KTH Royal Institute of Technology - Universitá degli Studi di Padova

School of Technology and Health - Department of Information Engineering
Master of Science Thesis in Bioengineering

Processing of CW Doppler images to extract velocity profile

Candidate:
Clara Dainese

Supervisor at KTH:
Prof. Fredrik Bergholm

Home Supervisor:
Prof. Enrico Grisan

Academic Year 2011-2012

Abstract

The present work aims to find a good way to automatically trace the velocity profile in vessels shown in continuous wave Doppler spectrograms, replacing the more traditional manual tracing, easy to perform but rich in drawbacks. Different methods of pre-processing this kind of images in order to prepare the edge detection step are presented. Various techniques, taken from the literature or newly created, are tested on a set of 18 CW Doppler spectrograms. The main purpose is to understand which is the best strategy to put into practice, remembering that the goal is to get the maximum velocity envelope, or the mean velocity in the vessel. Two main strategies are tested: a mild pre-processing before edge detection, followed by an edge linking step to fill the gaps in the extracted contour, or a stronger pre-processing that guarantees a continuous contour without further operations. A comparison is made on the obtained results and it is shown that the two approaches are somehow complementary in their pros and cons. In further work, the strengths of the two strategies should be combined in a hybrid method that would guarantee a good compromise between continuity of edges and mathematical-based detection of boundaries.

Contents

List of Notation

f	Grey-scale image
G	Gaussian distribution
g_x	Horizontal component of the gradient
g_y	Vertical component of the gradient
K	Generic kernel
M	Gradient magnitude
P	Pixel value
t	Template
T_{high}	High threshold for hysteresis thresholding
T_{low}	Low threshold for hysteresis thresholding

Introduction

Doppler spectrograms as source of information on blood flow

Doppler spectrograms, both continuous wave and pulsed wave, are very useful in giving information about blood velocities in vessels. Mean flow velocity and maximum flow velocity can be extracted from this kind of images.

The main problem with Doppler spectrograms is that until now the principal way of investigating them has been manual tracing of the edges seen in the images. This way of proceeding is clearly rich in drawbacks: although the technician is a well-trained person, the subjectivity of this method is great, and the results are not comparable with other ones traced by another technician. The velocity distribution needs to be quantified on the basis of recognized and replicable techniques.

The human eye is somehow deceived in detecting edges, because sometimes it sees something that does not really exists. For example a technician who tries to delineate outer contours in a Doppler sonogram would close the border even if it is not so clear where the real border is. This happens very easily above all with grey-scale images, and this is the case of study because Doppler sonograms are usually displayed as grey-scale images.

In this kind of images there are a lot of possibilities to detect different edges one next to the other and no one can say with absolute certainty which is the most significant one, or where it starts to be only noise instead of useful information. This is the reason why it is important to give more objective rules to trace the contour of the velocity profile.

Aim of the project

The purpose of this work is to overcome the limitations exposed above, and to improve the quantification step in the image analysis. We will develop a repeatable strategy for performing edge detection in Doppler sonograms, focusing on continuous wave Doppler ones.

Edge detection is however a challenging task and a lot of researchers are still working on it. The main problem is that the performance of the greatest part of edge detection algorithms is conditioned by the quality of the image and its typology.

However, of course edge detection is not the only possible approach to the problem. So, we will test also other techniques, trying to understand which is more reliable and best performing.

Structure of the work

In the first two chapters the background necessary for the comprehension of the work is presented. In chapter 1 principles of Doppler imaging technique are briefly introduced, with a particular attention to spectral continuous wave (CW) Doppler display. Chapter 2 presents some of the most common techniques for cleaning and enhancing medical images, focusing on those employed in the present work. In chapter 3 the implementative details of the different strategies adopted to extract the velocity profile from CW Doppler spectral images, and of further processing after it, such as extraction of the baseline and text and numbers recognition, are shown. Finally, a comparison between the different techniques seen in chapter 3 is provided by chapter 5. Matlab code described in chapter 3 is given as an appendix.

Chapter 1

Background. Doppler medical imaging

1.1 Doppler imaging techniques

Ultrasonic imaging is a very wide field in biomedical imaging. Besides the visualization of anatomical structures, which can be seen in A-mode, B-mode, C-mode or M-mode images, the Doppler mode exploits the Doppler effect in order to detect the motion of blood and tissue.

There are different Doppler modes and each one gives different information:

- Colour Doppler or Colour flow imaging, where velocity information is displayed as a colour code overlying on a B-mode echographic image.
- Continuous wave Doppler (CW), where the transducer emits and receives the ultrasound beam continuously. Because of the continuity in emission and reception of the signal, two separate piezoelectric elements are needed in the probe, one emitting and one receiving. Doppler information comes from a line through the body, and all the velocities found by the beam are shown at every time point.
- Pulsed wave Doppler (PW), where the velocity information comes only from a small sample volume at a known depth. The transducer emits ultrasound beams in pulses and the probe needs only one element to emit and receive.
- Duplex, where PW (or CW) Doppler is displayed together with 2D images (B-mode) or Colour Doppler images.

Both CW Doppler and PW Doppler techniques are called spectral Doppler because of the way of displaying the information: the spectrum of flow velocities on the y axis and time on the x axis, while the grey scale indicates the fraction of targets moving at that particular velocity.

In this work we will focus on Continuous wave Doppler images, mainly because there is not an upper limit to the Doppler shift that can be detected (like in PW Doppler systems). This is important when we are analysing diseased arteries: the lumen becomes narrower and the velocity of blood increases, thus increasing also the frequency shift.

1.2 The Doppler effect

The Doppler effect is the change of frequency of a received sound wave compared to the transmitted sound wave, and it occurs because of the relative motion between the observer and the source. If the source or the observer move one towards the other, the detected frequency will be higher than the emitted one, while it will be lower if they move away from each other. The resulting change observed in the frequency is proportional to the relative velocity between the source and the observer, and it is known as Doppler shift.

In biomedical devices, the transducer acts as a source and blood cells as the observers when the ultrasonic beam is sent towards the vessel, while on the way back blood cells become the source and the transducer (which acts also as a receiver) becomes the observer. So, the Doppler effect occurs twice from the transmission of the beam to its reception. In this case the transducer is stationary and the blood moves towards the transducer or away from the transducer.

The Doppler shift frequency f_d is the difference between the transmitted frequency (f_t) and the received frequency (f_r) and it is defined by the Doppler equation

$$f_d = \frac{2 f_t v \cos\theta}{c} \tag{1.1}$$

where v is the velocity of the blood, c is the speed of the ultrasound beam through the tissue (the average speed 1540 m/s is usually employed), and θ, also called angle of insonation, is the angle between the transducer and the vessel. This angle can change because of variations in the orientation of the vessels but it can also be altered manually by the operator, in order to obtain the highest Doppler frequency shift. The ideal situation is when the probe and the vessel are aligned, so that the cosine is 1.0, but often this is not achievable. Generally the goal is to keep the angle of insonation below 60°.

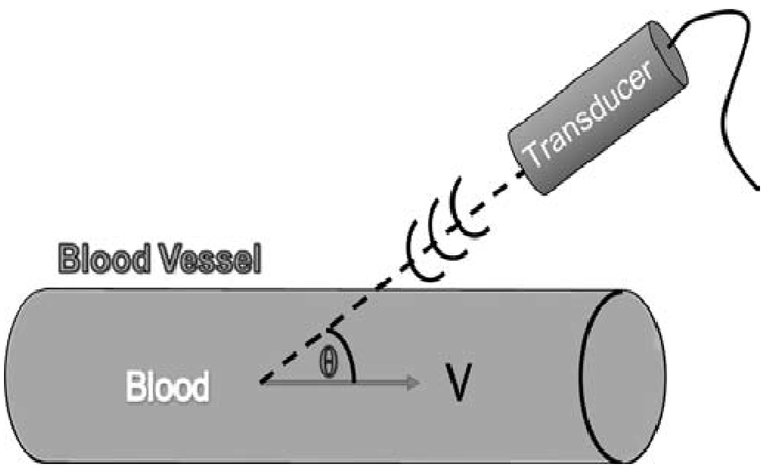


Figure 1.1: An ultrasonic transducer that transmits waves toward a blood vessel in which the blood flows with instantaneous speed V. There is an angle θ between the vessel's predominant dimension and the direction of the transducer.

Rearranging the Doppler equation (1.1), if we know θ (usually extracted by anatomical B-scan images), we can obtain an estimation of the blood velocity v. Actually, the Doppler shift will not contain a single frequency but a spectrum of frequencies, and so a spectrum of velocities will be displayed. This happens because all the targets in the path of the ultrasound beam are unlikely to move with the same velocity.

Looking at figure 1.1 we can see how the Doppler effect is exploited in medical ultrasound. Ultrasound waves produced by the transducer are scattered from the moving blood and this causes the change in frequency, that depends on the moving direction of the blood as already mentioned.

1.3 Spectral CW Doppler display

The spectrum is usually displayed as in figure 1.2. The vertical axis represents Doppler shift frequency or the velocity of blood, the horizontal axis indicates the time and the grey scale (or the brightness) is the amplitude of a certain frequency or velocity component at a certain time. In almost all the analysed images of this work the y-axis presents the velocity instead of the frequency.

The baseline in the centre of the spectrogram corresponds to zero velocity: the spectrum of positive velocities (flow towards the transducer) is displayed above the baseline and the spectrum of negative velocities (away from the transducer) is displayed below.

The range of blood velocities at each time is due both to the actual presence of different velocities in the sample volume and to the intrinsic spectral broadening. This phenomenon occurs because of the insonation of the blood at a range of different angles due to the shape of the beam, that is not planar and has a finite width. Thus it is a problem of the measurement system and it can not be completely avoided.

There are also other factors that affect somehow the shape of the velocity spectrum, making it not correspond exactly to the real shape of the velocity distribution (see [9]):

- Non-uniform insonation. The piezoelectric elements that transmit and receive the beams are not ideal, so some parts of the vessel could be preferentially sampled.
- Attenuation. Echoes from different parts of the vessel can experience different levels of attenuation.
- Filtering. Filtering must be performed on the received signal, that contains, besides the component of the blood flow, also information from other moving structures. After high pass filters, these components are rejected, perhaps also losing part of the low flow velocities in the vessel.
- Spectral analysis limitations. The analysis is performed in the frequency domain using fast Fourier transform algorithms, that lead sometimes to very noisy estimates.
- Multiple reflection. It happens in the presence of a strongly reflecting surface, giving the impression that a signal has been detected outside the vessel.

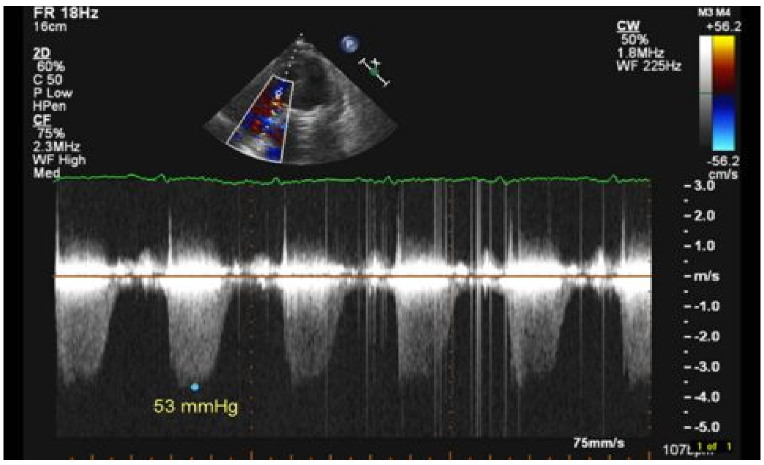


Figure 1.2: The typical format of a CW doppler sonogram, together with a colour flow image of the heart's section of interest.

1.4 Image resolution and quality

1.4.1 Resolution

Two kinds of resolutions must be considered and kept under control, axial and lateral. Resolution is the the smallest distance between two points that allows to see them as distinct. Axial resolution pertains to points that lie on the direction of propagation of ultrasound beams, while lateral resolution refers to points that lie at a right angle with respect to the direction of propagation. The first depends on the wavelength of the beam, and thus also on the frequency of the waveforms. This frequency is reciprocal to the ultrasound frequency, so the axial resolution improves with an increasing ultrasound frequency. On the other hand, lateral frequency depends on the width of the ultrasound wave, that has to be kept small.

However, an improved axial resolution is in conflict with the intensity of the received signal. In fact, the intensity of the signal depends on the attenuation experienced by the beam: the higher is the frequency, the higher is the attenuation. Therefore, there is a discrepancy between the resolution for images at smaller depths and images at higher depths. Usually the transmitted frequencies are in the range 1-10 MHz, but the frequency for a specific application must be chosen carefully considering what has been said.

1.4.2 Quality

One of the most significant disturbance factors in ultrasound images is speckle noise [22]. Noise degrades the quality of the image, thus affecting feature extraction and analysis.

Speckle noise is a pattern that arises because of the interference of many waves of the same frequency, but different phases and amplitudes, i.e. waves scattered from different sites. It implies an increase of the mean grey level of a local area.

Despeckling techniques, although very useful, must be carried out carefully to limit loss of information. Several filtering techniques have been developed in order to reduce this phenomenon, from linear methods to non-linear ones, and one should choose the most appropriate for the specific purpose.

Chapter 2

Background. Medical Image processing

The Doppler sonograms given by medical devices need to be refined in order to prepare the detection of the required features in an easier way. Doppler images are affected by noise, as already said in the previous section. First of all, we need to process the image somehow before performing the automatic extraction of the velocity profile. This is done in order to enhance the quality of the image, decreasing the contribution of noise without losing important details. As a matter of fact, we often do not know exactly what is noise and what is detail in the image. We have to choose the best compromise between "cleaning" the image and preserving useful information.

After this step, feature extraction can be performed in an easier way. Feature extraction includes different techniques, but we will concentrate here on edge detection (and techniques to improve it) and template matching, because we will use them in chapter 3.

2.1 Enhancement techniques

Image enhancement is an important step if we want to extract some features of an image, but it is not free from undesirable effects. While making useful information more apparent, we can lose something during the process, or even involuntarily increase the contribution of noise. There are several techniques of image enhancement, both in the spatial and in the frequency domain, and one should choose the best one, according to the original image and to the result he wants to get. We will explain only the methods used in this work.

2.1.1 Contrast stretching

Contrast stretching (also called normalization) is an operation that attempts to improve contrast by stretching the range of intensity values in an image. The aim is to bring the image into a range that is more familiar to the senses. It is a partly linear process.

First of all, we need to set an upper and lower pixel value limits (the new range of the image), for example 5% and 95% in the histogram. Each pixel with a value below the minimum will be set to 0, and each pixel above the maximum will be set to 255 (0 and 255 are often chosen values in 8-bit images). Looking at figure 2.1 as a reference, the slope of the straight line between k_0 and k_f is $K = \frac{255}{(k_f - k_0)}$, and the output value is $k_{out} = (k_{in} - k_0) \cdot K$. More generally, the transformation is:

$$P_{out} = \begin{cases} \theta'_{min} & \text{if} \quad P_{in} < \theta_{min} \\ (P_{in} - \theta_{min}) \left(\frac{\theta'_{max} - \theta' min}{\theta_{max} - \theta_{min}} \right) + \theta'_{min} & \text{if} \quad \theta_{min} \leq P_{in} \leq \theta_{max} \\ \theta'_{max} & \text{if} \quad P_{in} > \theta_{max} \end{cases} \tag{2.1}$$

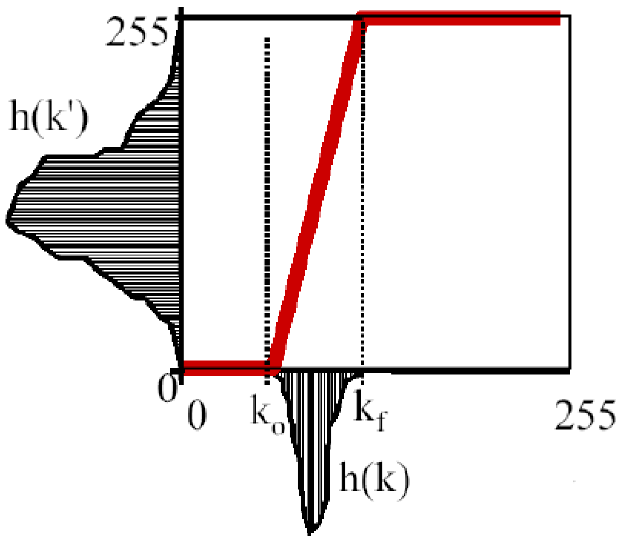


Figure 2.1: Linear contrast stretching in the dynamic range between the limits k_f and k_0. Every pixel below k_0 is mapped to 0, and every pixel above k_f is mapped to 255 (the latter operation is non-linear).

where P_{in} and P_{out} are the values of the pixels in input and output, while θ_{min}, θ_{max}, θ'_{min}, θ'_{max} are the bounds of the input linear region and of the output range, respectively.

2.1.2 Median filtering

Median filtering is a non-linear method often used to suppress some kind of noise (for example "speckle noise" or shot noise, in which some isolated pixels have extreme values), useful because it preserves edges.
This filter is a local operator. It needs a filter region (usually a $p \times p$ square), that must be centered on each pixel (i, j). The greater is p, the more the image will be filtered. The neighbouring pixels within the frame of the filter region of each pixel are sorted according to their brightness and the central pixel is replaced by the median value.

This kind of filtering is better than the smoothing filters, because contrast across steps is preserved, boundaries are not shifted, and outliers are efficiently removed, without introducing unrealistic values. However, median filtering is more time consuming than smoothing filtering.

2.1.3 Gaussian filtering

Gaussian smoothing is a convolution operation that works as a low-pass filtering. It is often applied in order to remove noise in many practical cases, even if noise having a Gaussian probability distribution is an ideal case. A 2-D isotropic Gaussian distribution has the form:

$$G(x, y) = \frac{1}{2\pi\sigma^2} \exp^{\frac{x^2+y^2}{2\sigma^2}}$$

It is shown in figure 2.2.

A two dimensional discrete Gaussian kernel must be defined in order to work with pixels of an image. By "kernel" is meant a matrix of values to be used for weighted sums or

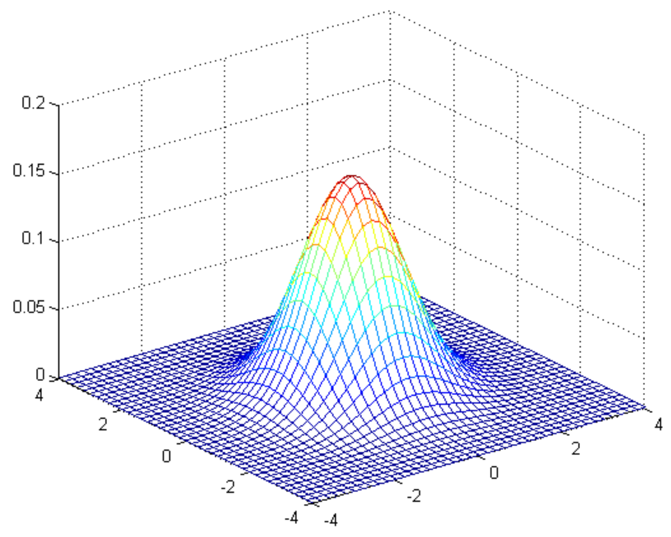


Figure 2.2: 2D Gaussian distribution with mean (0,0) and $\sigma = 1$.

integrals. In this case, it results from the sampling of a 2D Gaussian distribution and it represents a discrete approximation of that function. Although the Gaussian distribution is non-zero everywhere, we can assume that it is almost zero three standard deviations away from the mean. This fact allows us to truncate the kernel in order to keep a finite and reasonable dimension for it, and links the kernel dimension to the standard deviation of the considered Gaussian distribution.

A suitable kernel 5×5 deriving from a Gaussian distribution with a standard deviation of 1.0 is for example

$$\frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix} \tag{2.2}$$

Note that the mask is normalized, i. e. the sum of the coefficients is one, thus not introducing an intensity bias.

The Gaussian filter has some important properties:

- Isotropy.
- Monotonically decreasing both in spatial and frequency domains.
- Separability.

Isotropy This property guarantees an identical smoothing in all the directions, thus ignoring the orientation of structures in the image. This fact can be shown with a change of coordinates from Cartesian (x, y) to polar ones (r, θ):

$$G(r, \theta) = \frac{1}{2\pi\sigma^2} \exp^{-\frac{r^2}{2\sigma^2}}$$

where $r = x^2 + y^2$. As we can observe, the Gaussian does not depend on the angle θ.

Monotonically decreasing in frequency domain The Fourier transform of a Gaussian is a real-valued Gaussian itself. Let us prove this property in the one-dimensional case.

$$\begin{aligned}
F(G(x)) &= \int_{-\infty}^{+\infty} G(x) \exp\left(-jux\right)\, dx = \\
&= \int_{-\infty}^{+\infty} \exp\left(-\frac{x^2}{2\sigma^2}\right) \exp\left(-jux\right)\, dx = \\
&= \int_{-\infty}^{+\infty} \exp\left(-\frac{x^2}{2\sigma^2}\right) \left(\cos ux + \sin ux\right)\, dx = \\
&= \int_{-\infty}^{+\infty} \exp\left(-\frac{x^2}{2\sigma^2}\right) \left(\cos ux\right)\, dx + j\int_{-\infty}^{+\infty} \exp\left(-\frac{x^2}{2\sigma^2}\right) \left(\sin ux\right)\, dx = \\
&= \int_{-\infty}^{+\infty} \exp\left(-\frac{x^2}{2\sigma^2}\right) \left(\cos ux\right)\, dx = \\
&= \sqrt{2\pi}\sigma \exp\left(-\frac{u^2}{2\nu^2}\right)
\end{aligned} \tag{2.3}$$

(see [30], chapter 4 for the last equality) where $\nu^2 = \frac{1}{\sigma^2}$ represents the variance in the frequency domain. It is the reciprocal of the one in the spatial domain.
So, the higher is the value of the standard deviation in the spatial domain, the lower is the standard deviation in the frequency domain. If the smoothing is slight (low standard deviation in the spatial domain), the frequency cut off is high, so high frequencies (sharp variations) are kept in the processed image. On the contrary, for high values of the standard deviation in the spatial domain the smoothing will be greater and high frequencies will be removed, implying a greater attenuation of noise but also a greater removal of details.

