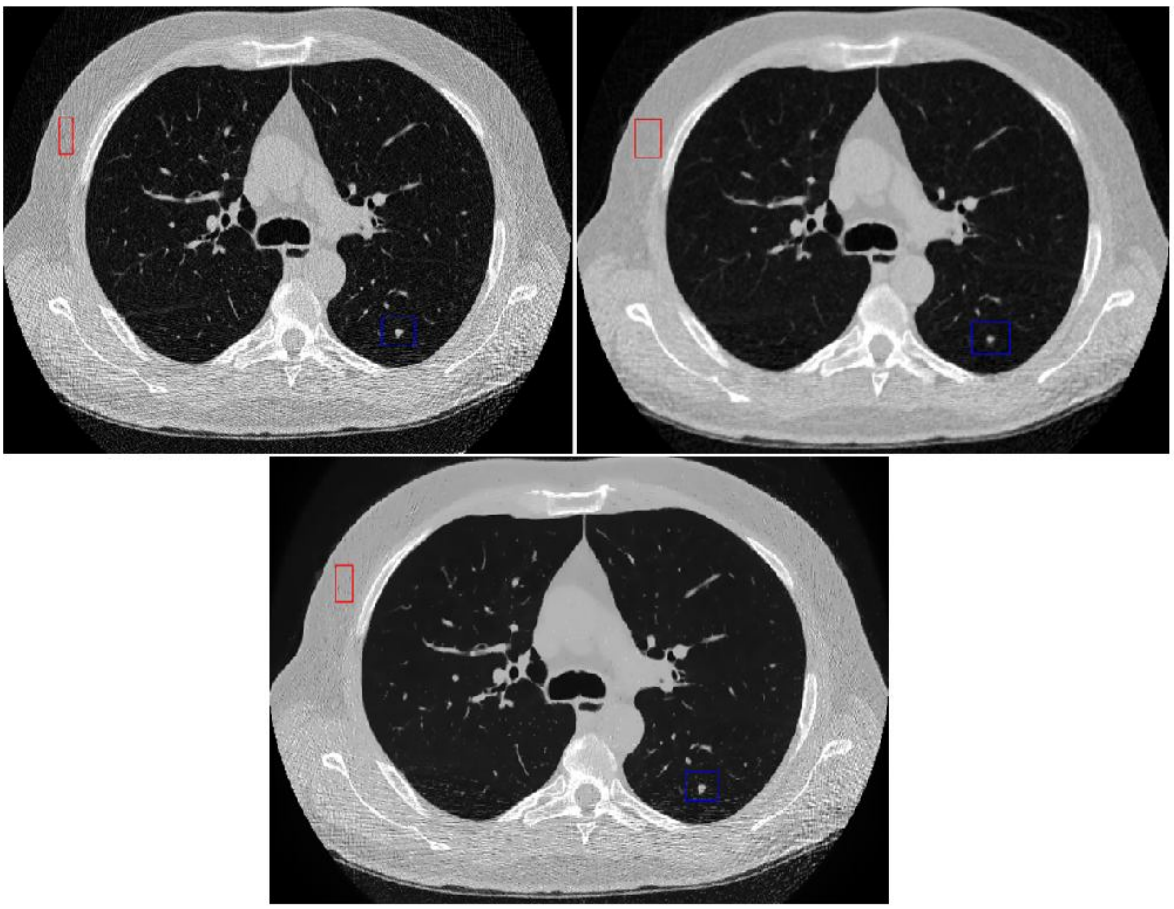


Figure 5.1 SNR of soft tissue. Red rectangle indicates the selected region. (a) LEFT: original image - 16.34 dB. (b)RIGHT: median filter - 16,83 dB. (c)DOWN: anisotropic diffusion, 50 iterations - 21.78 dB

The signal-to-noise ratios calculation confirms what the naked eye can easily observe: that the anisotropic diffusion filter produces a better image than the median filter. Also, looking at the small nodule (blue rectangle), we can see that the median filter has altered its shape, while anisotropic diffusion has kept it unchanged compared to the original image.

