Stony Brook University

Enforcing Invariance Constraints in Medical Imaging to Detect Heart Fibrillation

A Thesis Presented

by

George Valentin Iordache

to The Graduate School

in Partial Fulfillment of the

Requirements

for the Degree of

Master of Science

in

Computer Science

Stony Brook University

August 2008

Stony Brook University

The Graduate School

George Valentin Iordache

We, the thesis committee for the above candidate for the Master of Science degree, hereby recommend acceptance of this thesis.

Radu Grosu – Thesis Advisor
Associate Professor, Computer Science Department

Scott Smolka - Chairperson of Defense
Professor, Computer Science Department

Emilia Entcheva
Associate Professor,
Biomedical Engineering Department, Stony Brook University

This thesis is accepted by the Graduate School

Lawrence Martin
Dean of the Graduate School

Abstract of the Thesis

Enforcing Invariance Constraints in Medical Imaging to Detect Heart Fibrillation

by

George Valentin Iordache

Master of Science

in

Computer Science

Stony Brook University

2008

This thesis addresses the problem of shape identification in images, with invariance assurances for translation, scaling and rotation. The concrete application of our methods is detection of spiral electric waves, a precursor of atrial and ventricular fibrillation. We present a thorough theoretical analysis of shape identification methods, focusing on the Scale-Invariant Feature Transform (SIFT) algorithm. In parallel, we analyze a learning-based technique employed by the EMERALD tool, that uses spatial superposition to model excitable hybrid automata and detect the onset of spiral waves. We also propose an enhancement of the original methods, which consists in a pre-processing step for shape extraction. The advantage is an improvement of invariance to translation and contextual information, with better overall performance in shape identification.

Table of Contents

List of Figures

Chapter 1. Introduction

1.1 Goal of the Thesis

The electrical signal propagation in the cardiac tissue determines important aspects of heart functioning. Capturing and analyzing the electric signal could help detect and predict major heart problems, including arrhythmia that puts million of people at risk each year. Our thesis focuses on detecting electric spiral waves, which are a precursor of heart fibrillation.

Our study analyzes the spiral identification problem and conducts experiments in two directions. One approach employs feature-based detection algorithms to identify shapes in images, while the other uses a classifier to learn object classes (spirals) and directly classify shapes.

The major challenge that guides our work is to provide robust shape identification algorithms that would ensure invariance to rigid transformations, such as translation, rotation and scaling, as well as to noisy inputs.

1.2 Capturing and Analyzing Electrical Signal Snapshots in Cardiac Tissues

Heart cells display a hybrid behavior, responding to external excitations that exceed the *action potential* threshold. A novel approach uses Cycle-Linear Hybrid Automata (CLHA) to model the electrical behavior of cardiac cells [1]. The mode of each automaton (resting, simulated, upstroke, and plateau) is given as a probability distribution, which allows the definition of mode superposition for a set of CLHA. Applying superposition successively, gives a tree structure, and, in particular for 4 modes, a quad-tree structure, where inner nodes are mode-superposition of sub-trees, and leaves are modes of individual CLHA.

The experiments conducted in the thesis use images obtained with the CellExcite [27] simulator, which generates, at successive time steps, snapshots of a 400 x 400 CLHA network and their mode abstractions.

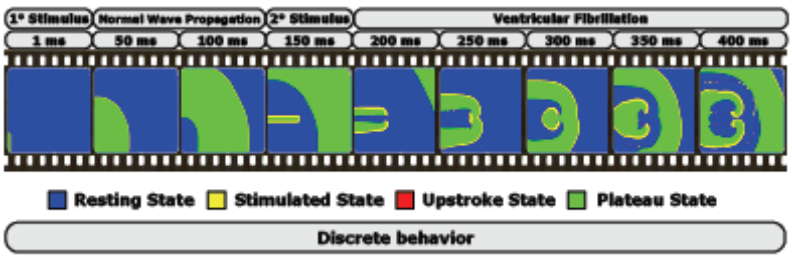


Figure 1.1 Successive signal snapshots obtained with CellExcite

Figure 1.1 [1] presents images of the electric signal obtained with the CellExcite simulator, where the value of the action potential has been discretely abstracted to the corresponding mode, to reduce the system state space. With the discrete behavior, the wave shape is preserved, presenting no impediment for spiral identification.