Separability The convolution of the 2-D Gaussian filter with an image can be split in two convolutions with two one-dimensional filters, as we show below:

$$\begin{aligned}
G(x,y) \otimes f(x,y) &= \sum_l \sum_k G(l,k) f(i-l, j-k) \\
&= \sum_l \sum_k \exp\left(-\frac{l^2+k^2}{2\sigma^2}\right) f(i-l,j-k) \\
&= \sum_l \exp\left(-\frac{l^2}{2\sigma^2}\right) \left[\sum_k \exp\left(-\frac{k^2}{2\sigma^2}\right) f(i-l,j-k)\right]
\end{aligned} \tag{2.4}$$

As we can see the convolution can be performed first with a mono-dimensional Gaussian in the vertical direction, then the result is convolved with a mono-dimensional Gaussian in the horizontal direction.

Because of the separability property, the convolution with the kernel of Eq. 2.2 can be replaced by two consecutive convolutions, one with the kernel

$$\frac{1}{17}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

and the second with its transposed.
So, we have reduced the computational complexity from $\mathcal{O}(j^2 \cdot mn)$ - where $j \times j$ is the dimension of the kernel, and $m \times n$ is the dimension of the image - to $\mathcal{O}\left((j+j) \cdot mn\right)$.

2.2 Edge detection

What is an edge? An edge is a sharp variation of intensity in an image. More exactly, the mathematical definition states that an edge point is a point in which the first derivative (in the one-dimensional case) of a function is maximum.
In the case of an image, we should talk about the gradient of the image rather than the first derivative, but the concept is the same. We should find the points of steepest slope on a curve running in the gradient direction.

Edge detection is a fundamental task in computer vision, and usually it is the first step before further manipulation of the images. It can be seen as a case of feature extraction.
The aim of edge detection is extracting an "edge image" from an RGB or a grey scale image (as an RGB image can be converted into three grey scale ones, we will concentrate on the second kind of images), in order to remove redundant information.
The edge image will contain only the contours, or edges, that provide more information than the original image. The edge image is a black and white image, in which 0 pixels correspond to everything that is not a contour, and 1 pixels correspond to the contours.

However, the edges obtained with most of the algorithms are usually not completely satisfactory, as there are some discontinuities in certain parts of a contour, and usually the detected edges are many more than the required ones (there are a lot of useless edges, often deriving from noisy details or insignificant structures in the original image).
So, after edge detection, we should perform a cleaning step in order to remove redundant edges, and a linking step in order to connect different parts of the same logical contour, if it results discontinuous.

There are two main strategies of edge extraction: the gradient based edge detection and the Laplacian based edge detection.
The gradient based method looks for local maxima in the first derivative of the image. The first derivative is positive at dark to bright transitions, and it is negative at bright to dark transitions. It is zero in constant grey level zones.
The Laplacian based method looks for zero crossings in the second derivative of the image. The second derivative is positive on the dark side of a contour, and negative on the bright side, crossing the zero level at the transitions.

The gradient ∇f is a two-dimensional vector (while the Laplacian is a scalar), having thus a direction, a verse and a magnitude.

$$\nabla f(x,y) = \begin{bmatrix} \frac{\partial f(x,y)}{\partial x} \\ \frac{\partial f(x,y)}{\partial y} \end{bmatrix}$$

$$\theta(x,y) = \arctan\left(\frac{\frac{\partial f(x,y)}{\partial x}}{\frac{\partial f(x,y)}{\partial y}}\right)$$

$$M(x,y) = |\nabla f(x,y)| = \sqrt{\left(\frac{\partial f(x,y)}{\partial x}\right)^2 + \left(\frac{\partial f(x,y)}{\partial y}\right)^2}$$

It is written as the couple (gx, gy), consisting of the horizontal and vertical first derivatives. Obviously they are only an approximation of the derivatives, because the intensity of a digital image is a discrete function.

The direction of the gradient is always the direction of maximum variation, perpendicular to the contour, the verse is positive going from darker to brighter regions, and the magnitude is maximum at maximum variations.

The first derivatives can be calculated with different types of operators, like Sobel, Roberts, Prewitt, Kirsch and Robinson, that provide quite different results.
Each operator uses two different masks (kernels), for examples the ones shown in Eqs. 2.5, 2.6, 2.7, respectively Sobel, Roberts and Prewitt.

$$K_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad K_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{2.5}$$

$$K_x = \begin{bmatrix} +1 & 0 \\ 0 & -1 \end{bmatrix} \qquad K_y = \begin{bmatrix} 0 & +1 \\ -1 & 0 \end{bmatrix} \tag{2.6}$$

$$K_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \qquad K_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \tag{2.7}$$

For example, if we consider a cross-correlation operation (while a convolution would use the mirrored masks), the application of the masks 2.5 gives respectively as results (see table 2.1):

$$\begin{aligned} f(i,j) = & [(P(i-1,j+1) - P(i-1,j-1)) + 2\cdot(P(i,j+1) - P(i,j-1)) \\ & + (P(i+1,j+1) - P(i+1,j-1))] \end{aligned}$$

$$\begin{aligned} f(i,j) = & [(P(i-1,j-1) - P(i+1,j-1)) + 2\cdot(P(i-1,j) - P(i+1,j)) \\ & + (P(i-1,j+1) - P(i+1,j+1))] \end{aligned}$$

(i-1,j-1)	(i-1,j)	(i-1,j+1)
(i,j-1)	(i,j)	(i,j+1)
(i+1,j-1)	(i+1,j)	(i+1,j+1)

Table 2.1: Indices of pixels involved in Eqs. 2.5-2.7

The x axis is oriented in the ordinary way, but the y axis is positive in downward because of a common convention in pixels representation (pixel (0,0) in the upper left corner).

Kirsch and Robinson masks (equations 2.8 and 2.9) are called compass masks because they are obtained from the rotation of a single mask to the eight major compass orientations (N, NW, W, SW, S, SE, E, NE). In this case the gradient magnitude in every pixel is the maximum value among the convolution of each mask with the image, while the direction is given by the mask that leads to the maximum magnitude.

$$\begin{aligned} &K_n = \begin{bmatrix} 5 & 5 & 5 \\ -3 & 0 & -3 \\ -3 & -3 & -3 \end{bmatrix} \quad K_{ne} = \begin{bmatrix} -3 & 5 & 5 \\ -3 & 0 & 5 \\ -3 & -3 & -3 \end{bmatrix} \quad K_{nw} = \begin{bmatrix} 5 & 5 & -3 \\ 5 & 0 & -3 \\ -3 & -3 & -3 \end{bmatrix} \quad K_e = \begin{bmatrix} -3 & -3 & 5 \\ -3 & 0 & 5 \\ -3 & -3 & 5 \end{bmatrix} \\ &K_w = \begin{bmatrix} 5 & -3 & -3 \\ 5 & 0 & -3 \\ 5 & -3 & -3 \end{bmatrix} \quad K_s = \begin{bmatrix} -3 & -3 & -3 \\ -3 & 0 & -3 \\ 5 & 5 & 5 \end{bmatrix} \quad K_{se} = \begin{bmatrix} -3 & -3 & -3 \\ -3 & 0 & 5 \\ -3 & 5 & 5 \end{bmatrix} \quad K_{sw} = \begin{bmatrix} -3 & -3 & -3 \\ 5 & 0 & -3 \\ 5 & 5 & -3 \end{bmatrix} \end{aligned} \tag{2.8}$$

$$\begin{aligned} &K_n = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \quad K_{ne} = \begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{bmatrix} \quad K_{nw} = \begin{bmatrix} 2 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & -2 \end{bmatrix} \quad K_e = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \\ &K_w = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} K_s = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad K_{se} = \begin{bmatrix} -2 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 2 \end{bmatrix} \quad K_{sw} = \begin{bmatrix} 0 & -1 & -2 \\ 1 & 0 & -1 \\ 2 & 1 & 0 \end{bmatrix} \end{aligned} \tag{2.9}$$

The sensitivity of these operators depends on the size of the masks used. The larger is the mask, the less the edge detector will be sensitive. Sobel operator and Prewitt's operator give better results in finding vertical and horizontal edges, while Robert's operator is designed to detect edges lying at 45° to the pixel grid, but it is more sensitive to noise. Sobel operator also smooths the image while deriving it (it emphasizes the pixel closer to the center of the mask), so it is less sensitive to noise.

Another gradient operator that performs some smoothing on the image is the derivative of the Gaussian (DroG).

Its two components can be calculated deriving the Gaussian with respect to x and y axis.

$$\frac{\partial h}{\partial x} = -\frac{x}{\sigma^2} \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right) \qquad \frac{\partial h}{\partial y} = -\frac{y}{\sigma^2} \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$$

The respective masks can be obtained sampling these two functions.

Unlike the gradient operator, the Laplacian operator is defined as

$$\Delta^2 f(x,y) = \frac{\partial^2 f(x,y)}{\partial x^2} + \frac{\partial^2 f(x,y)}{\partial y^2}$$

Three examples of Laplacian masks are:

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \begin{bmatrix} -2 & 1 & -2 \\ 1 & 4 & 1 \\ -2 & 1 & -2 \end{bmatrix}$$

There are a lot of strategies and algorithms to perform the operation of edge detection. The performance of each algorithm may change dramatically according to the quality of the image. Above all, it is very difficult to adapt an edge detector so that it can work properly in different situations. There are some values in the edge detector that are difficult to be set automatically in order to give satisfying results with different data. So an algorithm can run almost perfectly with an image but it can be unsatisfactory with another one.

However, there is a trade off between sensitivity and accuracy of an edge detector: if it is very sensitive, probably it will detect many edge points due to noise, if it is not sensitive enough, it will miss true edge points.

In this work I have been using a Canny edge detector, which is considered one of the best performing algorithms in this field.

2.2.1 Canny edge detector

The edge detector suggested by J. F. Canny [8] was designed for satisfying the following properties:

- Good detection. The operator has a low probability of missing existing edges (high sensitivity) and a low probability of detecting spurious edges (high specificity).
- Good location. The operator should mark edges as close as possible to the real edges in the image.
- Minimal response. A real edge should be marked only once.

Canny formalized mathematically these three criteria in order to find the optimal solution. The fundamental steps performed in the Canny edge detection algorithm are the following ones, but some variations are possible in order to obtain better results.

- Filtering noise with a Gaussian filter
- Finding the intensity gradient of the image, both its magnitude and its direction
- Performing non-maxima suppression
- Hysteresis thresholding.

We have already spoken about **Gaussian filtering** of images. The only parameter that needs to be chosen is the standard deviation σ of the Gaussian distribution (and the dimension of the kernel, but in general it is related to σ). Increasing the width of the Gaussian mask implies lowering the detector's sensitivity to noise and an increased localization error in the detected edges. Usually the mask is much smaller than the image.

Computing the intensity gradient of the image can be done in several ways, and all of them lead to quite different results. For example, we can make use of the Sobel operator. The kernels

$$K_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad K_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

are applied to the image in order to obtain the approximation of the gradient in the x and y directions respectively, as said in the previous section.

The gradient magnitude can be calculated from its two components by the theorem of Pythagoras:

$$|M| = \sqrt{g_x^2 + g_y^2} \tag{2.10}$$

Also the orientation of the gradient can be obtained with the following equation using the x and y components of the gradient:

$$\theta = \arctan \frac{|g_y|}{|g_x|} \tag{2.11}$$

Often the direction of the gradient is assigned to one of 4 default directions, for example see figure 3.4.

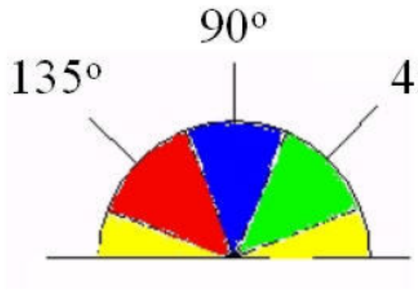


Figure 2.3: The gradient directions may be assigned to one of these default directions.

The gradient vector represents the direction and the magnitude of the fastest growth of intensity. The direction of the edge, that is tangent to the contour, is almost perpendicular to the gradient direction.

The next step is the so-called "**non-maxima suppression**". The purpose of this operation is that of setting to zero all pixels that do not represent a local maximum, obtaining if possible 1-pixel wide contours and removing spurious edges, also trying to preserve connectivity of the contours.
For each pixel (i, j), we have to check the gradient magnitude of its neighbouring pixels lying in the gradient direction. If (i, j) is a local maximum we keep $M(i, j)$, otherwise we set $M(i, j) = 0$ (with M meaning the gradient magnitude).

After non-maxima suppression, the final step is **hysteresis thresholding**. Some noisy local maxima are still present in the edge image, and we have to evaluate them exploiting the information given by the gradient magnitude.
So we choose the real edges by checking the edge strength.
However, usual thresholding with only one threshold may be too selective choosing a high threshold or too loose choosing a low one. We need to remove weak edges, but at the

same time we want to keep the connectivity of the contours. As a matter of fact, conventional thresholding based on the image histogram does not take into account the spatial information, while hysteresis thresholding does.
Two thresholds are needed, T_{high} and T_{low}. Every pixel (i, j) that is non zero after non-maxima suppression is classified as strong if $M(i, j) > T_{high}$, and weak if $M(i, j) \leq T_{low}$. Pixels with a value within the range $\{T_{high}, T_{low}\}$ are considered candidate pixels.
Every weak pixel is immediately discarded, and every strong pixel is accepted as an edge. We don't remove candidate pixels immediately, but postpone the decision after further investigations. We should follow the sequence of connected local maxima along the edge direction (perpendicular to the gradient direction) until the gradient magnitude is greater than T_{high}. If the point from which we started is connected in this chain to a strong pixel, it is upgraded to a strong pixel, otherwise it is discarded.

There is a component of subjectivity in the choice of the thresholds, above all because if we find good thresholds for some image, they could work bad with another image.
Canny suggested to choose the thresholds in order to satisfy the following constraint:

$$2 \leq \frac{T_{high}}{T_{low}} \leq 3$$

However, this is an empirical rule and the choice of at least one of the two thresholds is still subjective, and not so easy to be carried out. For example, the thresholds can be chosen from an initial set of possible values (e.g. sampling the whole range of intensity values in the image), on the basis of a visual comparison between the results. Obviously, this is a time consuming technique and the final choice is operator-dependent.

The thresholds can also be estimated adaptively, on the basis of the image we are working on. Several researchers have been working on this task in the last years, proposing different methods with more or less satisfactory results.

Adaptive hysteresis thresholding In recent years, various groups of researchers have looked for different strategies for finding suitable thresholds in an adaptive way, i. e. developing a method that takes advantage of the features of a specific image in order to tune the thresholds on that specific image. The goal is to achieve a completely unsupervised method for searching the thresholds.

We will briefly describe some of the most recent and significant works. The general idea is to perform an automatic statistical analysis on the results provided by different sets of given thresholds. The methods are quite complex to explain and describing them in detail goes beyond the purpose of this thesis, but we will make a brief outline of the main steps and ideas for each presented method.

Hancock and Kittler, in 1991, were the first to investigate on an adaptive estimation of hysteresis thresholds in the context of Canny edge detection [13]. They started from the assumption that the problem could be formulated in a Bayesian context, with a parametric approach. In their work, a model of the considered image is developed, taking into account both the filtering operations to remove noise, and the quantification of connected structures in the image by edge labelling. The hysteresis thresholds can thus be related to the parameters of the model by means of the Bayesian framework. What they deserve is the *a priori* probability $P(\theta_j = \epsilon)$ that a pixel j belongs to an edge, which is not easy to obtain, and the standard deviation σ of noise in the image.

The result of their work is the following:

$$T_{high} = 2\sigma\sqrt{f_u \ln 2} \qquad T_{low} = 2\sigma\sqrt{f_u \ln \frac{1 + 2P(\theta_j = \epsilon)}{1 + P(\theta_j = \epsilon)}} \tag{2.12}$$

where σ is the noise variance estimated from the mean gradient magnitude, $f_u = s - l$, with $s = \sum_{i=1}^{N} \sum_{j=1}^{N} a_{i,j}^2$ and $l = \sum_{i=1}^{N} \sum_{j=3}^{N} a_{i,j} \cdot a_{i,j-2}$, where the coefficients $a_{i,j}$ are taken from the kernel of the filter used in the pre-processing step.

The ratio between the two thresholds is

$$\frac{T_{high}}{T_{low}} = \sqrt{\frac{\ln 2}{\ln \frac{1+2P(\theta_j=\epsilon)}{1+P(\theta_j=\epsilon)}}}$$

that equals the ratio proposed by Canny 2:1 when the priors are in the ratio 1:4, i.e. the frequency of edge pixels is much smaller than the frequency of non-edge pixels.

Yitzhaky and **Peli** [33] more recently proposed another method that performs an automated statistical analysis among different detector parameters given as input, selecting thus the best set. They construct an EGT (estimated ground truth) with different results from the detection algorithm, compare them by means of a Chi-square test, and keep the best result as the best choice of the thresholds. The main problem with this strategy is the dependence of the result from the initial set of hysteresis thresholds employed.

Trying to fix this problem, **Medina-Carnicer** and **Madrid-Cuevas** in their first article in 2008 [25] proposed a method to determine initial candidates for hysteresis thresholds in an unsupervised way. One of the most interesting features of this method is its being completely non-parametric, thus not presupposing any model for the image under examination.

Given an image I, its gradient magnitude $M(I)$ (with values normalized in $[0,1]$, and the L-bins histogram, H, of $M(I)$, the main steps for the implementation of their method are:

1. Construct H_b^x for all $x \in \{1,\dots,L-1\}$ ($H_x^b(0) = \sum_{r=0}^{x-1} H(r)$, $H_x^b(1) = \sum_{r=x}^{L-1} H(r)$)
2. For each x value, and for all $y \in 1,\dots,L-1$, with $y \neq x$
   - if $x < y$
     $TP(H_y^b, H_x^b) = H_y^b(1)$
     $TN(H_y^b, H_x^b) = H_x^b(0)$
     $FN(H_y^b, H_x^b) = H_y^b(0) - H_x^b(0)$
     $FP(H_y^b, H_x^b) = 0$
   - else
     $TP(H_y^b, H_x^b) = H_x^b(1)$
     $TN(H_y^b, H_x^b) = H_y^b(0)$
     $FP(H_y^b, H_x^b) = H_y^b(1) - H_x^b(1)$
     $FN(H_y^b, H_x^b) = 0$
3. For each $x \in \{1,\dots,L-1\}$
   $\hat{TP}(x) = \frac{1}{L}\sum_{y=0}^{L-1} TP(H_y^b, H_x^b)$
   $\hat{TN}(x) = \frac{1}{L}\sum_{y=0}^{L-1} TN(H_y^b, H_x^b)$
   $\hat{FP}(x) = \frac{1}{L}\sum_{y=0}^{L-1} FP(H_y^b, H_x^b)$
   $\hat{FN}(x) = \frac{1}{L}\sum_{y=0}^{L-1} FN(H_y^b, H_x^b)$
   After this, set $\rho = \hat{TP}(x) + \hat{FP}(x)$ (they show that it is a constant)
4. Calculate $T_{low} = \frac{\rho(L-2)}{1+2\rho}$
5. T_{high} is the x threshold with $\hat{FP}(x) = \hat{FN}(x)$.

The thresholds obtained in this way are a gross approximation of the ideal thresholds, so this range can be used as a starting point to look for them.

The same authors, in two following articles, proposed methods to obtain optimal hysteresis. In the first article [26] they tried to threshold the images without directly determining the thresholds, while in the second [27], although the algorithm is very similar in the first steps, they also provide two thresholds as an output. Both articles assume that a set of candidate thresholds is known.

Let I be the image of interest, $M(I)$ an approximation of its gradient magnitude, $M_{high}(I)$ the edge image obtained after thresholding $M(I)$ with the high threshold. $M_{low,high}(I)$ is the edge image obtained after the hysteresis thresholding with the two given thresholds. A set of candidate hysteresis thresholds is known, $C(I) = (low, high) \in [0, 1]$. The main steps in the algorithm of the first article are:

1. Obtain the edge images $M_{low,high}(I)$ for each (low,high) couple in C, by hysteresis thresholding $M(I)$.

2. Obtain the edge images $M_{high}(I)$ for each (low,high) couple, thresholding $M(I)$ only with the high threshold.

3. Obtain $\Delta M_{low,high}(I) = M_{low,high}(I) - M_{high}(I)$.

4. Compute $SM = \sum_{(low,high)\in C(I)} \Delta M_{low,high}(I)$. In each pixel of this image the number of times that it has been added as an edge with all the candidate thresholds is contained.

5. Threshold the histogram of SM with a suitable threshold (for example Rosin's threshold if the histogram is unimodal, or Otsu method, if there are more modes) and obtain $Hyst_{SM}$.

6. Threshold $M(I)$ using sample values in the interval $[0, 1]$, starting from 0.01 and increasing with a step of 0.01, looking for the first edge image $M_{th}(I)$ that satisfies $M_{th}(I) \times Hyst_{SM} = 0$.

7. Find the solution in the edge image $Edge_{proposed}(I) = Hyst_{SM} + M_{th}(I)$.

The second algorithm is analogous until the fourth step, then the following operations are required:

- Obtain the distribution $P(F_l(x))$, with $x \in (0, 1)$, using these positions
  - $Prob(SM) = \frac{SM}{|C(I)|}$, where $|C(I)|$ is the cardinality of the set $C(I)$
  - $Prob_x(SM)$ is the binary image obtained by thresholding $Prob(SM)$ with $x \in (0, 1)$
  - $|Prob_x(SM)|$ is the number of pixels equal to 1 in the image $Prob_x(SM)$
  - $F_I(x) = M(I) \circ Prob_x(SM)$, with $\circ$ meaning the Hadamard product [1]
  - $|F_I(x)|$ is the number of pixels with grey level x in the image $F_I(x)$

- Finally, obtain the set $D = \{x \in (0, 1) | P(F_I(x)) \neq 0\}$, where

$$P(F_I(x)) = \begin{cases} \frac{|F_I(x)|}{|Prob_x(SM)|} & |Prob_x(SM)| > 0 \\ 0 & |Prob_x(SM)| = 0 \end{cases}$$

  The first and the last local maxima of the image $Prob(SM)$ for the values in D are the thresholds we were looking for.