5.2 Segmentation results

As mentioned in the previous chapters, anisotropic diffusion filtering is a very useful preprocessing technique meant to improve feature extraction and segmentation results within medical images. Chowdhury et al. [11] have developed a fully automatic CAD-CTC system for detecting polyps in the colon which can also be adapted to chest CT data. The main system components of their implementation are: 1) automatic colon segmentation; 2) candidate surface extraction; 3) feature extraction; and 4) classification. Their objective is to extract only the features from the candidate surfaces that offer the best discrimination between polyps and folds. Knowing that polyps and folds can be in some situations almost indistinguishable, the discriminative features employed to perform polyp detection have to optimally exploit the geometrical difference between polyps and folds in order to achieve robust polyp detection at a low level of false positives. Knowing that the nominal model for polyps is spherical and the nominal model for folds is cylindrical, a number of key features are extracted, namely: the standard deviation (SD) of the surface variation, the SD of the three axes of the fitted ellipsoid, SD of the ellipsoid fit error, and SD of the sphere fit error and the value of the Gaussian distribution. These features generate a low-dimensional model that describes the candidate surface that is used to classify the candidate surfaces into polyps and folds [11].

When applied to chest CT data, results show a significant improvement for anisotropic diffused images compared to median filtered images:

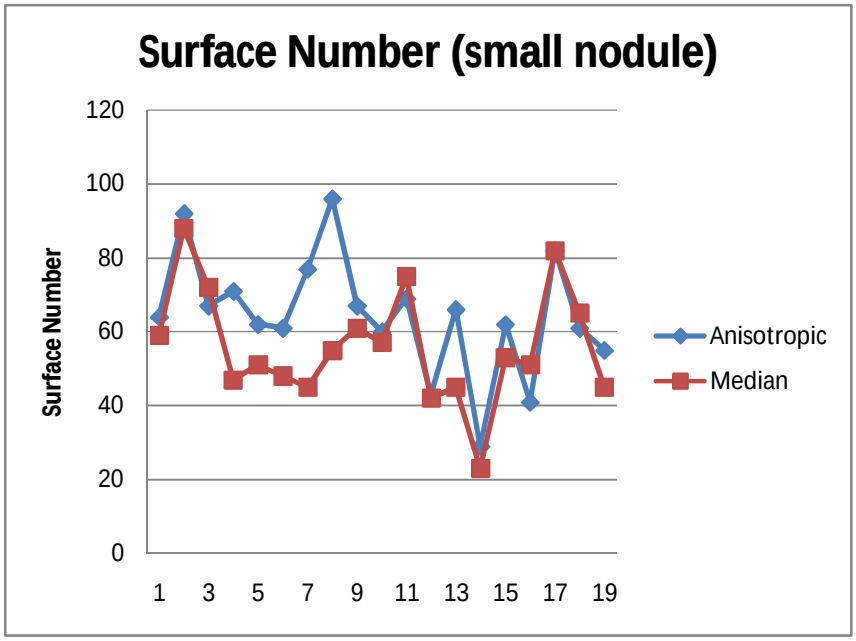


Figure 5.2 Surface Number for a small nodule

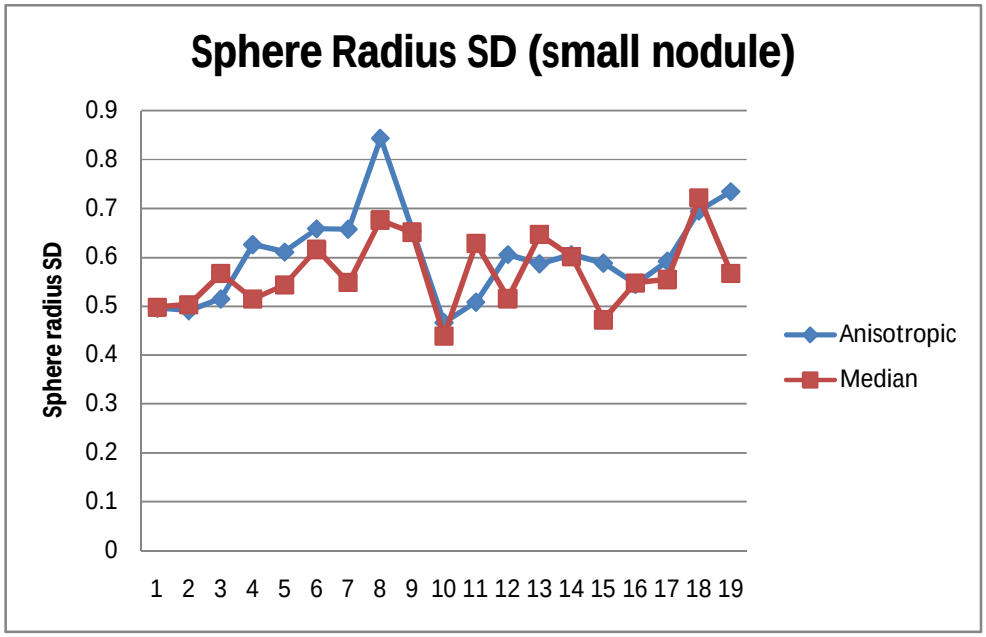


Figure 5.3 Sphere Radius standard deviation

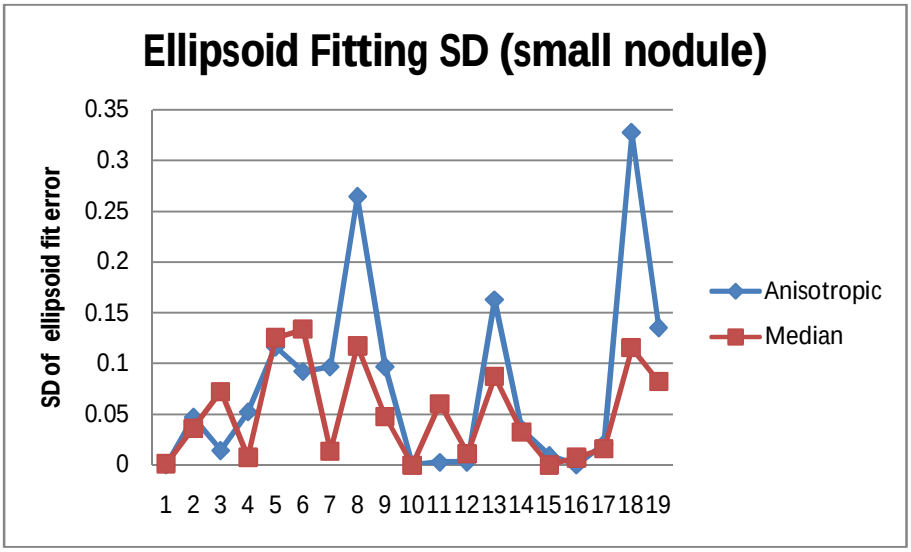


Figure 5.4 SD of the ellipsoid fit error

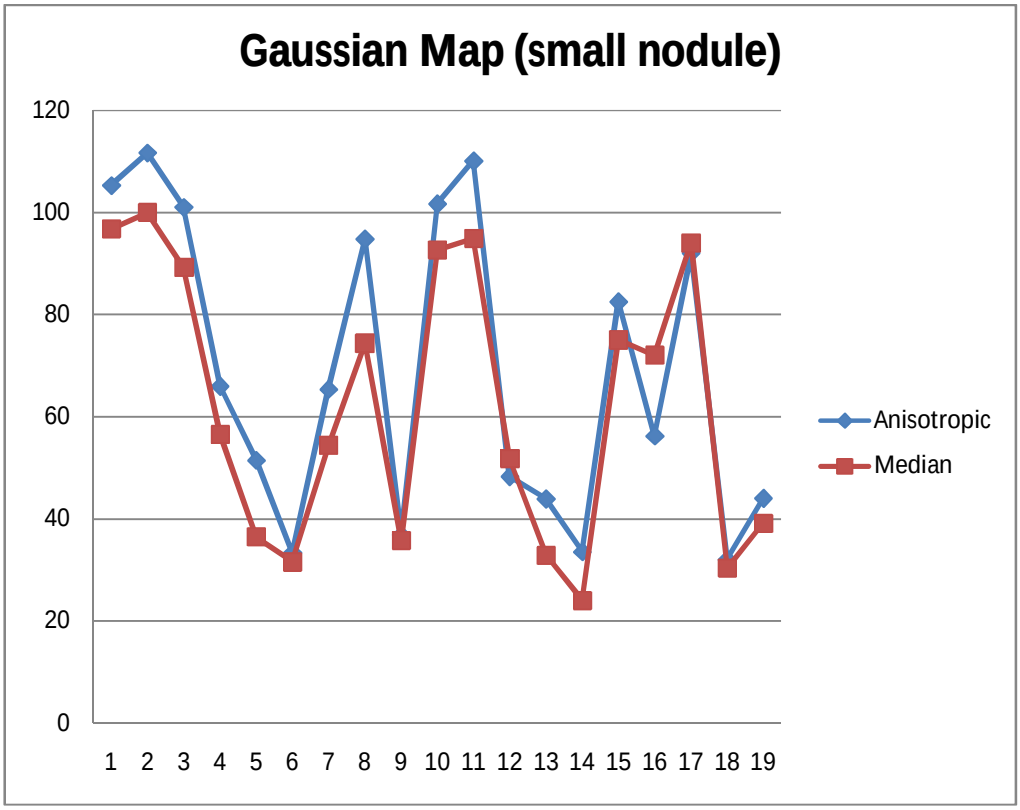


Figure 5.5 Gaussian distribution

5.3 Computational time