[1]Hadamard product is an operation that generates a matrix of the same dimensions of the two original matrices, each element (i, j) of the resulting matrix is the product of corresponding elements (i, j) of the input matrices.

Finally, we quote another method, reported in Gonzales-Woods [10], for finding hysteresis thresholds adaptively. Given an image size, this method is thought to give an almost constant number of edge points as a response.
At first, the histogram of the gradient magnitude (fitted in the range [0,255]) after non maxima suppression image is computed. From this histogram we can extract the number of non-zero pixels that cannot be edge points with the following formula:

$$NumNonEdge = (1 - p) \cdot (M \cdot N - Hist[0])$$

where M and N are respectively the number of rows and columns in the image, while parameter p represents the ratio between the number of edge pixels and the number of pixels with a gradient magnitude different from zero ($NumEdgePixels = p \cdot (NumPixels - ZeroGradientModulePixels)$. The value of p usually lies in the range between 0.01 and 0.1.
Excluding the first bin of the histogram, its values are summed until the sum reaches the value that was calculated before, $NumNonEdge$. The reached index i is the high threshold, and the low threshold can be determined multiplying it by a parameter $k < 1$, for example 0.4 to be in agreement with Canny's suggestions.

2.2.2 Edge linking (Gap linking)

Detected contours are typically discontinuous even if we have applied all precedent rules in order to guarantee their continuity. Edge linking or contour tracking can help joining separate edge segments according to some criteria. Also in this field, a huge number of different approaches is available, and for sure the research is still open to new strategies and proposals.
There are two main categories of methods for performing edge linking:

- Local methods. Starting from the end pixel of an edge segment, extend it by looking for the most suitable candidate pixel in its neighbourhood.
- Global methods. They perform a wider search on the whole image, but they are also more computationally expensive.

Although faster to be run, local methods fail in bridging big gaps.

Local edge linking methods Contour linking is based on some observations. For example, along boundaries both the image and the gradient have an approximately constant intensity, edges tend to have a low curvature, and there is a small difference between adjacent edge directions.
The strategy is to look in a neighbourhood (e.g. a 3×3 square) of every pixel that represents an endpoint for an edge segment, and link the central pixel with the neighbour (or neighbours) that satisfies some similarity constraints like the following

$$|M(p_i) - M(p_j)| \leq T_1$$

$$|\phi(p_i) - \phi(p_j)| \leq T_2$$
$$|M(p_j)| \geq T_3$$

i.e. the absolute difference between the magnitudes and the directions of two consecutive pixels should be smaller than a given threshold, and the magnitude of the new pixel should be large enough.

Heuristic search, for example, is based on graph searching: edge points correspond to graph nodes and they are connected to each other if the local rules written before are satisfied. The idea is to generate different paths between two possibly connected edge points and evaluate their cost by means of a cost function. The chosen path will be the one that leads to the minimum of the cost function. The factors that influence the cost can be for example the same used for the similarity constraints.

Dynamic programming can be used to search for maxima or minima of some function by subdividing the problem recursively into sub-problems. For example, in graph searching we can split a path into two optimal sub-paths recursively, and look for the minimum of the cost function in each of them, until we find the base case.

Global edge linking methods If the gaps between ending points of edge segments are wide, we should take into account global methods.

Hough transform can be used to detect shapes and object boundaries in an image, especially when they lie on curves of a specified shape, like lines, circles, ellipses, etc. (it is a model-based technique). We now briefly describe the method for detecting straight lines because we will use it in chapter 3.

A straight line is uniquely identified by two points $A(x_1,y_1)$ and $B(x_2,y_2)$. If we consider straight lines passing through point A, we can write equation $y_1 = kx_1 + q$ for every real k and q. The main idea of the method is to switch from the image space (x, y) to the parameter space (k, q), where the previous equation can be written as $q = -x_1k + y_1$. Straight lines conducted from point B are in the form $q = -x_2k + y_2$. If we intersect these two straight lines, the point (k', q') that we get is the couple of parameters representing the straight line passing through both point A and point B.
So, every straight line is uniquely identified by a point in the parameter space. To perform the recognition of straight lines in the image, at first all possible line pixels must be detected by means of an edge detector with a proper threshold. The parameter space must be discretized in a limited number of values to be handled easily. All straight lines that can go through the detected line pixels must be transformed in the parameter space, and finally the points of the parameter space that occur more frequently identify the straight lines that can be found in the image.
To avoid problems rising from vertical lines ($k \to \infty$), the line representation is indeed $s = x\cos\theta + y\sin\theta$.

Deformable models (Snakes) or **active contours**, on the contrary, are used to link contours of objects having arbitrary shapes. The underlying idea comes from Kass, Witkin, and Terzopoulos [17], and it is based on the concept of energy-minimizing spline. Active contours, or snakes, are characterized by an energy that depends on their shape and position in the image. A starting position and shape must be defined, then the snake will warp and move on the image according to the minimization criterion.
The energy function to be minimized is a weighted sum of internal forces, depending on the snake's shape, and external forces, depending on the image.

2.2.3 Performance evaluation of edge detectors

Although edge detection has been a very much wrought field in the last decades, no standardized method for evaluating edge detectors' performances are still available. Actually some methods were proposed of course, and they can be divided into two main categories: subjective methods, and objective methods.

Subjective methods are based on the direct observation of the extracted edges, and they are thus totally dependent on human judgement. The observer should visually detect differences between various operators, and he should also decide whether detected edges are real or spurious. This technique is simple to put into practice, but its faults are at the same time evident. The human eye cannot distinguish every detail of an image, besides results given by an observer cannot be compared with another one's judgement.

Objective methods are more reliable, because they are not operator-dependent, but they still raise some problems in their calculations. Two frequently used parameters are root mean square error (ERMS) and signal to noise ratio (SNR), defined as follows [16]:

$$ERMS = \sqrt{\frac{1}{M \cdot N} \sum_{r=1}^{M} \sum_{c=1}^{N} [E(r,c) - O(r,c)]^2} \tag{2.13}$$

$$SNR = \sqrt{\frac{\sum_{r=1}^{M} \sum_{c=1}^{N} [E(r,c)]^2}{\sum_{r=1}^{M} \sum_{c=1}^{N} [E(r,c) - O(r,c)]^2}} \tag{2.14}$$

where M and N are respectively the number of rows and columns of the image, $O(r,c)$ is an image containing true edges, and $E(r,c)$ is the image with detected edges.
Another useful parameter is the total localization error (LE)

$$LE = \frac{|\sum \text{pixels of exact edges} - \sum \text{pixels of output edges}|}{N} \cdot 100 \tag{2.15}$$

The main problem with these measurement techniques is the image $O(r,c)$, containing real edges. As a matter of fact, we often don't know what is a real edge and what is a spurious edge in an image. A common adopted way is to compare the edge image with the so called *ground truth*, i.e. a known reference image. We usually know them only for synthetic images, that are not so reliable in testing an edge detector as a real image can be.

In order to integrate subjective and objective evaluation, Heath et al. [14] in their article tried to combine human judgement with a statistical approach. They submitted the output of different edge detectors to a relevant number of trained observers, and on the basis of their responses (they were supposed to give a rate to the detected edges) they have developed statistics on different edge detectors' quality.

2.3 Template matching

Template matching refers to a search of a model pattern (the so called "template"), that represents the feature we want to detect, within a given image. The template (t) is slid all over the image, and the level of similarity with each possible position in the image is measured. The size of t is usually much less than the target image size.

One of the measures of similarity most often used for this purpose is the cross-correlation between the template and each portion of the image of the same dimension as the template. The region with the maximum value of correlation is chosen to contain the feature we are looking for.

Given a grey-value image f of size $M \times N$ and a template t of size $J \times K$, the correlation at the position (i,j) is defined as

$$R(i,j) = \sum_x \sum_y f(x,y) t(x-i, y-j) \tag{2.16}$$

where $1 \leq i \leq M$ and $1 \leq j \leq N$, and the summation is taken over the region of the image covered by the template. Computing Eq. 2.16 for each position (i,j) in the image, the maximum value of $R(i,j)$ indicates the position of the best match between the image and the template.

In order to overcome the dependence of $R(i,j)$ on the changes in intensity of the image, the normalized correlation coefficient is usually computed:

$$r(i,j) = \frac{\sum_x \sum_y \left[f(x,y) - \overline{f}(x,y)\right] \left[t(x-i, y-j) - \overline{t}\right]}{\sqrt{\sum_x \sum_y \left[f(x,y) - \overline{f}(x,y)\right]^2 \left[t(x-i, y-j) - \overline{t}\right]^2}} \tag{2.17}$$

with the same convention of Eq. 2.16, $\overline{f}(x,y)$ represents the local average of the image in the region covered by t, and $\overline{t}$ is the average value of t, computed only once.
This coefficient has an absolute value between 0 and 1.

Chapter 3

Extraction of the velocity profile

The following chapter outlines what has been done on the available images, i.e. what techniques among those described in the previous chapter have been employed and how they have been implemented.

The set of eighteen images studied in this work was provided by Britta Lind and Patric Jonsson from KTH (STH). These images were collected from exams performed on real patients, and they were stored thanks to the software EchoPAC^{TM} (Fairfield, Connecticut). This software allows the export of images in DICOM or RAW format. We have been using DICOM images, a format easily handled by Matlab thanks to the built-in function *dicomread.* All the considered images have a resolution of 434×636 pixels (they were already compressed by the software with a lossy compression ratio of 8.2). They are acquired as RGB images (because of some coloured parts like the Colour Doppler section on the top of the images) and then converted into grey-scale images. The images contain the spectral Doppler, a colour-Doppler part that shows the region of interest, some parameters of the machine, the heart rate, and the ECG line.

All of the following operations are carried out in Matlab (Natick, Massachusetts). Matlab functions are denoted by italics preceded by the word "function" the first time they are mentioned.

Matlab code can be found in Appendix A, while results of all the employed techniques are presented in chapter 4, where some resulting images are shown and commented.

3.1 Methods

Here follows a scheme that summarizes the alternative strategies adopted for the extraction of the velocity profile described in this chapter.

1. EDGE DETECTION AND LINKING OF GAPS:
   (a) Cleaning and enhancing the image
   (b) Canny edge detection
   (c) Gap linking
      - Exploiting gradient magnitude and grey value
        or
      - LBE method
2. STRONG PRE-PROCESSING
   (a) Magagnin or Kiruthika method, or region growing

   (b) Edge detection

3. LEVEL CURVES

   (a) Cleaning and enhancing the image

   (b) Tracing level curves

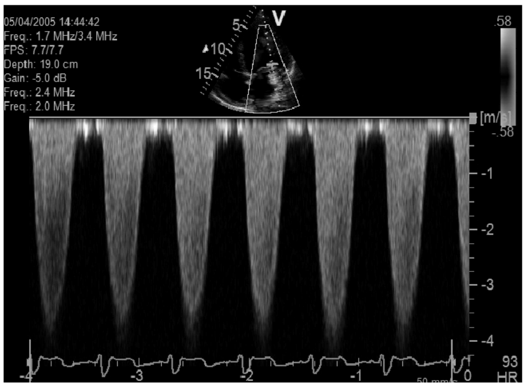
3.1.1 Edge detection and linking of gaps

In this section it is described how the strategy focused on extraction of the velocity profile by means of an edge detector, followed by a gap linking step has been put into practice.

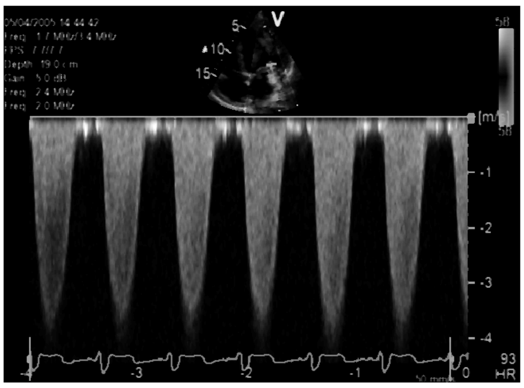
Cleaning and enhancing the image

After reading the image, and converting it from RGB to grey-scale, one or more cleaning steps must be carried out in order to remove noise from the image and enhance its contrast.

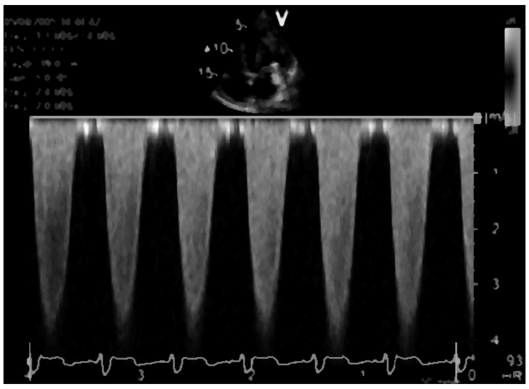
A median filtering on a 3×3 neighbourhood of each pixel is applied. This operation, as already said in 2.1.2, allows to remove speckle noise without altering the position of the edges, and thus can be repeated n times. In figure 3.1 it is shown how the repetition of the median filtering affects the image. However, we have been using images processed with the median filter only once.



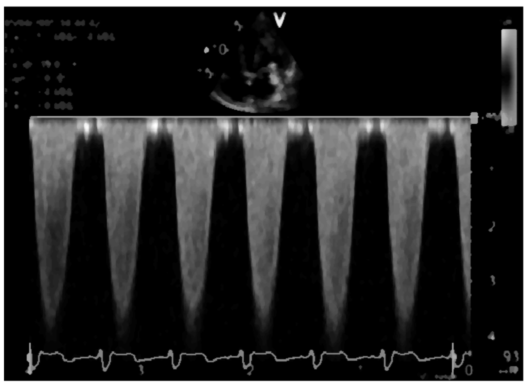
(a)



(b)



(c)

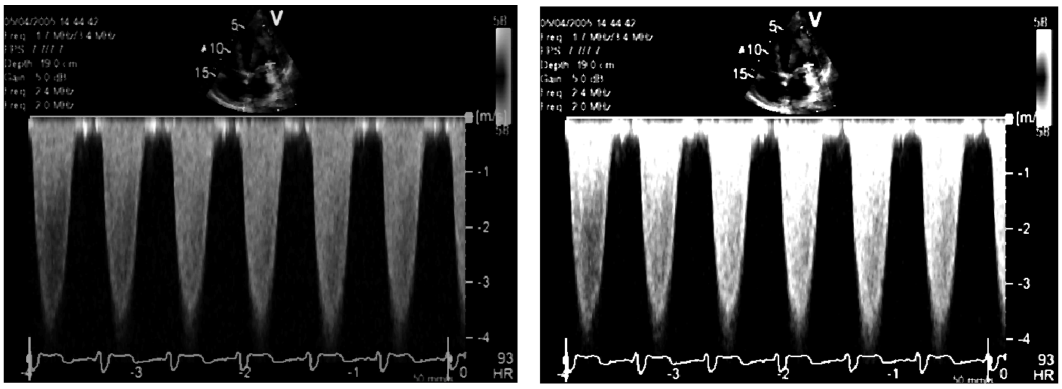


(d)

Figure 3.1: A Doppler sonogram to which a median filter on a neighbourhood 3×3 has been applied. **(a)** the original image, **(b)** the image after one iteration, **(c)** the image after 5 iterations, **(d)** the image after 10 iterations.

As a subsequent passage, a contrast stretching step has been applied. Referring to 2.1.1 and figure 2.1, the thresholds k_0 and k_f are obtained by means of a clustering operation on the grey value of pixels. The pixels of the image are divided into two clusters through a *k-means* algorithm (function *kmeans* in Matlab), that gives as an output the two centroids of the clusters created, used then as the looked-for thresholds. Equation 2.1 is applied and a contrast stretched image is obtained.

Giving two manually chosen fixed thresholds as k_0 and k_f would have lead to good results for some images, but bad ones for others. So the clustering step has been applied in order to make the operation more easily adaptable to different images. Because of the huge presence of background in this kind of images, the first centroid (used as k_0) is low enough to prevent too many significant pixels being set to zero in the stretching procedure, while the second centroid is not too high because only few pixels have an high intensity.



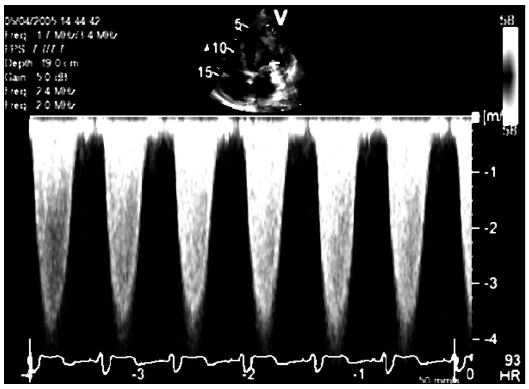
(a) (b)

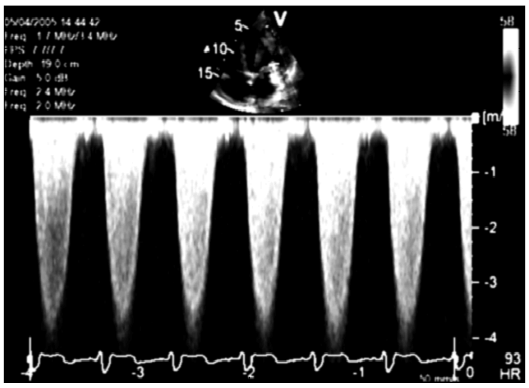
Figure 3.2: A Doppler sonogram before **(a)** and after **(b)** the contrast stretching step with $k_0 = 4.45$ and $k_f = 106.36$. After contrast stretching we can see that a greater number of pixels have a higher intensity than before, while the background is almost homogeneously zero. The image is the same of that in figure 3.1.

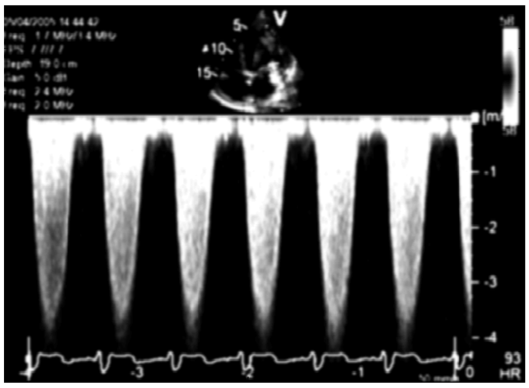
In figure 3.2 the effect of the contrast stretching operation can be observed: pixels lying in the region of interest are brighter than before, while background pixels are set to zero, except for some minor areas.

As a pre-processing before applying the Canny edge detector, a Gaussian filter has to be convolved with the image. The choice of the standard deviation σ of the kernel is not so trivial. A high value for σ would lead to an over-smoothed image, with a consequent shift of the detected edges from their original position, although they would be more connected and smooth. A low value for σ prevents undesired translations of edges but implies discontinuities in the detected edges. A compromise should be clearly chosen in order to guarantee the greatest possible level of continuity of the edges and at the same time keep them in their original place. A standard deviation of 1.5 and a mean of 0 have been chosen in the current work for all tested images.

Figure 3.3 shows the result after the application of Gaussian filters differing by their standard deviations, and their dimensions as well (the dimension of the applied kernel depends on the chosen standard deviation, see 2.1.3). In particular, the convolution was executed exploiting the separability property of the Gaussian distribution, making it more computationally efficient. Two mono-dimensional kernels of dimension $5{\cdot}\sigma{\cdot}2{+}1$ are obtained sampling the normal distribution on the values $[-5 \cdot \sigma, 5 \cdot \sigma]$ (97% of the area under the Gaussian curve is thus contained in the vector), and they are convolved one after the other with the image.


(a)


(b)


(c)

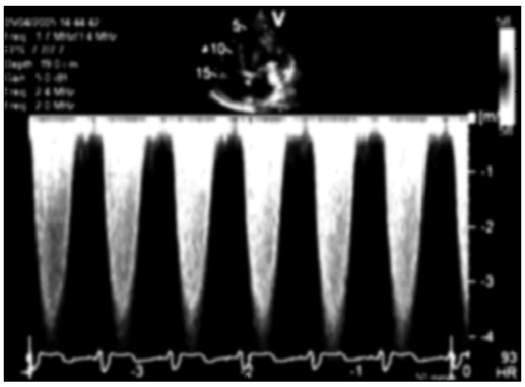

(d)

Figure 3.3: The image after the application of four Gaussian filters with different standard deviations. **(a)** $\sigma = 0.2$, **(b)** $\sigma = 0.8$, **(c)** $\sigma = 1.4$, **(d)** $\sigma = 2$. The image is the same of that in figures 3.1 and 3.2.

Canny edge detection

After we have cleaned and enhanced the image, we can perform the edge detection step, in particular implementing Canny's algorithm (see 2.2.1 for details).

First of all, the filter realizing the derivative of the mono-dimensional Gaussian used in the cleaning step is created. This filter is then convolved first with the image to obtain the x-component of the gradient, and then with the transposed version of the image to obtain the y-component of the gradient (g_x and g_y).

The gradient magnitude can be calculated from its two components according to equation 2.10, as well as the gradient direction. The latter is discretized in four main directions showed in figure 3.4.

Non-maxima suppression (cf. figure 3.5) is realized with an interpolation technique, that exploits the value d as a weight for near pixels. The value d is defined as the cotangent of the gradient direction if we are considering direction 1 or 2, or as the tangent otherwise. For example, in figure 3.5, the gradient direction is 2, according to the discretization of figure 3.4, so d is the cotangent of the angle between the two components of the gradient. If the gradient magnitude in the considered pixel is greater than the weighted sum of the two pixels on his left and on his right along the gradient direction, it is kept as a valid edge point, otherwise, it is set to zero.