Diffusion based filtering is a very slow process and even if it produces better results than median filtering, it is not very attractive because of the big computational time needed. Section 4.3 shows how anisotropic diffusion can be implemented minimizing the computational costs. For 50 iterations 19 minutes and 12 seconds are needed for filtering a dataset containing 276 slices, each having a resolution of 512 x 512, thus all in all 512 x 512 x 276 voxel values to be processed. The station on which the tests have been performed has an Intel Core 2 processor with 1.8 GHz and 2.0 Gb of RAM memory. Ignoring the surrounding air voxels reduces the computational time with approximately 18% while stopping the diffusion process when edges are close to zero reduces the time needed with approximately 5%. All in all, with the implementation described in section 4.3, the computational time is reduced to 15 minutes and 9 seconds, or by approximately 23%.

5.4 Summary

Anisotropic diffusion filtering reduces significantly the level of noise in the images without altering the small features from the datasets. Used as a preprocessing technique, it produces better segmentation results than median filtering with the disadvantage that it is much slower. However, the performances of the filter can be further improved and the computational time reduced. The next chapter presents the final conclusions and several directions for further research.

6. Conclusions and further research

In this paper the implementation of anisotropic diffusion-based smoothing scheme and its application to medical 3D data has been described. The interest has focused on chest CT data which are characterized by a low signal-to-noise ratio (SNR) and it has been shown that this kind of filtering can reduce the level of noise from images without deteriorating features of interest unlike linear low-pass filters or median filter. It has also been shown that, used as preprocessing technique, anisotropic diffusion filtering produces better segmentation results than median filtering.

The behavior of the anisotropic diffusion depends mainly on two parameters: the artificial time parameter *t* and the gradient thresholding parameter *K*. The appropriate choice of these parameters is essential to obtain a conveniently filtered image. The latter is calculated in order to maximize the signal-to-noise ratio from the image using robust statistics tools while the number of iterations is chosen so that the diffusion process stops when the small features of interest become altered.

Unfortunately, the major drawback of diffusion-based filtering is that it requires a large computational time. Techniques of reducing this computational time have been described in this paper.

A number of related issues remain to be investigated:

1. Developing a fully automated solution; so far, the solution is not automated and the user has to indicate certain regions of the image, in order to calculate the diffusion parameters. This implies that the user has a prior knowledge of the datasets, which is not the case in most of the times.
2. Extending the rationale for applying robust statistics in order to estimate the K parameter from the case of piecewise-constant images to more general ones.
3. Investigating other choices of the gradient thresholding parameter K (see [3]).

4. Developing a more robust stopping condition; so far, the stopping condition proposed in this paper does not always stop the diffusion process.
5. Investigating other gradient calculation methods (26 voxel connected neighborhood, diagonal derivates, etc.).
6. Reducing the computational time; so far, the techniques described in this paper reduce the computational time with 20%-25% but even so, diffusion filtering remains a very slow process in comparison to classical filtering techniques.

References

[1] Black M.J., Sapiro G. , Marimont D.H and Hegger D., "Robust Anisotropic Diffusion," *IEEE Trans. Image Processing*, vol. 7, no. 3, pp. 421–432, Mar. 1998.

[2] Ghita O., Robinson K., Lynch M., Whelan P., "MRI diffusion-based filtering: a note on performance characterisation",*Computerized Medical Imaging and Graphics*, pp 267-227, 2005.