Finally, hysteresis thresholding is performed thanks to the function *bwselect*, with an eight-connectivity setting. In practice, pixels above the low threshold are found in the

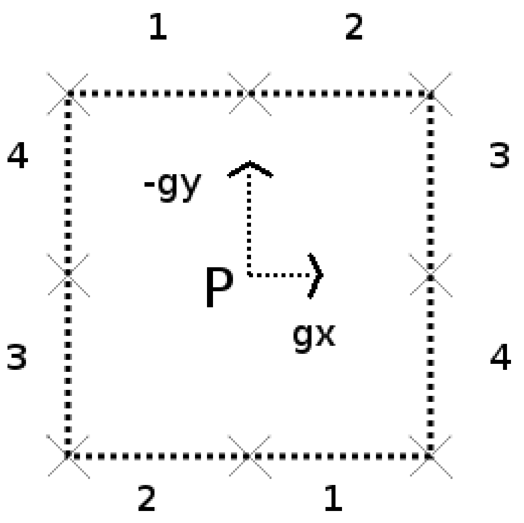


Figure 3.4: The four directions in which the gradient direction has been discretized.

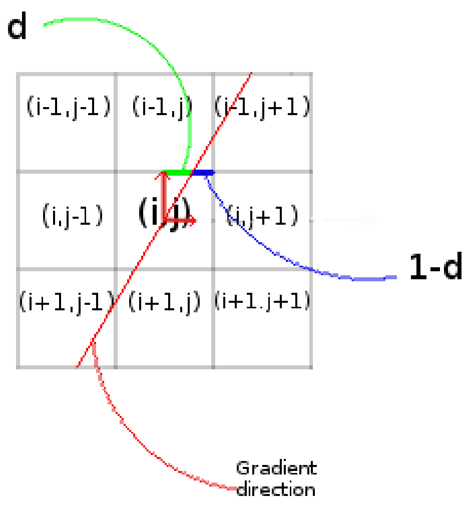


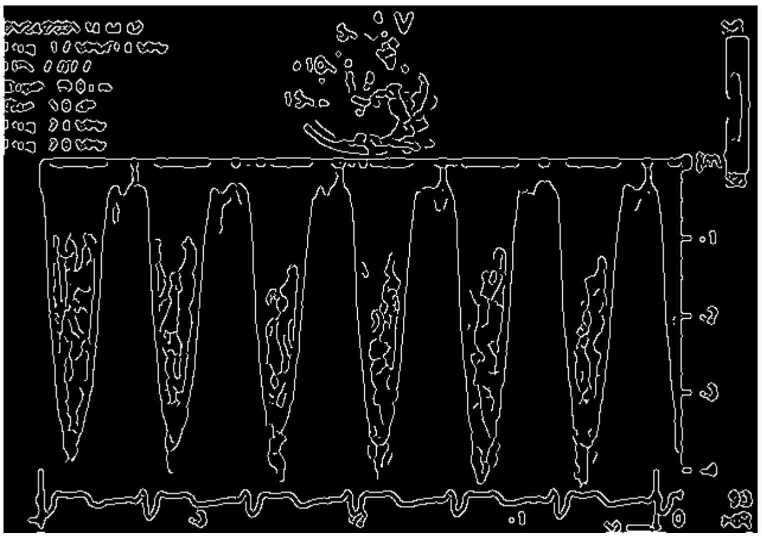
Figure 3.5: Illustration of non maxima suppression. The gradient direction (2, according to the discretization) is highlighted in red. Pixel $(i-1, j+1)$ is weighted with d, while pixel $(i-1, j)$ is weighted with $1-d$. The same occurs with pixels $(i+1, j-1)$ and $(i+1, j)$.

gradient suppressed image, as well as pixels above the high threshold, and then they are connected with the help of *bwselect*.

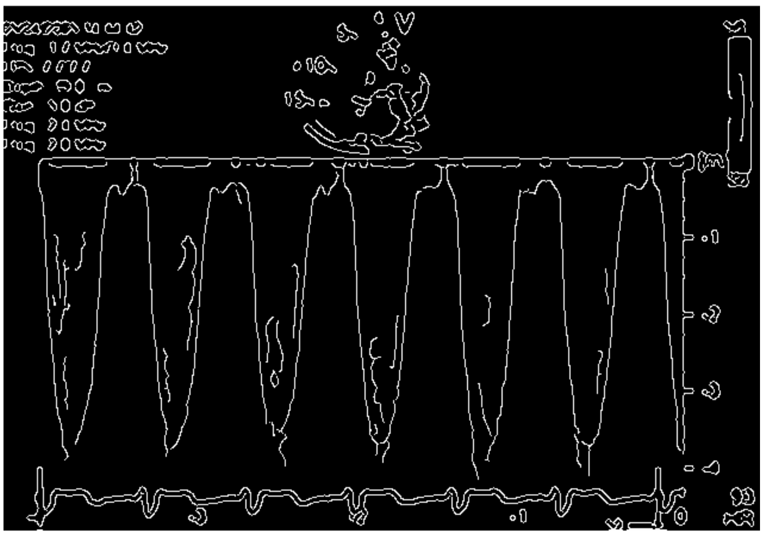
The output for the image we are considering is something like what we can see in figure 3.6. As expected, when we use a higher upper threshold we get less edges than with a lower one.

In order to make the algorithm more user-independent, the author tried to implement Medina et al. [27] method (see 2.2.1) for finding the hysteresis thresholds in an automatic way. Unless the code is quite slow, the thresholds found by it are quite satisfying. Also Gonzales and Woods' method [10] shows good results, with a value of p (see 2.2.1 for details) set to 0.3. We can see both the results in figure 3.7: the first method is more selective than the second one, with two thresholds significantly higher than the other two.

Main problems in the edge detection algorithm As we can see from figure 3.6, most of the gaps in the detected edges are situated in those areas that result more faded, where there is not a clear distinction between the noisy background and the velocity information. Where does the velocity information end and where does the background start? It is not so easy to give an answer, neither with eyes nor with a computer. What we find will always be an approximation. Canny's algorithm, as well as almost every other edge detector, fails



(a)



(b)

Figure 3.6: Two results of the Canny edge detector after all the pre-processing steps described before, with different hysteresis thresholds. On the left the high threshold is 0.117, and the low threshold 0.04, on the right the high threshold is 0.273 and the low threshold is 0.109.

to recognize edges there because there is a long and slow transition from brighter to darker regions without a maximum slope.

Another problem that may occur with the detection of edges is when there is a long gradual transition in the gradient direction followed by a sudden steep slope. Only the last one will be detected, unless we can speak about "edge" (even if in a broad sense) also for the gradual transition zone, that we may call "shoulder" instead of edge.

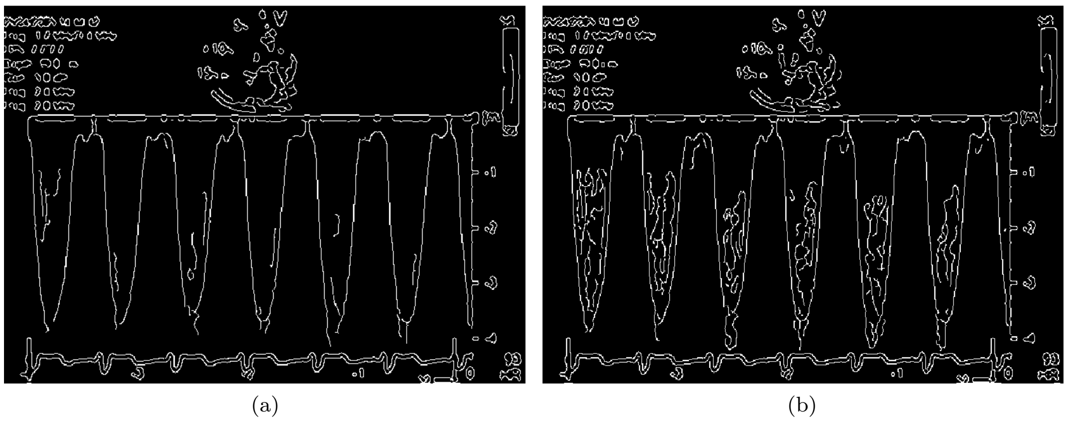


Figure 3.7: Two different adaptive hysteresis thresholds applied to the Canny edge detector. **(a)** Gonzales and Woods' method (t_{high}=0.248, t_{low}=0.0992), **(b)** Medina et al. method (t_{high}=0.14, t_{low}=0.011).

Edge linking (gap linking)

After the extraction of the edges with a Canny edge detection algorithm, we can see that the contour is not connected in every part, making it difficult to obtain useful information like the maximum velocity envelope (MVE) or the mean velocity from the Doppler image. The idea is to start from the endpoints of detected edges, i.e. the points of discontinuity in the traced contour, and extend the contour exploiting information given by the edge direction, the gradient magnitude, and other features of the image.

First of all, the ending points of every connected part of an edge must be recognized and marked in order to be used as starting points for the following operations. To this purpose, the Matlab function *endpoints* written by F. Bergholm [6] was used. To correctly run the code, we need to label previously all the connected parts in the edge image, and we do this by means of the function *komponentm4*, also written by F. Bergholm [6]. Every connected part of the edge image is given an identification (id) number, and this can be displayed graphically with the help of different colours (figure 3.8).

After generating the list of the id numbers present in the edge image, without repetitions and excluding the background (which is given an id of 0 by default), it is given as input to the function *endpoints*, together with the edge image and the value assumed by pixels both in its background (0) and in its endpoints (1). This function recognizes an endpoint as such, when surrounded by six or more background pixels. In particular, when the number of background neighbours is more than six, i.e. seven or eight, the pixel is immediately recognized as an endpoint, while when the number of background neighbours is exactly six further controls are performed. Neighbour pixels are investigated in a spiral clockwise way, and only if non-background pixels are contiguous, the examined pixel is considered an endpoint.

The output of *endpoints* is displayed thanks to the function *viewEndpoints*, and the result is shown in figure 3.9.
Once we get the coordinates of the endpoints, we can start from them to continue the interrupted paths of the edges. Two different strategies have been followed in order to reach our goal.

Exploiting edge directions, gradient magnitude and grey value of the pixels (*edgelinking.m*) Assuming the direction perpendicular to the gradient as the direction along which edges are expected to be found, for the following operations we refer to "edge direction", meaning the direction perpendicular to the gradient one. It is a local direction,

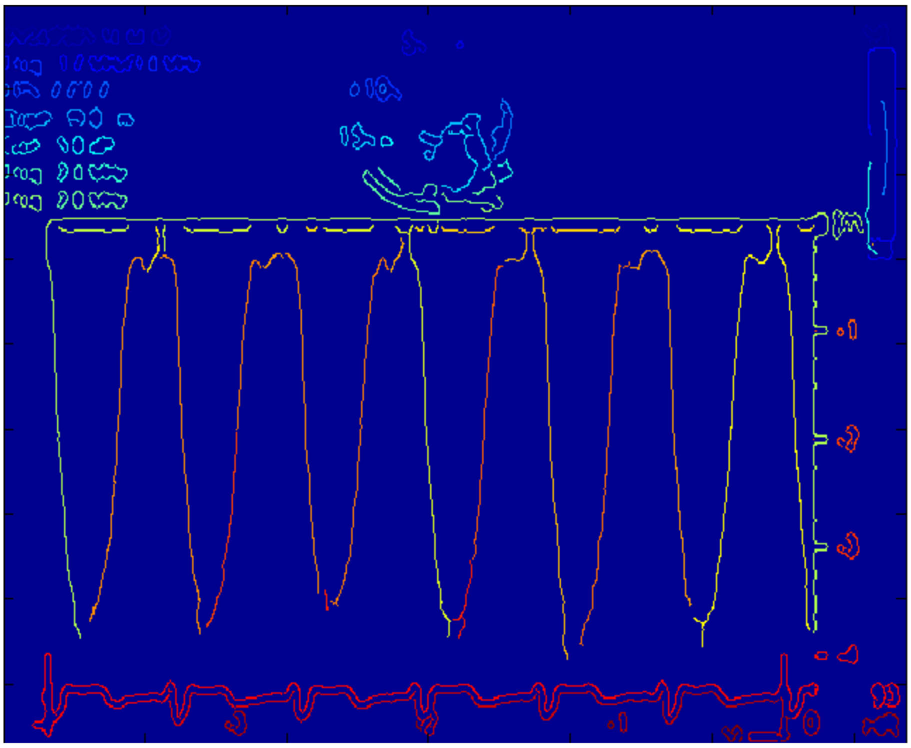


Figure 3.8: Example of the output of the function *komponentm4*. All pixels lying on the same connected portion of an edge are marked with the same colour, that corresponds to its id number.

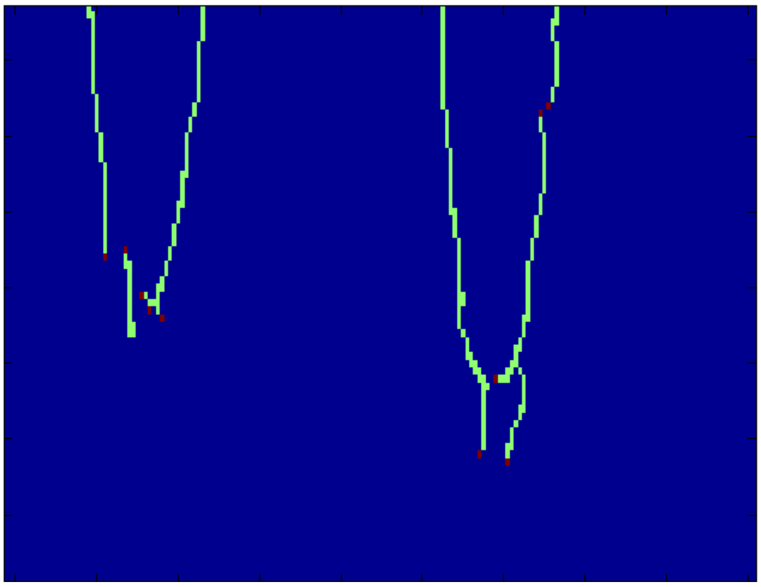


Figure 3.9: Detail of the output of the function *endpoints* displayed thanks to *viewEndpoints*. In red we can see the endpoints, in green the connected edges, and in blue the background.

that takes into account information coming from the pixels involved in the computation of the gradient.

Three parameters have to be chosen as check conditions for the prosecution of the search along the same edge branch:

- *Mthres*, that guarantees that we are looking for edge pixels in a region with a sufficiently high value of the gradient magnitude
- *greythres*, that guarantees that we are looking for edge pixels in a region with a sufficiently high value of grey level in the filtered image
- *epsgrey*, representing a tolerance on the accepted difference between the grey level of two consecutive edge pixels.

In order to keep the level of subjectivity low, instead of choosing their values empirically, we select at least the first two of them in an automatic way. Using *kmeans* set with two clusters, we find *Mthres* as the minor centroid of the gradient magnitude image, and *greythres* as the minor centroid of the filtered image. The last parameter is arbitrarily set to 5.

The list of the endpoints is scanned: the edge direction is checked at every endpoint, and accordingly to that the edge branch is expanded in the most plausible direction. Two main criteria are used: look for the candidate pixel with the highest value of gradient magnitude, or look for the candidate pixel with the lowest absolute value of difference with the grey value of the current endpoint. The first criterion is justified by the observation that along the contour the gradient magnitude is greater than in the rest of the image. The second relies on the fact that for two contiguous pixels along the contour, grey level does not change abruptly.

The structure of the code is the following. For every endpoint:

1. check the edge direction in that point
2. check if $d > 1 - d$ or not (see 3.1.1 for the definition of d)
3. check if the two adjacent pixels in the correct direction are already edge points or not in the edge image (black and white image given as output by the edge detector)
   - if both of them are not already edge points, choose the one with the greater value of gradient magnitude, or the one for which the absolute value of the difference between grey values of the endpoint and the candidate edge point is minor in the filtered image, checking if it is less than *epsgrey*. Then the inspection is restricted between the chosen point and its contiguous, with the same criteria. In fact, even if we have checked the value of d, it is not always true that the most proper candidate is the one with a minor value of d. If a point satisfies all the requests, it is marked with one in the edge image and the list of the endpoints is updated with this last added point, otherwise, we go to the next endpoint of the list and repeat the process from point 1.
   - if only one of them is not an edge point, we check directly this pixel and its contiguous, with the same criteria of the previous point.

A detail of the output of this code can be seen in figure 3.10. As we can see, most of the gaps are closed or at least reduced considerably. The main problem is that sometimes also undesired edges are extended, and not all of the gaps are closed producing a completely continuous (or connected) edge, especially the bigger gaps in the bottom of the spectrogram are not always successfully closed.

LBE method (*linklbe.m*) A second strategy is based on the concept of Likelihood for Being Edge (LBE), introduced by V. Lacroix in her work in 1987 [19].
Her idea has been adapted to the present case and exploited to extend existent edges from their endpoints. In practice, a pixel is assigned a LBE=1 if it's a "pure local directional

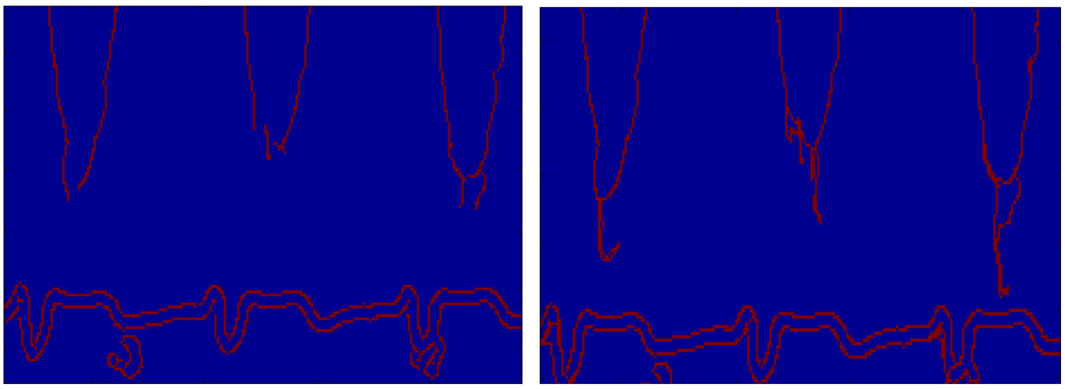

(a) (b)

Figure 3.10: Detail of the edge image **(a)** before and **(b)** after the linking step.

maximum", a LBE=0 if it is a "pure local directional minimum", and a LBE lying between 0 and 1 according to its probability of being an edge.

First of all, LBE is calculated for all the pixels. Two matrices of the same dimension as the image, m and v, are computed scanning all the pixels in the image.
Every pixel is inspected together with its two neighbours in the gradient direction (also considering the quantity d, see 3.1.1): $v(i,j)$ is incremented every time that a pixel (i,j) is visited, while $m(i,j)$ is incremented only if a pixel (i,j) has the maximum value of gradient magnitude among the three pixels examined.
The quantity LBE for every pixel is defined as the ratio between m and v. Therefore, the likelihood for being an edge is calculated as the ratio between the times that a pixel has been recognized as a local maximum of the gradient and the times that it has been visited. Figure 3.11 shows an example of LBE image.

Once we have the likelihood for every pixel, we use it to extend edges provided by the edge detector, still starting from the endpoints.
We check for possible paths both on the right and on the left of every endpoint, in the direction perpendicular to the gradient one.
The algorithm is the following:

- check the direction of the gradient

- both for the left and the right of the endpoint, look for the pixel with maximum LBE between its two neighbours in the direction perpendicular to the gradient (an example is provided by figure 3.12: if it is not already an edge point, mark it as an edge point and update the current endpoint. If both of the neighbour pixels have LBE=1, choose arbitrarily one of them as the new edge point and set the LBE of the other to 0.99, thus making it likely to be chosen as an edge point.

The result after the application of this method is shown in figure 3.13.
We can see that all the main gaps are closed or almost closed, but still some spurious branches can be found.

3.1.2 Strong pre-processing before edge detection

Instead of performing edge detection and then trying to fill the gaps by means of an edge linking process, another way of proceeding could be performing a more incisive pre-processing on the given image.

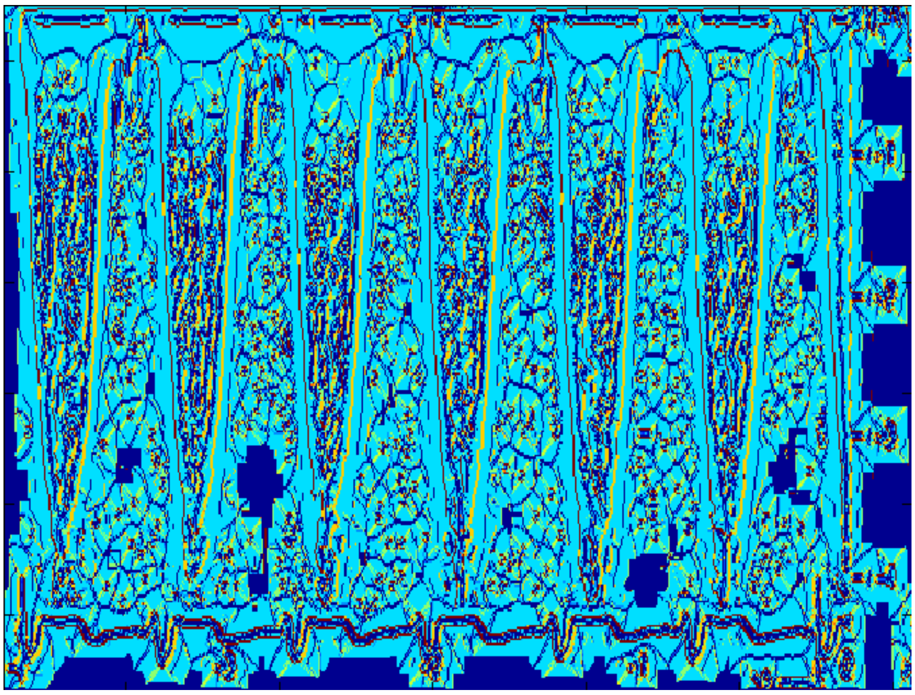


Figure 3.11: Calculated values for LBE. Red corresponds to an LBE of 1 and yellow to an LBE of 0.67. Note that the highest LBEs are concentrated on the contour of the spectrogram, as expected.