[3] Kim H. Y., "An anisotropic diffusion with meaningful scale parameter", *Universidade de Sao Paulo, Escola Politecnica, Dept. Eng. Sistemas Eletronicos*

[4] Weickert J. , "Anisotropic diffusion in image processing" . Stuttgart: Teubner Verlag; 1998.

[5] Perona P, Malik J. "Scale-space and edge detection using anisotropic diffusion". *IEEE Trans. Pattern Anal. Machine Intell.* 1990;12(7): 629–39.

[6] Weickert J. , "Application of non-linear diffusion in image processing and computer vision". *Acta Math. Univ. Comenianae Vol. LXX, 1(2001), pp. 33–50 Proceedings of Algoritmy 2000.*

[7] Jacobs D., "Image gradients", *Class Notes for CMSC 426*, Fall 2005

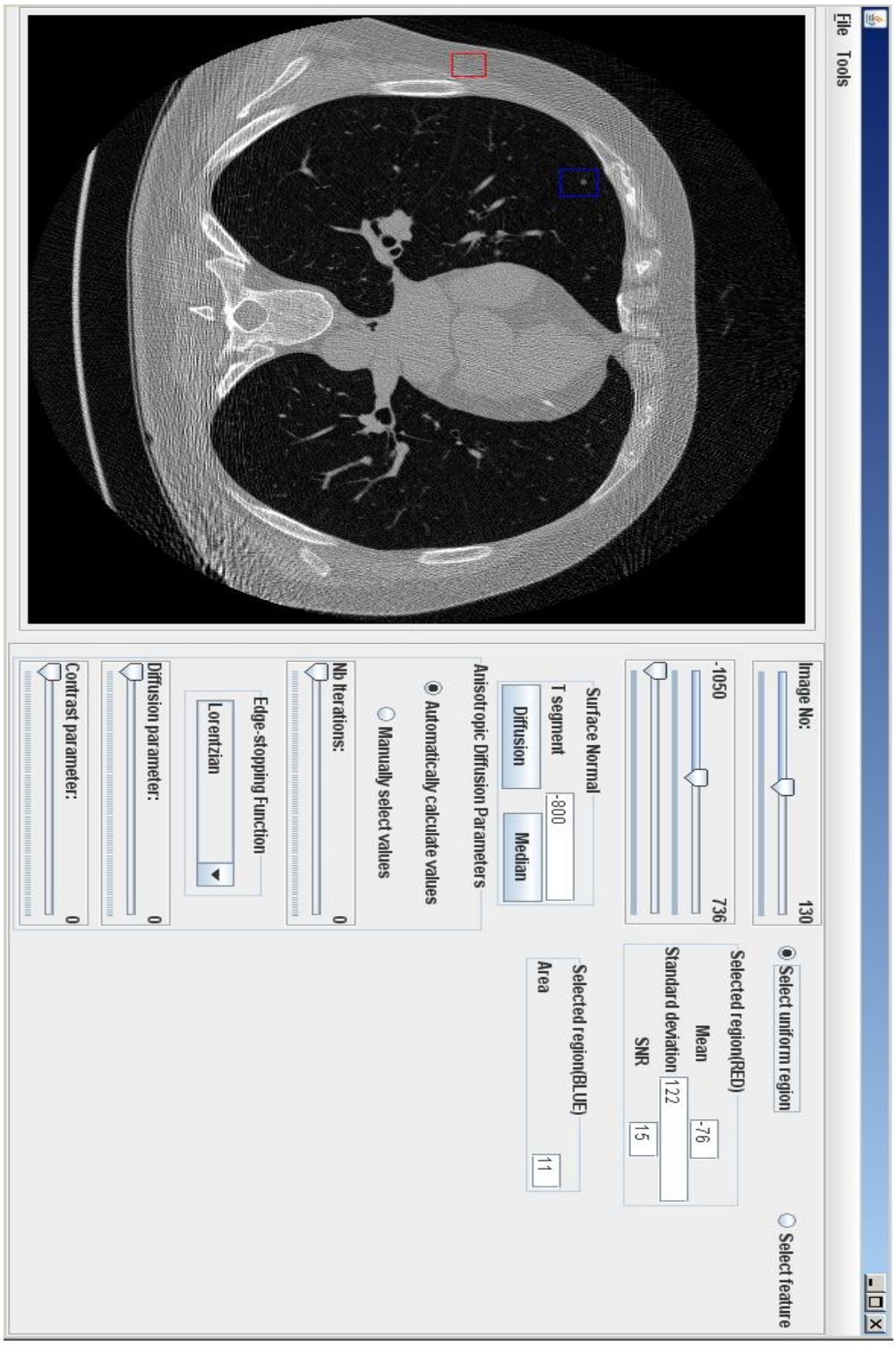
[8] Mikula K., "Slow and Fast Diffusion Effects in Image Processing", http://www.math.sk/mikula/sfd.html#top