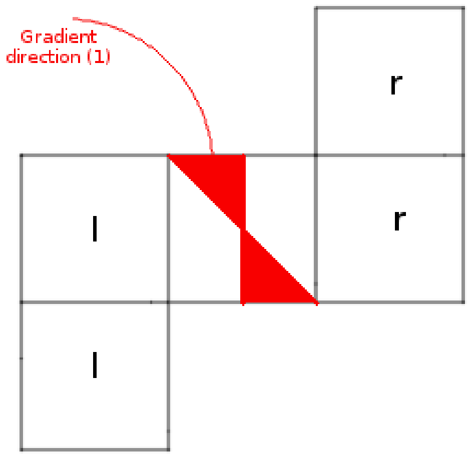


Figure 3.12: Example of the extension algorithm for direction 1. The gradient direction of the central pixel is highlighted in red, and its two right neighbours and left neighbours perpendicular to the gradient direction are marked.

The aim is to obtain a new image in which edges, even the ones that are weaker in the original image, are so pronounced to guarantee the continuity of the edges returned by the edge detector.

Two methods taken from the literature

The first thing that comes to mind is thresholding the image, but an appropriate threshold, if exists, is not so trivial to determine. Where does the useful signal end and where do disturbances and noise start? The problem is still the same.

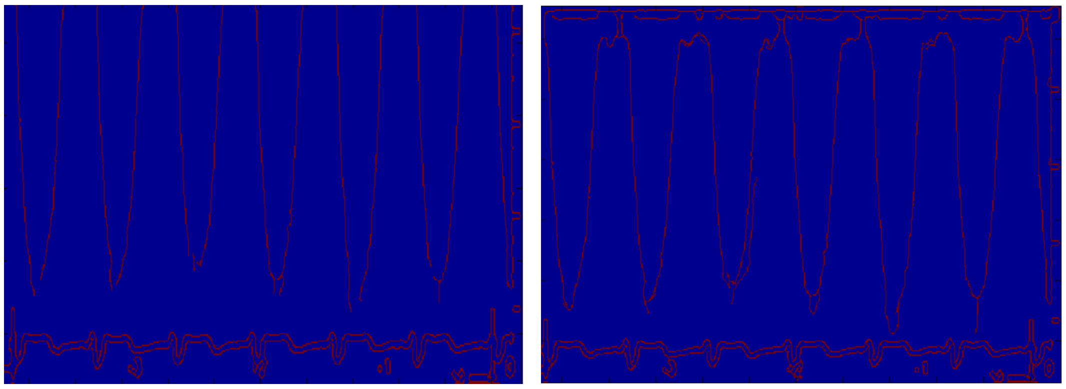

(a) (b)

Figure 3.13: Detail of the edge image **(a)** before and **(b)** after the LBE extension.

For example, **Magagnin** et al. [23] in their article divide the image in three partially overlapped regions. The intensity histogram is computed for each region, and the threshold L is chosen as the intensity at which the 25% of the pixels has an intensity greater than L. For the overlapped areas, the threshold is set to the average between the two thresholds. They don't even use an edge detector, they just look for the first pixel with intensity lower than the relative threshold in each region in each column (scanned from bottom to top), and mark it as edge. In this work we preferred to use an edge detector after the application of the thresholds. However, this method implies a choice on the division of the image, and dividing it in an effective way is difficult, also because the performance is image-dependent. Moreover, 25 % value to select the thresholds was set arbitrarily.

Anyway, testing the method with a set of images (see figure 3.14) gave quite good results, except for one of them, in which much of the useful signal is clearly missing, because of its wide range of intensities.

Kiruthika, Prabhakar and **Reddy** [18] instead, managed to obtain an image with very sharp edges only through pre-processing steps for noise removal and image enhancement. The output of the edge detector is already a connected contour that does not need to be processed further, as can be seen in figure 3.15 d.
The steps of their method are:

1. The original image (converted to greyscale if it is RGB) is filtered with a Gaussian kernel ($\sigma = 1.5$) and contrast enhanced

2. The original image undergoes a morphological closing[1] with a disk-shaped structuring element

3. Images gotten in the previous two steps are subtracted and then the result is adjusted in intensity (function *imadjust* in Matlab)

4. Canny edge detection is performed on the output of the third step.

The selection of these steps is justified by the features of Doppler images, in which edges are more pronounced in the lateral part of the peaks, and fade towards the peaks' top.

[1]Morphological closing, in image processing, is used to remove small holes. It needs a structuring element (a given shape), and it consists of a dilation operation followed by an erosion operation with the same structuring element. For binary images, dilation works like this: when the center of the structuring element is at a given point and the structuring element is overlying a non-zero point of the image, this point is set to one in the resulting image. Regarding erosion, when the center of the structuring element is at a given point, only if the entire structuring element is included in a non-zero part of the image this point is set to one in the resulting image.

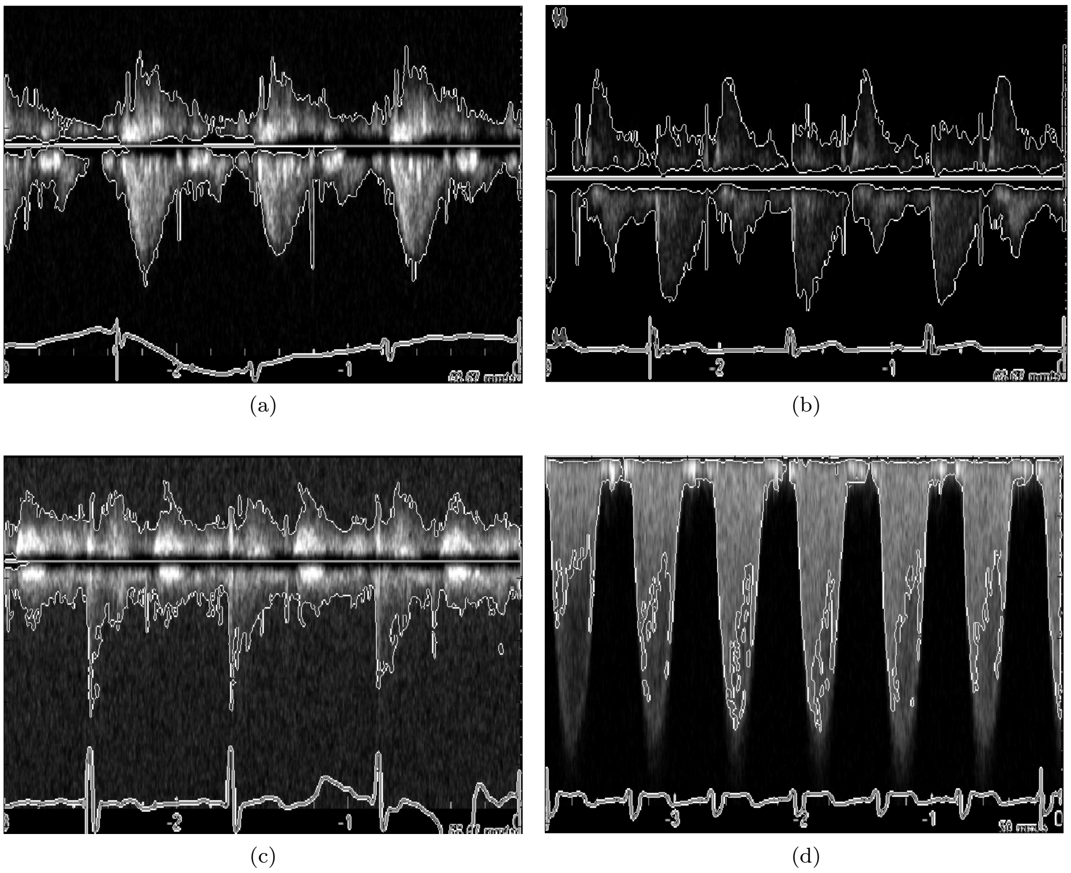


Figure 3.14: Four examples of the application of Magagnin's method. As we can see, it works pretty well even for the third noisy image, but the last one is completely unacceptable.

Region growing

We have seen that because of the morphology of Doppler images, edge based methods are not always so effective, especially where the signal fades slowly, and borders become difficult to detect or even don't exist at all. A different approach for handling the problem can be the identification of the spectra by region growing methods. The spectrum of the blood flow can be seen as a region with different properties from the background. So, starting from one (or more) so called *seed point* in the region we are interested in, we can associate to it other neighbouring pixels that are similar somehow. As an example of an aggregation criterion we can look at the grey value of candidate pixels and compare it with the mean grey value of the region already created: if the difference is below a certain threshold level, the candidate is added to the region, and clearly the mean grey value has to be updated. What we look for in a region is some kind of homogeneity.

The simple algorithm used in this work employs a threshold, checking if the grey value of the candidate pixel differs from the initial pixel grey value more than the threshold. As usual, some parameters must be set manually. In this case, we decided to take the strongest valued pixel in the region of interest as a seed point, and the threshold is in most cases chosen to be the difference between this value (255, because of the contrast stretching), and 10 times the lowest valued non-zero pixel (its value is 1 because of the contrast stretching). So in most part of images the threshold is 245.
After region growing, the contour can be extracted and it will be obviously continuous, as we can see in figure 3.16.

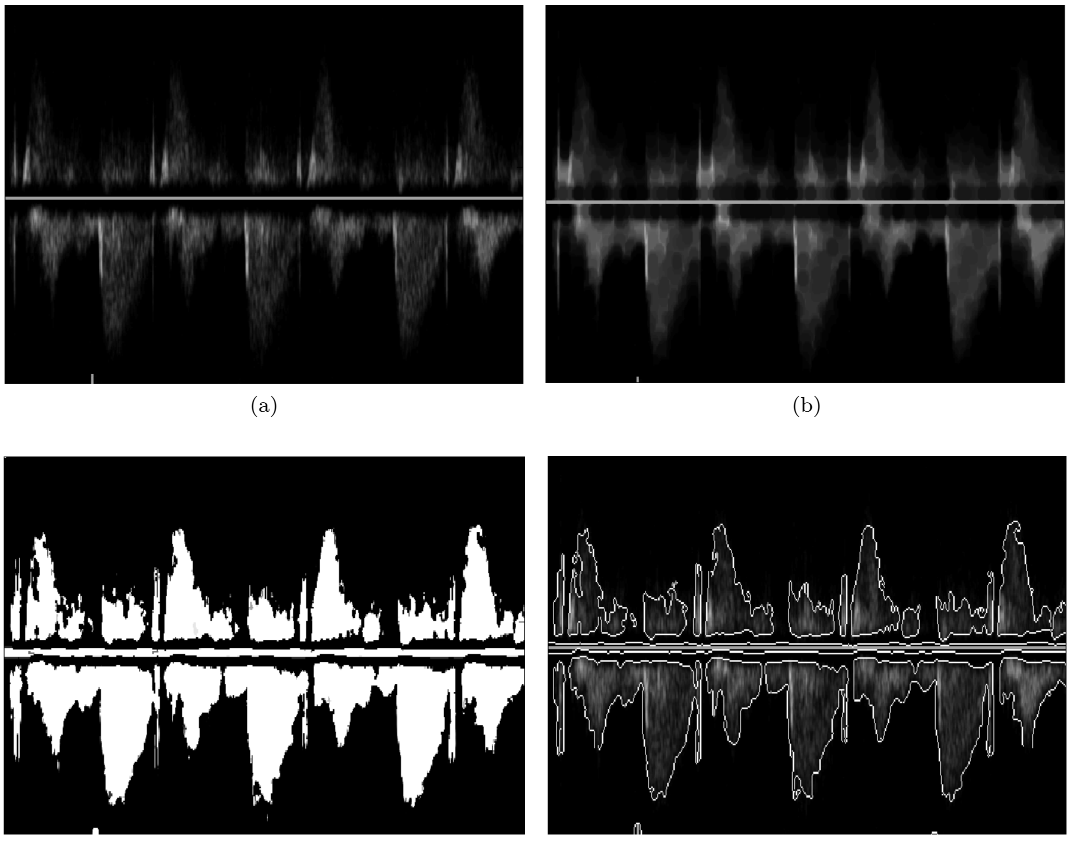


Figure 3.15: The steps of Kiruthika et al. method. **(a)** original image, **(b)** morphological closed image, **(c)** result of the subtraction between filtered and closed image, **(d)** edge detected image superimposed to the original image.

3.1.3 Level curves

Observing the structure of Doppler spectrograms, we can see that they can be compared to hills: near the baseline there is the strongest signal, coming from blood cells moving at lower velocities, and it gradually slopes in intensity towards greater velocities until it merges with the background. In this way, we can get some useful information from image's level curves, i.e. lines delimiting those regions of the image having an intensity in a certain range.
So, exploiting the information coming from these level curves we can trace a connected contour, completely without the use of an edge detector.

The point is that we have to decide how to define the proper ranges for the intensities in an effective way. In this respect, a clusterization step helped us to divide the image intensities in homogeneous groups.
The steps of the algorithm are the following:

1. As a previous cleaning step, filter the image with a median filter 3×3, perform a contrast stretching like in 3.1.1, and filter the result with a Gaussian filter with $\sigma = 1$.

2. Clusterize the processed image using the function *kmeans*. Divide the image pixels in 12 clusters according to their grey value and set the parameter *'emptyaction'* with the option *'singleton'* to prevent the interruption due to the creation of an empty cluster.

3. Use the Matlab function *contour* on the filtered image, giving as input the vector of

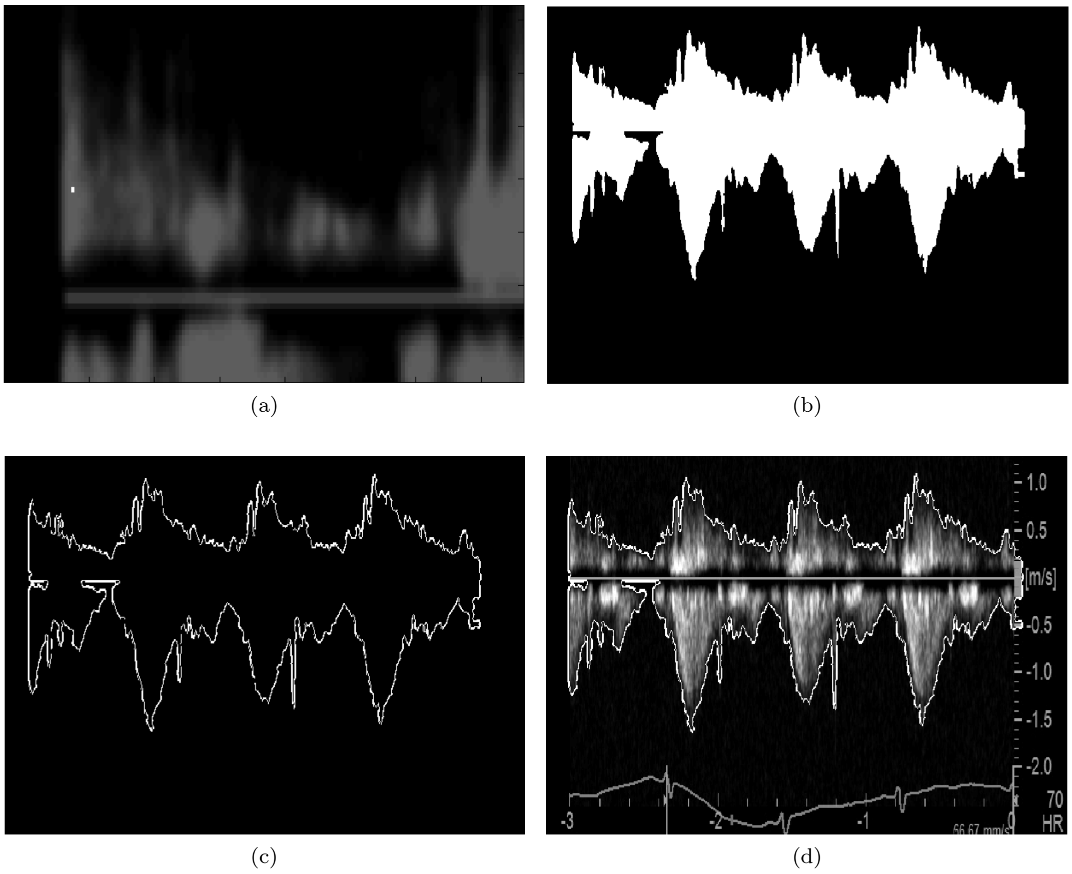


Figure 3.16: Region growing method. **(a)** the seed point, **(b)** the selected region after the application of region growing, **(c)** contour obtained with the Canny edge detector, **(d)** contour superimposed to the original image.

the ordered centroids (mean values of the clusters) created in the previous step, and saving the output C. C is a two-row matrix specifying all the contour lines. Each contour line defined in matrix C begins with a column that contains the value of the contour and the number of (x, y) vertices in the contour line. The remaining columns contain the data for the (x, y) pairs.

4. Look for the positions in the image of the level curve corresponding to the centroid chosen as a representative of the contour. (The fourth centroid, chosen in an empirical way, works well for several images, but it does not suit all of them).

5. Plot the level curve on the original image.

The results can be seen in figure 3.17. Different choices for the level curve are reported, obviously leading to different contours. Where does the velocity profile end and where does the noise start?

3.2 Further processing after velocity profile extraction

According to the literature, after performing edge detection somehow (a lot of different approaches have been given, and only few of them have been discussed in this chapter),

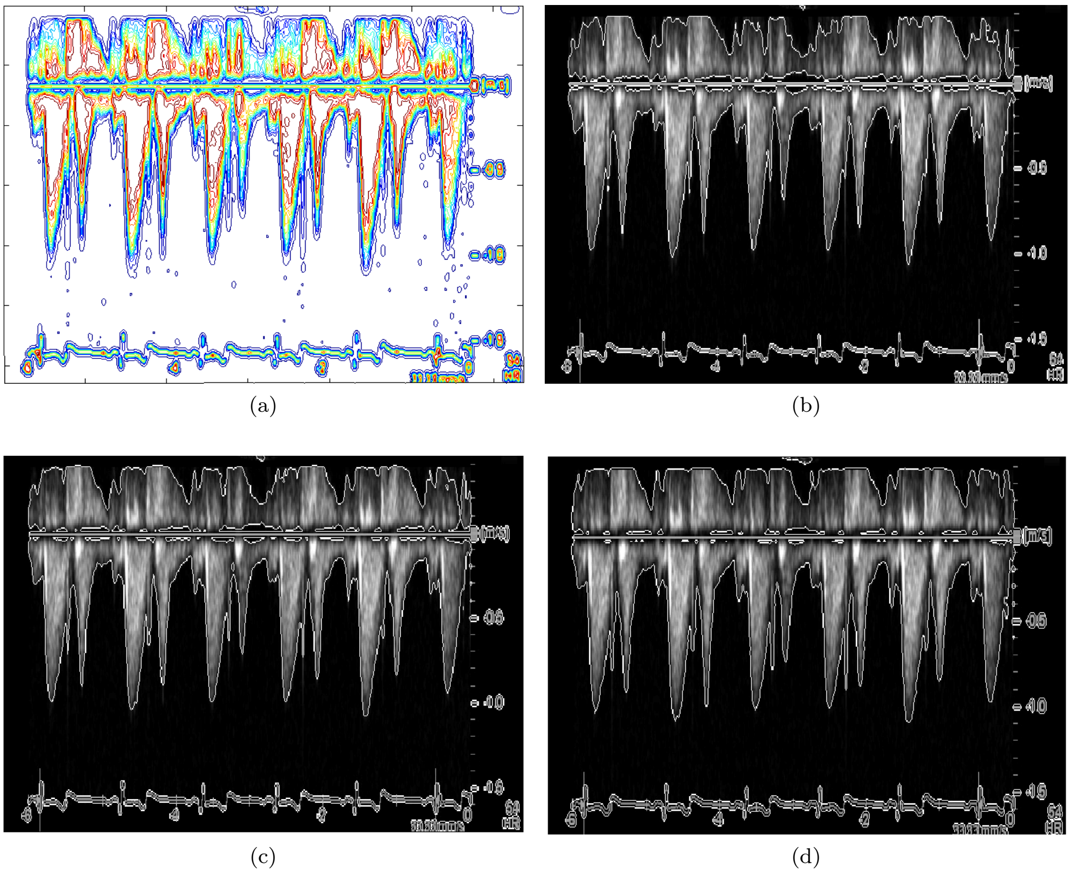


Figure 3.17: Method of the level curves. **(a)** level curves for all of the clusters, **(b)** contour obtained using the fourth centroid superimposed to the original image, **(c)** contour obtained using the third centroid superimposed to the original image, **(d)** contour obtained using the second centroid superimposed to the original image.

further processing can be done on the image. First of all, the baseline, that is the line separating positive velocities from negative ones, can be recognized automatically.

Doppler images show also some extra information besides the Doppler spectrum. For example, next to the baseline the text $[m/s]$, indicating the units for the velocity measure, can be found, as well as the text HR, that stands for heart rate, near the ECG. Moreover there are numbers on the right side of the figure indicating the velocity at a certain level of height in the image. Numbers and text can be recognized in the image, and they can help to map the information from the intensity of Doppler spectrum to the value of the velocity profile in that point.

3.2.1 Baseline identification

In order to detect with certainty the position of the baseline, we can take advantage of the Hough transform, already briefly outlined in 2.2.2. We used the Matlab built in function *hough* on the output of the Canny edge detector. This function gives as output the matrix H (the Hough transform matrix), *rho* and *theta*, which are the arrays of the values used to compute the transform. Then the function *houghpeaks* looks for the peaks in the matrix H. We set 5 as number of peaks to be found. Finally, the function *houghlines* takes as input the edge image, *theta*, *rho* and the output peaks of *houghpeaks*, and gives as output the lines.

We are interested in finding the position of the longest line, because the baseline is for sure the longest straight line in the image. So, we scan all the detected lines and check which is the longest one among them.

The result is displayed in figure 3.18. The baseline is always positioned in the right zone but sometimes a bias of one pixel occurs in the vertical dimension.

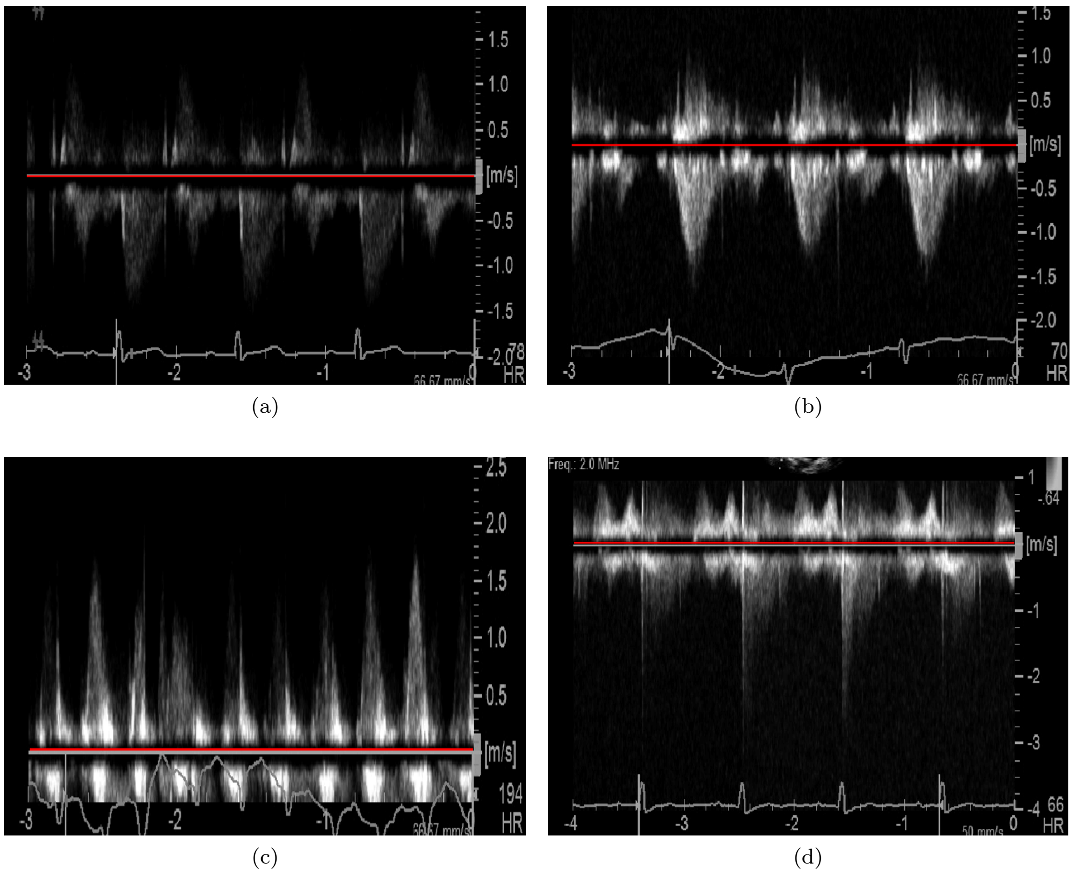


Figure 3.18: Results of the detection of the baseline on different images. The baseline is positioned correctly in **(a)** and **(b)**, while there is a bias of one pixel in **(c)** and **(d)**.

3.2.2 Numbers and text recognition

Recognizing the numbers that appear in Doppler images is quite an easy task if we exploit template matching techniques (see 2.3).

For template matching, the original image (before filtering and other processing) is employed, because little features like numbers and text are more visible and clear in it.
The interactive function *template_matching* has been created for finding the coordinates of the center of the region that best matches a template. It asks the user to type in the number or the text (among the allowed ones) he is interested in, and then let him crop the corresponding template from the image by himself. This is repeated until a signal of stop is given by the user.
For each selected template the normalized correlation coefficient is computed at each position (i, j) in the image by means of the Matlab function *normxcorr2*. In figure 3.19 we can see an example of correlation coefficient calculated for an image. Then the position of the

maximum of the correlation coefficient is found, and from it the center of the best matching region is obtained.

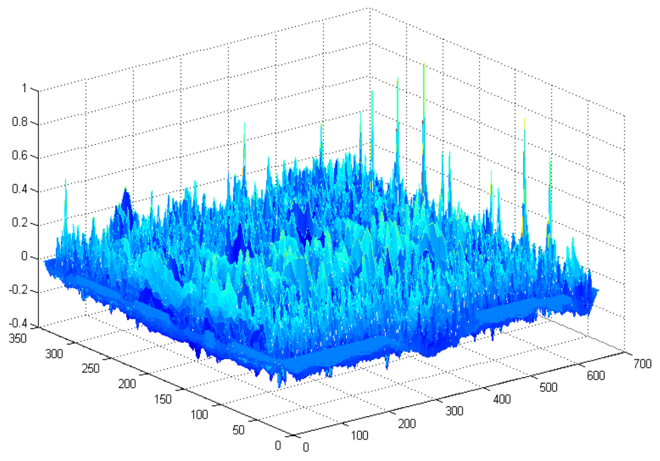


Figure 3.19: Computed correlation coefficient for a Doppler spectral image with the template representing ”-1”. The position of the maximum peak localizes the template in the image.

After finding the position of the numbers an idea is that of associating every pixel in the Doppler spectrum region with its velocity value, thus making possible for example the calculation of the mean velocity. In fact, as we know the position of the baseline, and the position of 1 or -1, it is possible to compute the sampling step of the velocity for each pixel in the vertical direction. So every pixel within the Doppler spectrum can be associated to the velocity of the blood particles it represents.

The intensity of a pixel is in relation with the quantity of blood elements moving with a certain velocity, so computing a weighted sum of each pixel’s velocity for every column gives as a result the mean velocity of the flow at each time sample.

Chapter 4

Results, discussion, and comparison

The results obtained with all the techniques we have tested in chapter 3 are compared here, trying to understand which are better and more reliable. While doing this comparison we must take into account that what we have obtained should be useful for finding the numerical values of the maximum velocity envelope, the mean flow velocity, and other parameters usually extracted to analyse this kind of data.

The so called "filtered image" is obtained from the original image after a median filtering (3×3 window, one application), a contrast stretching, and a Gaussian filtering (σ=1.5, same for all images), carried out as described in 3.1.1. All figures are displayed in *colormap(jet)* to make the eye better appreciate changes in intensity, even if usually the part of Doppler images containing velocity spectrum is displayed in grey scale mode.

4.1 Visual comparison of results

As we can see from figure 4.1, different approaches show different strengths and weaknesses.

In fig. 4.1 **(b)** the edge image after hysteresis thresholding ($T_{high} = 0.14$, $T_{low} = 0.012$, chosen with the adaptive method in 2.2.1) can be seen. It is superimposed to the filtered image, because edges have been actually extracted from that processed image, and detected contours are highlighted in red. Noticeable discontinuities can be found above all in those regions where intensity fades slowly, and there is not a neat separation between background and velocity spectrum. In addition to the external contour, also some internal branches detect abrupt changes in intensity within the spectrum.

Fig. 4.1 **(c)** and **(d)** show the edge image after two different gap linking attempts: in **(c)** edges are extended from endpoints of contour branches along the local edge direction (see 3.1.1 for details), exploiting information coming from the grey level of the filtered image and from the gradient magnitude, while in **(d)** from endpoints the so called likelihood for being an edge is taken as an expansion criterion (see 3.1.1). In both cases some gaps are successfully closed and some others are not, and of course not the same in the two cases, because of the different expansion criterion adopted. Sometimes the resulting edge is no more thin, filling more than one pixel in width.

In fig. 4.1 **(e)** and **(f)** the edge images resulting from the application of Magagnin [23] and Kiruthika [18] are shown respectively, superimposed to the original image. In these two cases, in fact, extracted contours are obtained without the pre-processing steps that lead to the filtered image visible in **(b)**-**(d)**. In **(e)** some minor discontinuities can be still found, while in **(f)** almost all contours are continuous. The second method seems to be more conservative, keeping most of the spectrum inside the bevelled contour, while the first leaves the weaker part of the signal out of the sharper contour.

Method	**Pros**	**Cons**
Gap linking 1	Mathematically based (edge detection); low subjectivity; almost closed contour.	Some persisting gaps; some spurious branches; loss of thinness; dependence of the results on the chosen thresholds.
LBE	Mathematically based (edge detection); low subjectivity; almost closed contour.	Some persisting gaps; some spurious branches; loss of thinness; dependence of the results on the chosen thresholds; time consuming.
Magagnin et al.	Closed contour; adaptive threshold-based method.	Performance depending on the image; tendency to cut part of probable useful signal.
Kiruthika et al.	Closed contour; quite conservative.	Heavy pre-processing on the image.
Level curves	Closed contour; no need of an edge detector.	Subjectivity in the choice of the level curve representing the contour; great dependence of the contour on the chosen level.
Region growing	Closed contour.	Too conservative; great dependence of the correct region selection on the chosen threshold.

Table 4.1: Comparison of all the methods employed. Magagnin, Kiruthika, and region growing are intended together with edge detection.

Fig. 4.1 **(g)** shows the contour resulting from the application of the level curves method (3.1.3), superimposed to the filtered image. The fourth centroid of the 12 clusters in which pixels of the filtered image have been divided has been chosen as a limit. The contour is closed and plausible, but maybe a different choice of the level curve could be better, having some criteria for deciding where the useful signal stops.

Finally, in fig. 4.1 **(h)** the contour of region growing strategy (3.1.2) is highlighted, superimposed to the filtered image. The homogeneous region representing the spectrum is grown from one of the brightest pixels in the central part of the image. In this case the chosen threshold is 240 and in fact we can see that some less bright pixels are left out of the spectrum.

4.2 Discussion

Making a visual comparison of the different techniques employed for the velocity profile extraction, it is clear that some approaches are somehow complementary to others. As table 4.1 summarizes, edge detection oriented methods rest on a solid mathematical basis, but the counterpart is that the obtained profile is not completely continuous, and sometimes the gap linking step creates some spurious and undesired branches. The last are created because of ”wrong” decisions taken at a certain point of the extension of an edge branch. In fact, the choice of the prosecution pixel done by the gap linking methods (both of them) is not always appropriate, leading to undesired ramifications as we can see from figure 4.2. We can also notice that the different methods close in a different way the gap left by the edge detector, and that the spurious branch on the left (first gap linking method) goes down, while the one on the right (LBE method) goes up. So these two methods do not behave in the same way.

Image	Gap linking 1	LBE
test image 1	57	102
test image 2	32	69
test image 3	69	136
test image 4	57	126
test image 5	77	134

Table 4.2: Number of endpoints in five test images after gap linking with two different methods.

Looking at figure 4.2, at a first glance it seems like the gap linking method that exploits gradient magnitude and grey value of the pixels closes more edges than the other one. Actually, checking how many endpoints are still present in the image after gap linking, it comes out that they are more after the application of the LBE method, in all tested images. Table 4.2 shows this phenomenon for five test images. As we can see, endpoints after LBE gap linking are about double of endpoints after the other gap linking method. This means that the second method is more effective in filling the gaps in discontinuous contours.

With regard to Magagnin method, it can be observed that its performances change deeply according to the image we are considering.
The shape of the velocity profile is different according to the vessel or the place of the vessel in which the exam was performed. In particular, for shapes analogous to that in figure 4.2, Magagnin method does not work very well, leaving out of the traced contour most part of the profile, as we can see in figure 4.3. This is highly undesirable of course. The reason for this might be sought in the fact that Magagnin method is threshold-based (even if more refined than a trivial unique threshold). For images in which a large range of intensities appears, the chosen threshold may be too low, and this is what actually happens with this kind of images. Kiruthika method, on the contrary, is not affected by these problems and it performs quite well even with elongated shapes. For other shapes, where the velocity range is more limited, both methods work in a satisfactory way, even if they give different results (figure 4.4).

Level curves and region growing methods do not show this kind of weaknesses and their performance is quite independent of the shape, but it deeply depends on the selection of the level curve for the first method, and of the threshold for the second one.

4.3 Possible improvements

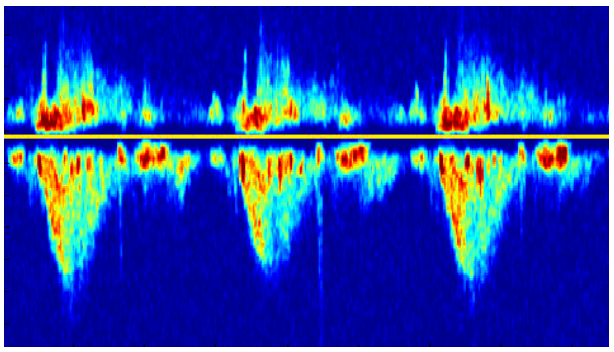
Considering weaknesses and strengths of the proposed methods emerged from the previous comparison, we can say that there is not an optimal strategy to trace the ”true” velocity profile.
However, considering that we would like to obtain a contour that is continuous as much as possible but preferably supported by edge detection at the same time, we can think about integrating two methods, thus creating a hybrid approach to the problem.

The method that exploits gradient magnitude and grey level of pixels for extending branches of open contours after edge detection gave good results, surely better than LBE method. As we can see in table 4.1, combining pros of this gap linking method with level curves method, for example, could give rise to an interesting compromise.

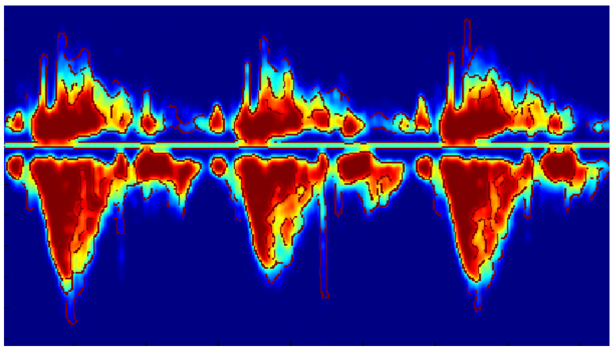
For example, starting from an endpoint of a discontinuous branch in the edge image, the edge can be extended, still following its local direction, finding the level curve that passes through that point. This level curve should be traced until it reaches another endpoint or an edge point.

Another possible strategy of integration could be the following. Still starting from an endpoint of a discontinuous branch and considering the local direction of the edge, edges could be extended with a more efficient method than those already tested (in a way that it does not stop until it does not reach another endpoint or edge point), with an additional constraint of being between two predetermined level curves. In this way, we guarantee that the edge does not exceed a zone of homogeneity in the image.

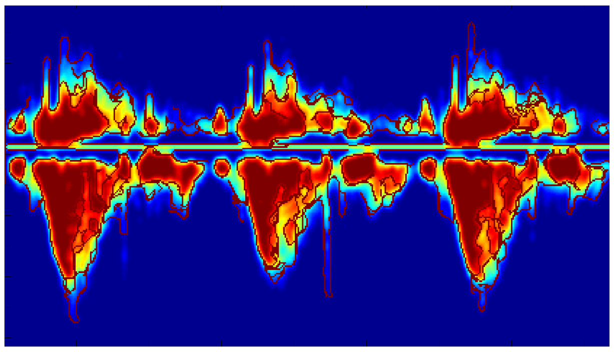
Obviously, these proposals should be tested in order to check the final result, but they seem ideally promising, opening the way to further investigation.



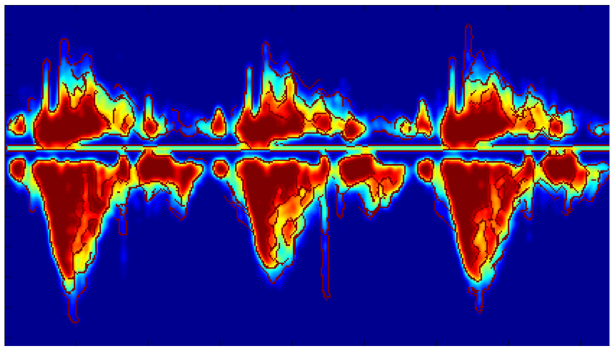
(a)



(b)

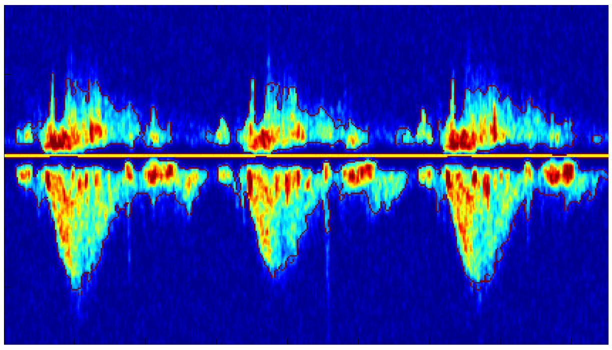


(c)

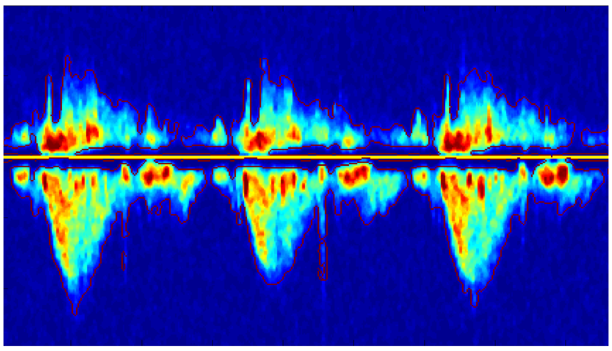


(d)

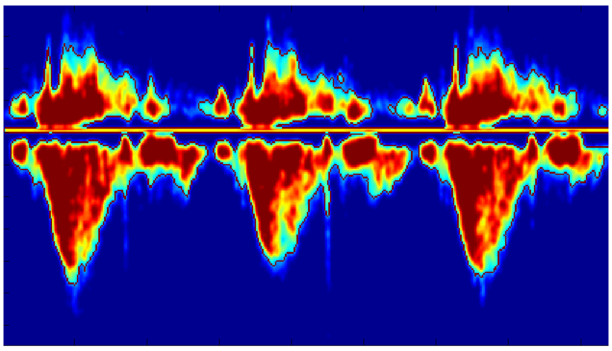
Figure 4.1: Comparison of different employed techniques. **(a)** original image, **(b)** edge image after Canny edge detector superimposed to the filtered image, **(c)** edge image after edge linking superimposed to the filtered image, **(d)** edge image after LBE method superimposed to the filtered image.



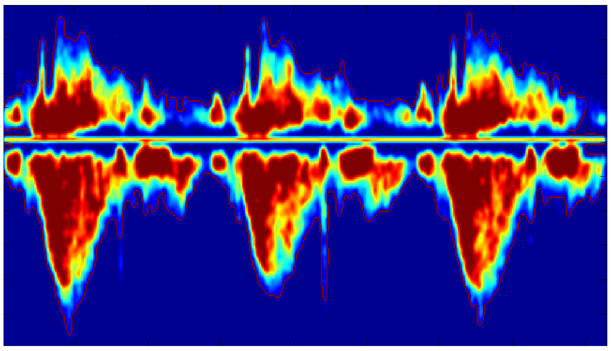
(e)



(f)



(g)



(h)

Figure 4.1: Comparison of different employed techniques. **(e)** edge image after Magagnin method superimposed to the original image, **(f)** edge image after Kiruthika method superimposed to the original image, **(g)** edge image after level curves method superimposed to the filtered image, **(h)** edge image after region growing method superimposed to the filtered image.

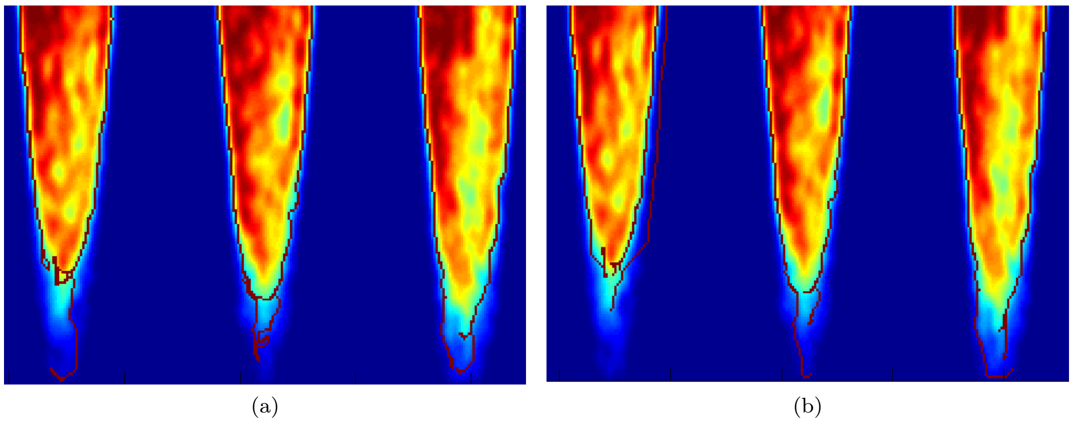


Figure 4.2: Comparison of the method that exploits gradient magnitude and grey level of pixels (**a**) and LBE (**b**) method for a particular image.