[9] Häder D.P., "Image analysis: methods and applications", *CRC Press, 2000*

[10] Young I.T., Gerbrands J.J., van Vilet L.J., "Fundamentals of Image Processing", http://www.ph.tn.tudelft.nl/Courses/FIP/noframes/fip-Statisti.html

[11] Chowdhury T.A., Whelan P.F., Ghita O., "A Fully Automatic CAD-CTC System Based on Curvature Analysis for Standard and Low-Dose CT Data", *IEEE TRANSACTIONS ON BIOMEDICAL ENGINEERING*, VOL. 55, NO. 3, MARCH 2008

Appendix 1 – Graphical user interface



Appendix 2 – Rectangle selection java code

ImageSelectionPanel class

```
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import java.awt.image.BufferedImage;
import java.io.*;
import java.net.*;
import javax.swing.*;
import javax.swing.event.*;


public class ImageSelectionPanel extends JLabel
{
    BufferedImage image;
    Rectangle clipROI;
    Rectangle clipFeature;
    JFrame interfaceP;
    int clipType;

    public ImageSelectionPanel(JFrame InterfaceP)
    {
      this.interfaceP = InterfaceP;
        clipROI = new Rectangle();
        clipFeature = new Rectangle();
        setBackground(Color.black);
        Selector selector = new Selector(this);
        addMouseListener(selector);
        addMouseMotionListener(selector);
    }

    protected void paintComponent(Graphics g)
    {
      super.paintComponent(g);
        Graphics2D g2 = (Graphics2D)g;
        g2.setPaint(Color.red);
        g2.draw(clipROI);
        g2.setPaint(Color.blue);
        g2.draw(clipFeature);
    }

    public void setClipFrameROI(Point start, Point end)
    {

        repaint();
        LungInterfaceP lp = (LungInterfaceP) interfaceP;
        if(lp.getSelectType()=="ROI"){
            clipROI.setFrameFromDiagonal(start, end);
            clipType=0;
        }else{
            clipFeature.setFrameFromDiagonal(start, end);
            clipType=1;
```

```
        }
    }


    public Rectangle getClipFrameROI(){

      return clipROI;
    }
    public Rectangle getClipFrameFeature(){

      return clipFeature;
    }

    public JFrame getInterface(){

      return interfaceP;
    }

    public int getClipType(){
      return clipType;
    }
```

Selector class

```
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.net.*;
import javax.swing.*;
import javax.swing.event.*;


public class Selector extends MouseInputAdapter
{
    ImageSelectionPanel selectionPanel;
    Point start;
    boolean dragging, isClipSet;

    public Selector(ImageSelectionPanel isp)
    {
        selectionPanel = isp;
        dragging = false;
        isClipSet = false;
    }

    public void mousePressed(MouseEvent e)
    {
        if(isClipSet)             // clear existing clip
        {
            selectionPanel.setClipFrameROI(start, start);
            isClipSet = false;
        }
        else                      // or start new clip
        {
```

```
            start = e.getPoint();
            dragging = true;
            isClipSet = true;
        }
    }

    public void mouseReleased(MouseEvent e)
    {
       dragging = false;
       if(selectionPanel.getClipType()==0){
         LungInterfaceP lp = (LungInterfaceP)
selectionPanel.getInterface();
         lp.displayROI();
       }else{
         LungInterfaceP lp = (LungInterfaceP)
selectionPanel.getInterface();
         lp.displayArea();
       }
    }

    public void mouseDragged(MouseEvent e)
    {
        if(dragging)
            selectionPanel.setClipFrameROI(start, e.getPoint());
    }
}
```