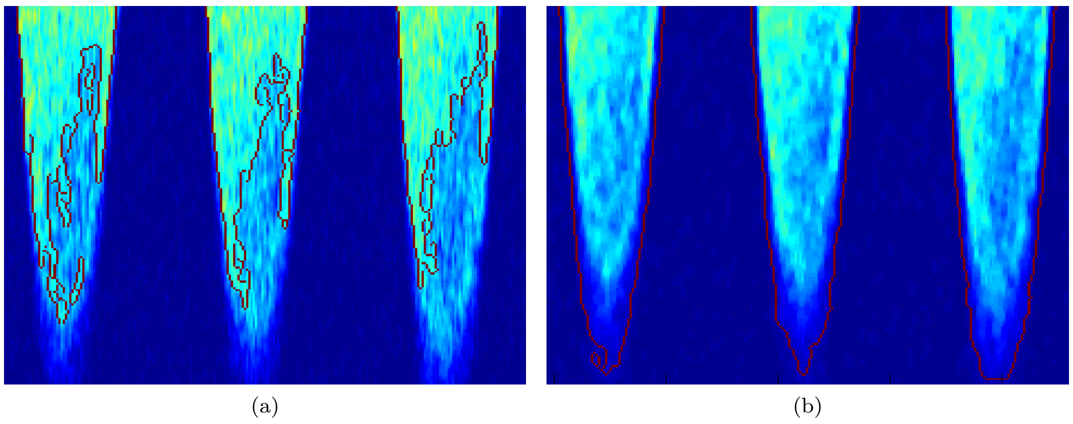


Figure 4.3: Comparison of Magagnin (**a**) and Kiruthika (**b**) methods for a particular image.

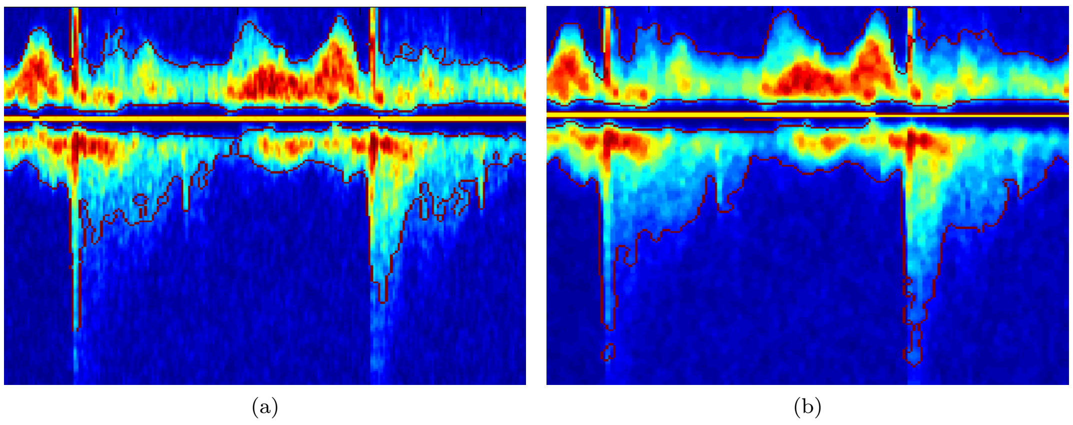


Figure 4.4: Comparison of Magagnin (**a**) and Kiruthika (**b**) methods for a particular image.

Conclusions

In the present work, different techniques for the automatic extraction of the velocity profile from Doppler spectral images have been tested and compared. Most of the tested methods can be divided in two main groups, i. e. approaches based on edge detection and gap linking, and approaches that are more focused on the pre-processing phase, resorting to edge detection only at a later time. Level curves method lies outside these two groups, as it is totally independent from edge detection. In this way, it can represent an alternative, or, more realistically, an interesting supplement to integrate with edge detection. In fact, strengths and weaknesses of edge detection are complementary to those of level curves. The first is based on a mathematical definition of edge, meaning a discontinuity in the grey level of the image, thus being reliable and little subjected to arbitrary decisions. The only elements of subjectivity are the choice of the standard deviation of the Gaussian filter and of the thresholds adopted for the selection of meaningful edges (we have seen that this choice can be somehow automatized). However, the main weakness of edge detection is the fragmented result almost always obtained: edges are far from being continuous and it is not so trivial to fill the gaps in a proper way. On the other hand, level curves guarantee closed contours, but the price is the manual choice of the limit level we assume to convey useful information. Combining these two approaches can be interesting and promising in further work.

Certainly, edge extraction is still a challenging task not only regarding Doppler images. In this work we have only seen a few techniques, but further investigation with other approaches is possible.

Appendix A

Matlab code

```
function imstretch=contraststretch(im)
% This function returns the contrast-stretched version of the input
image

[nr,nc]=size(im);

%Clusterization step (2 clusters)
[x,c]=kmeans(im(:),2,'emptyaction','singleton');

c=sort(c);

%Output levels
y1=0;
y2=255;

imstretch=zeros(nr,nc);

%Contrast stretching
for i=1:nr
    for j=1:nc

        if im(i,j)>=0 && im(i,j)<=c(1)
            imstretch(i,j)=y1;

        elseif im(i,j)>c(1) && im(i,j)<=c(2)
            imstretch(i,j)=(y2-y1)/(c(2)-c(1))...
                           *(im(i,j)-c(1))+y1;
        else
            imstretch(i,j)=y2;
        end
    end
end
```

```
function filt=gaussfilt(im, sigma)
% Function performing gaussian filtering with a standard deviation
sigma
% on the image im

% Vector containing 97% of the area under the Gaussian curve
x = [-(5*sigma):(5*sigma)];
```

```

% Creation of the Gaussian curve on the points specified by the vector
G = normpdf(x,0,sigma);

% Convolution along the rows
filtx=conv2(im,G,'same');

% Second convolution along the columns
filt=conv2(filtx,G','same');
```

```
function [bw,M,dir,gx,gy,d,M1,th,tl]=imcanny(Img,sigma,method)
%Function that extracts the edges in an input image with a
%Canny algorithm.

%--- Input ---
%Img = input image
%sigma = standard deviation of the Gaussian filter
%method = method employed for the choice of the double threshold
%('gonz'= Gonzalez-Woods method, 'med' = Medina et al. method,
%'0'= fixed thresholds)

%--- Output ---
%bw = edge image
%M = gradient magnitude
%dir = gradient direction
%gx,gy = components of the gradient
%d = gradient slope
%M1 = gradient magnitude after non maxima suppression
%th,tl= hysteresis thresholds


[nr,nc]=size(Img);

%% Creation of the Gaussian
% Vector with 97% of the area under the Gaussian curve
x = [-(5*sigma):(5*sigma)];
% Gaussian monodimensional distribution
G = normpdf(x,0,sigma);
% Derivative of Gaussian
dG  = diff(G);


%% Separability
% Convolution of the image with dG (along the rows)
gx = conv2(Img,dG,'same');
% Convolution of the image with dG (along the columns)
gy = conv2(Img',dG,'same')';

%% Gradient magnitude and direction
% Gradient magnitude
M =sqrt(gx.^2 + gy.^2);
% Gradient magnitude normalized
M1=M/(max(max(M)));
% Vectorization and normalization
M_=(M(:)/max(M(:)));

%% Choice of the double threshold
```

```

% Gonzalez method
if strcmp(method,'gonz')
n_pixels=length(M_);

ct=0;
for i=1:n_pixels
    if M_(i)==0
        ct=ct+1; %number of zero pixels
    end
end

p=0.3;
n_edgep=p*(n_pixels-ct); %number of edge points

x=[0:0.004:1];
%Histogram of the gradient magnitudes on the bins in vector x
[n,c]=hist(M_,x);
%Number of non edge pixels
n_nonedge=(1-p)*(n_pixels-n(1));

r=1;
s=0;
while s<n_nonedge && r<length(n)
   s=s+ n(r+1);
   r=r+1;
end

th=c(r); %High threshold
tl=0.4*th; %Low threshold

% Medina method (see Medina et al.
% "On Candidates selection for hysteresis thresholds
% in edge detection", "A novel method to look for the
% hysteresis thresholds for the Canny edge detector")
elseif strcmp(method,'med')

L=100; %number of bins
[p,q]=hist(M1(:),L); %Histogram of gradient magnitudes
p=p/sum(p);
x=[1:1:L-1];


for x=2:1:L
    sumh0=0;
 for r=1:x
    sumh0=sumh0+p(r);
 end
H0(x-1)=sumh0;
sumh1=0;
 for r=(x+1):L
    sumh1=sumh1+p(r);
 end
H1(x-1)=sumh1;
end


for x=2:L
```

```
    for y=2:L
        if x<y
            TP(y-1,x-1)=H1(y-1);
            TN(y-1,x-1)=H0(x-1);
            FN(y-1,x-1)=H0(y-1)-H0(x-1);
            FP(y-1,x-1)=0;
        elseif x>y
            TP(y-1,x-1)=H1(x-1);
            TN(y-1,x-1)=H0(y-1);
            FP(y-1,x-1)=H1(y-1)-H1(x-1);
            FN(y-1,x-1)=0;
        end
    end

end

for x=2:L
    sumtp=0;
sumtn=0;
sumfp=0;
sumfn=0;
    for y=2:L
        sumtp=sumtp+TP(y-1,x-1);
        sumtn=sumtn+TN(y-1,x-1);
        sumfp=sumfp+FP(y-1,x-1);
        sumfn=sumfn+FN(y-1,x-1);
    end
    TP_(x-1)=1/L*sumtp;
    TN_(x-1)=1/L*sumtn;
    FP_(x-1)=1/L*sumfp;
    FN_(x-1)=1/L*sumfn;
end

rho=TP_(x-1)+FP_(x-1);
TL=(L-2)*rho/(1+2*rho);
perc=1-rho;
x=1;
s=0;
while s<perc
    s=s+p(x);
    x=x+1;
end

TH=x;
for x=round(TL):TH
SN_(x)=TP_(x)/(TP_(x)+FN_(x));
SP_(x)=TN_(x)/(FP_(x)+TN_(x));
end
SP2=diff(SP_,2);
SN2=diff(SN_,2);
[maxsp2,imaxsp2]=max(SP2);
[minsp2,iminsp2]=min(SP2);
[maxsn2,imaxsn2]=max(SN2);
[minsn2,iminsn2]=min(SN2);


seth=[TL:1:TH]/255; %set of possible high thresholds
setl=0.4*seth; %set of possible low thresholds
```

```
setl2=0.5*seth;
setl3=0.3*seth;

%Set contains possible couples of thresholds
set(1,:)=[seth,seth,seth];
set(2,:)=[setl,setl2,setl3];

deltaG=zeros(nr,nc,length(set));

bw=zeros(nr,nc);
for i=1:length(set)
    th=set(1,i);
    tl=set(2,i);
    aboveT1=M1>tl;
    [aboveT2r, aboveT2c] = find(M1 >th);

    bw = bwselect(aboveT1, aboveT2c, aboveT2r, 8);
    G_high=M1>th;
    deltaG(:,:,i)=bw-G_high; %eqn 1 paper
end

SM=sum(deltaG,3);
ProbSM=SM/length(set);
setx=[0.001:0.001:0.999]; %possible values for x
pos=1;

for i=1:length(setx)
    x=setx(i);
ProbxSM=ProbSM>x;
numProbxSM=sum(sum(ProbxSM));
Fx=chop(M1.*ProbxSM,2); %rounding at two significative digits

ind=find(Fx==x); %indices at which Fx=x
[k,l]=ind2sub([nr,nc],ind);
numFx=sum(Fx(k,l));
if numProbxSM>0
    PFx=numFx/numProbxSM;
    if ne(PFx,0)
    D(pos)=x;
    pos=pos+1;
    end
else
    PFx=0;
end
end

[m,n]=hist(ProbSM(:),D);
bar(n,m)

tl=n(1); %low threshold
th=n(end); %high threshold

%Fixed threshold
elseif method==0
th=0.28;
tl=0.4*th;

end
```

```

%% Searching the local maxima
%
%        1          2
%    X---------X---------X
%    |                   |
% 4  |     -gy  ^        |  3
%    |          |        |
%    |          |    gx  |
%    X          P----->  X
%    |                   |
% 3  |                   |  4
%    |                   |
%    |                   |
%    X---------X---------X
%        2          1


M2=M1>0;

%% Computation of the gradient direction according to
%% the directions in the figure
dir=zeros(nr,nc);
for i=1:nr
    for j=1:nc
        dir(i,j)=direction(gx(i,j),gy(i,j));
    end
end

%% Non maxima suppression
d=zeros(nr,nc);
for i=2:nr-1
    for j=2:nc-1

            switch dir(i,j)
             case 1
                 d(i,j)=abs(gx(i,j)/gy(i,j)); %cotangent
                 if M2(i,j)
                 M2(i,j)=M1(i,j)>(1-d(i,j))*...
                     M1(i+1,j)+d(i,j)*M1(i+1,j+1) &...
                     M1(i,j)>(1-d(i,j))*M1(i-1,j)+d(i,j)*M1(i-1,j-1);
                 end
             case 2
                 d(i,j)=abs(gx(i,j)/gy(i,j)); %cotangent
                 if M2(i,j)
                 M2(i,j)=M1(i,j)>(1-d(i,j))*...
                     M1(i-1,j)+d(i,j)*M1(i-1,j+1) &...
                     M1(i,j)>(1-d(i,j))*M1(i+1,j)+d(i,j)*M1(i+1,j-1);
                 end
             case 3
                 d(i,j)=abs(gy(i,j)/gx(i,j)); %tangent
                 if M2(i,j)
                 M2(i,j)=M1(i,j)>(1-d(i,j))*...
                     M1(i,j+1)+d(i,j)*M1(i-1,j+1) &...
                     M1(i,j)>(1-d(i,j))*M1(i,j-1)+d(i,j)*M1(i+1,j-1);
                 end
             case 4
                 d(i,j)=abs(gy(i,j)/gx(i,j)); %tangent
```

```
                    if M2(i,j)
                    M2(i,j)=M1(i,j)>(1-d(i,j))*...
                         M1(i,j+1)+d(i,j)*M1(i+1,j+1) &...
                         M1(i,j)>(1-d(i,j))*M1(i,j-1)+d(i,j)*M1(i-1,j-1);
                    end
               end

    end
end


%M2 is a mask containing the local maxima of the gradient
M1=M1.*M2;

%% Hysteresis thresholding
aboveT1=M1>tl;
[aboveT2r, aboveT2c] = find(M1 >th);
bw = bwselect(aboveT1, aboveT2c, aboveT2r, 8); %obtained edge
```

```
function dir = direction(gx,gy)
%Function that calculates the direction of the gradient
%(subfunction of imcanny)
%gx,gy=components of the gradient

dir=0;

if abs(gy)>abs(gx)
     if(gx*gy)>0
          dir=1;
     elseif (gx*gy<0)
          dir=2;
     end
else
     if(gx*gy)>0
          dir=4;
     elseif (gx*gy)<0
          dir=3;
     end
end
```

```
function [bw2,endp2,i]=rep(bw2,r,c,dir,i,d,M,filt,Mthres,greythres,
epsgrey,endp2)
%subfunction of edgelinking

if dir(r,c)==1
     left=[r+1,c+1];
     right=[r-1,c-1];
     up=[r-1,c];
     down=[r+1,c];
elseif dir(r,c)==2
     left=[r+1,c-1];
     right=[r-1,c+1];
     up=[r-1,c];
     down=[r+1,c];
elseif dir(r,c)==3
```

```
     left=[r+1,c-1];
     right=[r-1,c+1];
     up=[r,c+1];
     down=[r,c-1];
else
     left=[r-1,c-1];
     right=[r+1,c+1];
     up=[r,c+1];
     down=[r,c-1];
end

if d(r,c)>1-d(r,c)
 if bw2(left(1),left(2))==0 && bw2(right(1),right(2))==0
  if (M(left(1),left(2))>=M(right(1),right(2))...
     && ne(M(left(1),left(2)),0))  ||...
      (abs(filt(r,c)-filt(left(1),left(2)))<=...
      abs(filt(r,c)-filt(right(1),right(2)))...
     && abs(filt(r,c)-filt(left(1),left(2)))<=epsgrey)
   if ((M(down(1),down(2))>=M(left(1),left(2))...
       && ne(M(down(1),down(2)),0) && (M(down(1),down(2))>=Mthres))...
        ||  (abs(filt(r,c)-filt(down(1),down(2)))<=...
        abs(filt(r,c)-filt(left(1),left(2)))...
       && filt(down(1),down(2))>=greythres)) && bw2(down(1),down(2))==0
    bw2(down(1),down(2))=1;
    endp2(i,:)=[down(1),down(2)];
   elseif (M(left(1),left(2))>=M(down(1),down(2))  &&...
            ne(M(left(1),left(2)),0)&&(M(left(1),left(2))>=Mthres))  ||...
            (abs(filt(r,c)-filt(down(1),down(2)))>...
            abs(filt(r,c)-filt(left(1),left(2)))...
           && filt(left(1),left(2))>=greythres)
    bw2(left(1),left(2))=1;
    endp2(i,:)=[left(1),left(2)];
   else i=i+1;
   end
  elseif (M(right(1),right(2))>=M(left(1),left(2))  &&...
            ne(M(right(1),right(2)),0))  ||...
            (abs(filt(r,c)-filt(left(1),left(2)))>...
            abs(filt(r,c)-filt(right(1),right(2)))...
           && abs(filt(r,c)-filt(right(1),right(2)))<=epsgrey)
   if ((M(up(1),up(2))>=M(right(1),right(2))...
       && ne(M(up(1),up(2)),0) && (M(up(1),up(2))>=Mthres))  ||...
        (abs(filt(r,c)-filt(up(1),up(2)))<=...
        abs(filt(r,c)-filt(right(1),right(2)))...
       && filt(up(1),up(2))>=greythres)) && bw2(up(1),up(2))==0
    bw2(up(1),up(2))=1;
    endp2(i,:)=[up(1),up(2)];
   elseif (M(right(1),right(2))>=M(up(1),up(2))  &&...
          ne(M(right(1),right(2)),0)&&(M(right(1),right(2))>=Mthres))...
           ||  (abs(filt(r,c)-filt(up(1),up(2)))>...
           abs(filt(r,c)-filt(right(1),right(2)))...
           && filt(right(1),right(2))>=greythres)
    bw2(right(1),right(2))=1;
    endp2(i,:)=[right(1),right(2)];
   else i=i+1;
   end
  else i=i+1;
  end
 elseif  bw2(right(1),right(2))==0
```

```
  if ((M(up(1),up(2))>=M(right(1),right(2)) &&...
      ne(M(up(1),up(2)),0) && (M(up(1),up(2))>=Mthres)) ||...
      (abs(filt(r,c)-filt(up(1),up(2)))<=...
      abs(filt(r,c)-filt(right(1),right(2)))...
      && abs(filt(r,c)-filt(up(1),up(2)))<=epsgrey  &&...
      filt(up(1),up(2))>=greythres)) && bw2(up(1),up(2))==0
   bw2(up(1),up(2))=1;
   endp2(i,:)=[up(1),up(2)];
  elseif (M(right(1),right(2))>=M(up(1),up(2)) &&...
      ne(M(right(1),right(2)),0)&&(M(right(1),right(2))>=Mthres))...
          || (abs(filt(r,c)-filt(up(1),up(2)))>...
          abs(filt(r,c)-filt(right(1),right(2)))...
          && abs(filt(r,c)-filt(right(1),right(2)))<=epsgrey...
          && filt(right(1),right(2))>=greythres)
   bw2(right(1),right(2))=1;
   endp2(i,:)=[right(1),right(2)];
  else i=i+1;
  end
 elseif  bw2(left(1),left(2))==0
  if ((M(down(1),down(2))>=M(left(1),left(2)) &&...
      ne(M(down(1),down(2)),0)&&(M(down(1),down(2))>=Mthres)) ||...
     (abs(filt(r,c)-filt(down(1),down(2)))<=...
     abs(filt(r,c)-filt(left(1),left(2)))...
     && abs(filt(r,c)-filt(down(1),down(2)))<=epsgrey  &&...
     filt(down(1),down(2))>=greythres))&& bw2(down(1),down(2))==0
   bw2(down(1),down(2))=1;
   endp2(i,:)=[down(1),down(2)];
  elseif (M(left(1),left(2))>=M(down(1),down(2)) &&...
      ne(M(left(1),left(2)),0)&&(M(left(1),left(2))>=Mthres)) ||...
        (abs(filt(r,c)-filt(down(1),down(2)))>...
        abs(filt(r,c)-filt(left(1),left(2)))...
        && abs(filt(r,c)-filt(left(1),left(2)))<=...
        epsgrey && filt(left(1),left(2))>=greythres)
   bw2(left(1),left(2))=1;
   endp2(i,:)=[left(1),left(2)];
  else i=i+1;
  end
 else i=i+1;
 end
elseif d(r,c)<=1-d(r,c)
 if bw2(down(1),down(2))==0 && bw2(up(1),up(2))==0
  if (M(down(1),down(2))>=M(up(1),up(2)) &&...
     ne(M(down(1),down(2)),0)) ||...
     (abs(filt(r,c)-filt(down(1),down(2)))<=...
     abs(filt(r,c)-filt(up(1),up(2)))...
     && abs(filt(r,c)-filt(down(1),down(2)))<=epsgrey)
   if ((M(left(1),left(2))>=M(down(1),down(2)) &&...
     ne(M(left(1),left(2)),0)&&(M(left(1),left(2))>=Mthres))||...
       (abs(filt(r,c)-filt(left(1),left(2)))<=...
       abs(filt(r,c)-filt(down(1),down(2)))...
     && filt(left(1),left(2))>=greythres))&& bw2(left(1),left(2))==0
    bw2(left(1),left(2))=1;
    endp2(i,:)=[left(1),left(2)];
   elseif (M(down(1),down(2))>=M(left(1),left(2)) &&...
       ne(M(down(1),down(2)),0)&&(M(down(1),down(2))>=Mthres))||...
          (abs(filt(r,c)-filt(left(1),left(2)))>...
          abs(filt(r,c)-filt(down(1),down(2)))...
          && filt(down(1),down(2))>=greythres)
      bw2(down(1),down(2))=1;
      endp2(i,:)=[down(1),down(2)];
     else  i=i+1;
     end
   elseif  (M(up(1),up(2))>=M(down(1),down(2))  &&...
            ne(M(up(1),up(2)),0))  ||...
        (abs(filt(r,c)-filt(down(1),down(2))) >...
        abs(filt(r,c)-filt(up(1),up(2)))...
          && abs(filt(r,c)-filt(up(1),up(2)))<=epsgrey)
     if  ((M(right(1),right(2))>=M(up(1),up(2))  &&...
      ne(M(right(1),right(2)),0)&&(M(right(1),right(2))>=Mthres))||...
       (abs(filt(r,c)-filt(right(1),right(2)))<=...
       abs(filt(r,c)-filt(up(1),up(2)))...
      &&  filt(up(1),up(2)-1)>=greythres))&& bw2(up(1),up(2)-1)==0
      bw2(right(1),right(2))=1;
      endp2(i,:)=[right(1),right(2)];
     elseif   (M(up(1),up(2))>=M(right(1),right(2))  &&...
              ne(M(up(1),up(2)),0)  &&  (M(up(1),up(2))>=Mthres))  ||...
             (abs(filt(r,c)-filt(right(1),right(2))) >...
             abs(filt(r,c)-filt(up(1),up(2)))...
            &&  filt(up(1),up(2))>=greythres)
      bw2(up(1),up(2))=1;
      endp2(i,:)=[up(1),up(2)];
     else  i=i+1;
     end
   else  i=i+1;
   end
  elseif   bw2(up(1),up(2))==0
   if  ((M(right(1),right(2))>=M(up(1),up(2))&&...
      ne(M(right(1),right(2)),0)&&(M(right(1),right(2))>=Mthres))  ||...
       (abs(filt(r,c)-filt(right(1),right(2)))<=...
       abs(filt(r,c)-filt(up(1),up(2)))...
      && abs(filt(r,c)-filt(right(1),right(2)))<=epsgrey  &&...
      filt(right(1),right(2))>=greythres))&& bw2(right(1),right(2))==0
    bw2(right(1),right(2))=1;
    endp2(i,:)=[right(1),right(2)];
   elseif  (M(up(1),up(2))>=M(right(1),right(2))  &&...
            ne(M(up(1),up(2)),0)&&  (M(up(1),up(2))>=Mthres))  ||...
           (abs(filt(r,c)-filt(right(1),right(2))) >...
           abs(filt(r,c)-filt(up(1),up(2)))...
          && abs(filt(r,c)-filt(up(1),up(2)))<=epsgrey ...
          &&  filt(up(1),up(2))>=greythres)
    bw2(up(1),up(2))=1;
    endp2(i,:)=[up(1),up(2)];
   else  i=i+1;
   end
  elseif  bw2(down(1),down(2))==0
   if  ((M(left(1),left(2))>=M(down(1),down(2))  &&...
      ne(M(left(1),left(2)),0)&&(M(left(1),left(2))>=Mthres))  ||...
       (abs(filt(r,c)-filt(left(1),left(2)))<=...
       abs(filt(r,c)-filt(down(1),down(2)))...
      && abs(filt(r,c)-filt(left(1),left(2)))<=epsgrey   &&...
       filt(left(1),left(2))>=greythres))&& bw2(left(1),left(2))==0
    bw2(left(1),left(2))=1;
    endp2(i,:)=[left(1),left(2)];
   elseif  (M(down(1),down(2))>=M(left(1),left(2))  &&...
       ne(M(down(1),down(2)),0)&&(M(down(1),down(2))>=Mthres))  ||...
       (abs(filt(r,c)-filt(left(1),left(2))) >...
      abs(filt(r,c)-filt(down(1),down(2)))...
           && abs(filt(r,c)-filt(down(1),down(2)))<=epsgrey...
           && filt(down(1),down(2))>=greythres)
    bw2(down(1),down(2))=1;
    endp2(i,:)=[down(1),down(2)];
   else i=i+1;
   end
  else i=i+1;
  end
end
```

```
function bw2=linklbe(bw,endp,M,dir,pend)
%Function that extends edges from endpoints according
%to LBE

%---- Input ----
%bw=edge image
%endp=endpoints in bw
%M=gradient magnitude of the source image
%dir=gradient direction of the source image
%pend=slope (see imcanny, parameter d)

%---- Output ----
%bw2= edge image after extension


[nr,nc]=size(M);

bw2=bw; %copy of edge image

%matrices initialized
v=zeros(nr,nc);
m=zeros(nr,nc);

%% Computation of matrices m, v, and LBE for each pixel
for i=2:nr-1
    for j=2:nc-1

        [m,v]=comp(i,j,M,pend,m,v,dir);


    end
end

LBE=zeros(nr,nc);
for i=1:nr
    for j=1:nc
        if ne(v(i,j),0) %handle the case of null denominator
        LBE(i,j)=m(i,j)/v(i,j);
        end
    end
end


endpl=endp; %copy of endpoints' list (left branch)
endpr=endp; %copy of endpoints' list (right branch)

```

```
pos=1; %initialization

 while pos<length(endp)
        y=endp(pos,1); %row
        x=endp(pos,2); %column
        fr=1; %flag for the right branch of the edge
        fl=1; %flag for the left branch of the edge
        while fr==1 %right branch
            y=endpr(pos,1);
            x=endpr(pos,2);
        if y>1 && x>1 && x<(nc) && y<(nr)

            [bw2,endpr,fr,LBE]=right(y,x,pos,bw2,LBE,dir,endpr,fr);


        else fr=0;
        end
        end
        while fl==1 %left branch
            y=endpl(pos,1);
            x=endpl(pos,2);
        if y>1 && x>1 && x<(nc) && y<(nr)

            [bw2,endpl,fl,LBE]=left(y,x,pos,bw2,LBE,dir,endpl,fl);

         else fl=0;
         end
        end
    pos=pos+1;
 end

%Delete short chains
bw2=bwareaopen(bw2,10);
```

```
function [m,v]=comp(i,j,M,pend,m,v,dir)
%subfunction of linklbe to compute vectors m and v

if dir(i,j)==1
    a=[i-1,j-1];
    b=[i+1,j+1];
    c=[i-1,j];
    e=[i+1,j];

elseif dir(i,j)==2
    a=[i-1,j+1];
    b=[i+1,j-1];
    c=[i-1,j];
    e=[i+1,j];

elseif dir(i,j)==3
    a=[i-1,j+1];
    b=[i+1,j-1];
    c=[i,j+1];
    e=[i,j-1];

else
    a=[i-1,j-1];
    b=[i+1,j+1];
    c=[i,j+1];
    e=[i,j-1];

end

if pend(i,j)>1-pend(i,j)

                v(a(1),a(2))=v(a(1),a(2))+1;
                u=M(a(1),a(2));
                v(b(1),b(2))=v(b(1),b(2))+1;
                d=M(b(1),b(2));
                v(i,j)=v(i,j)+1;
                p=M(i,j);
                vect=[u,d,p];
                [va,in]=max(vect);
                if ne(va,0)
                if in==1

                    m(a(1),a(2))=m(a(1),a(2))+1;

                elseif in==2
                     m(b(1),b(2))=m(b(1),b(2))+1;

                else
                    m(i,j)=m(i,j)+1;
                end
                end
            else
                v(c(1),c(2))=v(c(1),c(2))+1;
                u=M(c(1),c(2));
                v(e(1),e(2))=v(e(1),e(2))+1;
                d=M(e(1),e(2));
                v(i,j)=v(i,j)+1;
                p=M(i,j);
                vect=[u,d,p];
                [va,in]=max(vect);
                if ne(va,0)
                if in==1

                    m(c(1),c(2))=m(c(1),c(2))+1;

                elseif in==2
                     m(e(1),e(2))=m(e(1),e(2))+1;

                else
                    m(i,j)=m(i,j)+1;
            end
      end
end
```

```
function [bw2,endpr,fr,LBE]=right(y,x,pos,bw2,LBE,dir,endpr,fr)
%subfunction of linklbe to extend edges towards right direction

if dir(y,x)==1
    a=[y-1,x+1];
    b=[y,x+1];
```

```

elseif dir(y,x)==2
    a=[y+1,x+1];
    b=[y,x+1];

elseif dir(y,x)==3
    a=[y+1,x+1];
    b=[y+1,x];

else
    a=[y+1,x-1];
    b=[y+1,x];

end

                 pr=LBE(a(1),a(2));
                 dr=LBE(b(1),b(2));

                 r=[pr,dr];
                 [vr,ir]=max(r);

                 if ne(vr,0)

                     if ir==1 && bw2(a(1),a(2))==0
                         bw2(a(1),a(2))=1;
                         endpr(pos,:)=[a(1),a(2)]; %update endp right
                         if dr==1
                             LBE(b(1),b(2))=0.99;

                         end
                     elseif ir==2 && bw2(b(1),b(2))==0
                         bw2(b(1),b(2))=1;
                         endpr(pos,:)=[b(1),b(2)];


                     else fr=0;
                     end

                 else fr=0;

                 end
```

```
function [bw2,endpl,fl,LBE]=left(y,x,pos,bw2,LBE,dir,endpl,fl)
%subfunction of linklbe to extend edges towards left direction

if dir(y,x)==1
    a=[y+1,x-1];
    b=[y,x-1];

elseif dir(y,x)==2
    a=[y-1,x-1];
    b=[y,x-1];

elseif dir(y,x)==3
    a=[y-1,x-1];
    b=[y-1,x];

```

```
else
     a=[y−1,x+1];
     b=[y−1,x];

end

                    pl=LBE(a(1),a(2));
                    dl=LBE(b(1),b(2));

                    l=[pl,dl];
                    [vl,il]=max(l);

                    if ne(vl,0)

                         if il==1  && bw2(a(1),a(2))==0
                              bw2(a(1),a(2))=1;
                              endpl(pos,:)=[a(1),a(2)];
                              if dl==1
                                   LBE(b(1),b(2))=0.99;
                              end

                         elseif il==2 && bw2(b(1),b(2))==0
                              bw2(b(1),b(2))=1;
                              endpl(pos,:)=[b(1),b(2)];

                         else fl=0;
                         end
                    else fl=0;

                    end
```

```
function [imm,bw]=magagnincaiani(imm)
%Function that implements Magagnin and Caiani's method
%in "Semi−Automated analysis of coronary flow Doppler images:
%validation with manual tracings"



%% Creation of the overlapping regions
reg1=imm(:,1:200);
reg2=imm(:,150:350);
reg3=imm(:,300:end);

%% Histograms of the regions
[p1,q1]=hist(reg1(:),256);
[p2,q2]=hist(reg2(:),256);
[p3,q3]=hist(reg3(:),256);

r=0.25;
tot1=sum(p1);
target1=r*tot1;
i=length(q1);
sum1=0;
while sum1<target1 && i>1
     sum1=sum1+p1(i);
     i=i−1;
end
```

```
L1=q1(i);

tot2=sum(p2);
target2=r*tot2;
i=length(q2);
sum1=0;
while sum1<target2 && i>1
     sum1=sum1+p2(i);
      i=i-1;
end
L2=q2(i);

tot3=sum(p3);
target3=r*tot3;
i=length(q3);
sum1=0;
while sum1<target3 && i>1
     sum1=sum1+p3(i);
      i=i-1;
end
L3=q3(i);

%% Thresholded regions
reg1=imm(:,1:149)>L1;
reg12=imm(:,150:200)>((L1+L2)/2);
reg2=imm(:,201:299)>L2;
reg23=imm(:,300:350)>((L2+L3)/2);
reg3=imm(:,351:end)>L3;
im1=[reg1,reg12,reg2,reg23,reg3];

im1=imfill(im1,'holes');
im1=bwmorph(im1,'clean');
im1=bwareaopen(im1,30);

bw=edge(im1,'canny');
```

```
function [J,bw]=kiruthika(imm)
%function implementing Kiruthika et al. method in
%"Automated assessment of aortic regurgitation using
%2D Doppler echocardiogram"

%--- Input ---
% imm = cropped image (region of interest)

%--- Output ---
% filt = region of interest after median filtering, contrast stretching
  and
% Gaussian filtering
% J = processed image
% bw = extracted edge

%% Pre-processing
% Median filtering
im=medfilt2(imm,[3 3]);

% Contrast stretching
filt=contraststretch(im);
```

```

%Gaussian filtering
sigma=1;
filt=gaussfilt(filt,sigma);

%structuring element
se=strel('disk',5);
%morphological closing
im2=imclose(imm,se);

%difference image
out=filt-im2;
%intensity adjustement
J=imadjust(out);
%filling of holes
J=imfill(J,'holes');


%% Edge extraction
bw=edge(J,'canny',0.4,0.7);
bw=bwareaopen(bw,20);
```

```
function b=levelcurves(filt,cluster)
%Function that extracts the edge of a Doppler spectrum on the basis of
the
%level curves of its intensity

%--- Input ---
%filt = filtered image
%cluster = cluster to be taken as a contour

%--- Output ---
%b = level curve of interest

%% Clusterization (12 clusters)
[z,c]=kmeans(filt(:),12,'emptyaction','singleton');
[nr,nc]=size(filt);
%reconstructed image with clusters
pos=1;
imgrec=zeros(nr,nc);
for r=1:nc
    for k=1:nr

            imgrec(k,r)=z(pos);
            pos=pos+1;


    end
end

c=sort(c);

%% Extraction of the level curves from the filtered image
figure(10)
[C,h]=contour(filt,c(1:end));
title('Level curves')

```

```
% find the position of the level curve corresponding
% to the fourth centroid
pos=find(C(1,:)==c(cluster));
numpoints1=C(2,pos);
k=1;
for r=1:length(pos)
    for i=pos(r)+1:pos(r)+numpoints1(r)
        y1(k)=round(C(1,i));
        x1(k)=round(C(2,i));
        k=k+1;
    end
end

% b contains the chosen level curve (edge)
b=zeros(nr,nc);

for i=1:length(y1)
    b(x1(i),y1(i))=1;
end
```

```
function J = regiongrowing(im, pos, thres, dist)
% Region growing algorithm for 2D grayscale images
% --- Input ---
% im: grayscale image
% pos: Coordinates for initial seed position (if not given ginput is
used}
% thres: Absolute threshold level (if not given
% it is set to 5% of max-min)
% dist: Maximum distance to the initial position in [px] (if
% not given it is set to Inf)
%
%--- Output ---
% J: Binary mask (with the same size as the input image) indicating
% 1 for pixels in the region and 0 for outside

if ~exist('pos', 'var') || isempty(pos)
    himage = findobj('Type', 'image');
    if isempty(himage)
        himage = imshow(im, []);
    end

    % graphical user input for the initial position
    p = ginput(1);

    % get the pixel position concerning to the current axes coordinates
    pos(1) = round(axes2pix(size(im, 2), get(himage, 'XData'), p(2)));
    pos(2) = round(axes2pix(size(im, 1), get(himage, 'YData'), p(1)));
end

if ~exist('thres', 'var') || isempty(thres)
    thres = double((max(im(:)) - min(im(:)))) * 0.05;
end

if ~exist('dist', 'var') || isempty(dist)
    dist = Inf;
end

[nr, nc] = size(im);
```

```

% initial pixel value
val = double(im(pos(1), pos(2)));

% text output with initial parameters
disp(['RegionGrowing: Initial position (' num2str(pos(1))...
      '|' num2str(pos(2)) ') with '...
      num2str(val) ' as initial pixel value'])

% preallocate matrix
J = zeros(nr, nc);

% add the initial pixel to the queue
queue = [pos(1), pos(2)];

%% REGION GROWING ALGORITHM
while size(queue, 1)
  % the first queue position determines the new values
  x = queue(1,1);
  y = queue(1,2);


  % delete the first queue position
  queue(1,:) = [];

  % check the neighbors of the current pixel
  for i = -1:1
    for j = -1:1


      if x+i > 0  &&  x+i <= nr  &&...
         y+j > 0  &&  y+j <= nc  &&...
         ...
         any([i, j])       &&...    % the current pixel is redundant
         ~J(x+i, y+j) &&...
         sqrt( (x+i-pos(1))^2 +...
               (y+j-pos(2))^2) < dist &&...
         im(x+i, y+j) <= (val + thres) &&...
         im(x+i, y+j) >= (val - thres)


         J(x+i, y+j) = 1;


         % add the current pixel to the computation queue
         queue(end+1,:) = [x+i, y+j];
      end
    end
  end
end

   %Fill the holes in the region, if they exist
   J(:,:) = imfill(J(:,:), 'holes');

% text output
disp(['RegionGrowing Ending: Found ' num2str(length(find(J)))...
      ' pixels within the threshold range'])
```

```
function r=baseline(im)

%This function returns the row containing the baseline
%in Doppler spectral images

[nr,nc]=size(im);

BW=edge(im,'canny');

%% Hough transform

[H,theta,rho] = hough(BW);
P = houghpeaks(H,5,'threshold',ceil(0.3*max(H(:))));
lines = houghlines(BW,theta,rho,P,'FillGap',5,'MinLength',7);

figure, imagesc(im),colormap(gray), hold on

%Detection of the longest straight line
max_l = 0;
for k = 1:length(lines)
   %starting (column,row) and ending point of the line segment
   xy = [lines(k).point1; lines(k).point2];

   len = norm(lines(k).point1 - lines(k).point2);
   if ( len > max_l)
      max_l = len;
      xy_long = xy;
   end
end

% highlight the longest line segment
plot(xy_long(:,1),xy_long(:,2),'LineWidth',2,'Color','red');
r=xy_long(1,2); %row corresponding to the baseline
```

Bibliography

[1] Edges: The Canny Edge Detector, 1996.

[2] Fast course of mathematical morphology, 1997.

[3] Canny Edge Detector, 2003.

[4] MATLAB and Octave Functions for Computer Vision and Image Processing, 2010.

[5] Azhari, H. *Basics of biomedical ultrasound for engineers.* Wiley & sons.

[6] Bergholm, F. Documentation of developed Image Segmentation and Methodology. Tech. rep., KTH Division of Highway and Railway Engineering, 2010.

[7] Briechle, K., and Hanebeck, U. D. Template matching using Fast Normalized Cross Correlation.

[8] Canny, J. F. A computational approach to edge detection. *IEEE Trans. on Pattern Analysis and Machine Intelligence 8*, 6 (1986), 679–698.

[9] Evans, D. H., and McDicken, W. N. *Doppler ultrasound: physics, instrumentation and signal processing.* Wiley & sons.

[10] Gonzales, R. C., and Woods, R. E. *Digital image processing.* Prentice Hall, 2008.

[11] Gonzalez, R. C., and Wintz, P. *Digital image processing.* Addison Wesley.

[12] Greenspan, H., Shechner, O., Scheinowitz, M., and Feinberg, M. S. Doppler echocardiography flow-velocity image analysis for patients with atrial fibrillation. *Ultrasound in medicine and biologigy 31*, 8 (2005), 1031–1040.

[13] Hancock, E. R., and Kittler, J. Adaptive estimation of hysteresis thresholds. *Computer vision and pattern recognition* (1991), 196–201.

[14] Heath, M., Sarkar, S., Sanocki, T., and Bowyer, K. Comparison of edge detectors. A methodology and initial study. *Computer vision and image understanding 69*, 1 (1998), 38–54.

[15] Hoskins, P., Martins, K., and Thrush, A. *Diagnostic ultrasound. Physics and equipment.* Cambridge University Press.

[16] Hussein, W. B., Moaty, A. A., Hussein, M., and Becker, T. A novel edge detection method with application to the fat content prediction in marbled meat. *Pattern Recognition 44* (2011), 2959–2970.

[17] Kass, M., Witkin, A., and Terzopoulos, D. Snakes: active contour models. *International Journal of Computer Vision 44* (1987), 321–331.

[18] Kiruthika, N. V., Prabhakar, B., and Reddy, M. R. Automated assessment of aortic regurgitation using 2D Doppler Echocardiogram. In *IST 2006 - International Workshop on Imaging* (2006).

[19] Lacroix, V. Combining evidence for edge detection. In *Proceedings of the 5th Scandinavian conference on Image Analysis* (1987).

[20] Lewis, J. P. Fast template matching. *Vision interface* (1995), 120–123.

[21] Li, J., Randall, J., and Guan, L. Perceptual image processing for digital edge linking. 2003.

[22] Loizou, C. P., and Pattichis, C. C. *Despeckle filtering algorithms and software for Ultrasound Imaging.* Morgan & Claypool, 2008.

[23] Magagnin, V., Caiani, E. G., L.Delfino, Champlon, C., Cerutti, S., and Turiel, M. Semi-Automated analysis of coronary flow Doppler images: validation with manual tracings. In *Proceedings of the 28th IEEE. EMBS Annual International Conference* (2006).

[24] Maini, R., and Aggarwal, H. Study and Comparison of Various Image Edge Detection Techniques. *International Journal of Image Processing 3*.

[25] Medina-Carnicer, R., Madrid-Cuevas, F. J., Carmona-Poyato, A., and MuÑoz-Salinas, R. On candidates selection for hysteresis thresholds in edge detection. *Pattern Recognition 42* (2008), 1284–1296.

[26] Medina-Carnicer, R., Madrid-Cuevas, F. J., MuÑoz-Salinas, R., and Carmona-Poyato, A. Solving the process of hysteresis without determining the optimal thresholds. *Pattern Recognition 43* (2009), 1224–1232.

[27] Medina-Carnicer, R., MuÑoz-Salinas, R., Yeguas-Bolivar, E., and Diaz-Mas, L. A novel method to look for the hysteresis thresholds for the canny edge detector. *Pattern Recognition 44* (2011), 1201–1211.

[28] Miller, F. L., Maeda, J., and Kubo, H. Template based method of edge linking using a weighted decision. In *Proceedings of the 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems* (2003).

[29] Nadernejad, E., Sharifzadeh, S., and Hassanpour, H. Edge Detection Techniques: Evaluation and Comparison. *Applied Mathematical Sciences 2* (2008).

[30] Ramesh, J., Kasturi, R., and Schunck, B. G. *Machine vision.* McGraw-Hill.

[31] Sonka, M., Hlavac, V., and Boyle, R. *Image processing, analysis, and machine vision.* Brooks/Cole, 1999.

[32] Tschirren, J., Lauer, R. M., and Sonka, M. Automated analysis of Doppler ultrasound velocity flow diagrams. *IEEE Transactions on medical imaging 20*, 12 (2001), 1422–1425.

[33] Yitzhaky, Y., and Peli, E. A method for objective edge detection and detector parameter selection. *IEEE Trans. on Pattern Analysis and Machine Intelligence 25*, 8 (2003), 1027–1033.