1.3 2D Shape Identification with Invariance Assurances

The analyzed snapshots are in the form of 2D images, where the pattern of interest is represented by spirals. The position, orientation and scale of the shapes in images may vary significantly, rendering the shape identification process difficult. Therefore, spiral detection presents some major challenges, given by the need to enforce invariance to rigid transformations, such as translation, rotation and scaling. Furthermore, the images may present noise that can as well interfere with a correct identification. Throughout the thesis, we analyze shape identification algorithms that are robust at geometric transformations and noise in input images.

Chapter 2. Related Work

In visual systems, shape identification goes beyond finding identical shapes in images. The target is to identify similar shapes after possible spatial transformations, like translation, scale or rotation, and also in spite of different illumination conditions, noise or even partial occlusion. Therefore, geometric transformations that achieve invariance do not suffice for a correct shape matching in real images. This topic has been extensively explored in literature, but, due to its difficulties mentioned before, remains an open and challenging research theme. A comprehensive description of the state of the art in shape identification can be found in [10, 11].

2.1 Current Research on Shape Identification

We continue with an overview of the approaches taken in the literature to detect and identify shapes in images. Depending on the image processing method, there are two major approaches: a global one, based on brightness, color or texture distribution, in which the image is analyzed as a whole, and a local approach, based on local features, namely local image structures formed by pixels of high intensity variation.

2.1.1 Global Identification Methods

With global shape identification, the focus is on the integral visible part of the object to find similarities in images, rather than on the contours. For greyscale images, one can measure the grey level, or brightness, of the pixels after applying specific transformations to find correspondences [12, 13, 14].

Yet another approach builds a classifier directly, without finding explicit similarities. A good classifier needs a strong learning algorithm and a large number of examples to handle invariance aspects. A few examples where this method has been successfully applied include the principal component analysis (PCA) in face recognition [18], LeNet classifier for appearance-based face matching [19], and the Support Vector Machine (SVM) – based methods [20, 21].

2.1.2 Local Identification Methods

One direction that has benefited from extensive attention in recent years studies the information at the boundary of silhouette images, shapes where the internal contours are ignored, and the boundary is conveniently represented as a closed curve, parameterized by arclength [22]. Within this category fall methods that detect the skeleton of an image [15, 16], which characterizes the boundary shape using the differential singularities of the reaction equation introduced by Kimia et al. [17]. The skeleton represents the singularities in the curve (boundary) evolution [15]. In this approach, the skeleton is computed first, from the shape-boundary information, then it is used to represent the differential structure of the original boundary, and finally the matching phase follows by comparing the skeletons of

different shapes [15].

There are also several approaches in shape recognition based on interest points, keypoints or landmarks, where the image content locally changes in the two directions. These points convey more information due to signal changes, being therefore more representative for the image, and having high discriminative power [23, 24, 25]. Furthermore, they are invariant to geometric transformations, and partially to other types of changes, as illumination, viewpoint, etc. Lindeberg conducts a comprehensive mathematical analysis of scale-space theory in computer vision [34].

2.2 Contribution

The principal objective of this study is to develop an interest point detector capable of dealing with significant image transformations. The points extracted by this detector are used for matching and recognition. We are mainly concerned with the identification of a particular pattern, namely 2D spirals that represent the electric signal in the heart tissue. Since our images model the electric signal in the heart, we are not concerned with variations in illuminations or occlusion that may appear in photographic images, but only with rigid transformations, to ensure invariance to translation, scale and rotation. Our images can also display noise and non-regularities.

In the first part of the thesis we apply local identification methods based on interest points to identify the shapes. It is important to notice that smooth shapes (e.g. circles) or regions do not have distinguished keypoints, therefore shape information available in smooth regions is not used. We enforce stronger invariance constrains for translation by first extracting the shape to remove the surrounding context and then applying the matching algorithm for images containing individual shapes only. Since our shapes are not overlapping, and are clearly separated, with few interior contours, a silhouette-related approach is appropriate to detect the contours and extract the shape. Yet we do not limit the detection algorithm to closed external contours, but also extend it to give the complete shape even in the case of objects that have holes or several adjacent contours.

In the second part of the thesis, we compare the first approach described above with a novel approach based on a global identification method that uses a classifier to recognize spirals in images. This method uses the EMERALD tool suite [1]. The attributes for classification correspond to regions of the image along the path from the potential center of the spiral, which are modeled by the nodes in a superposition quad-tree. We employ learning and classification methods to train the classifier and decide if a shape is spiral.

Chapter 3. Scale-invariant Feature Transform (SIFT)

This chapter starts with an analysis of images in the spatial and frequency domains. Next, we look at theoretical aspects of Gaussian filters and signal decompostion with Fourier series. The study is intended to favor a better understanding of the Scale-invariant Feature Transform algorithm presented in the last part of the chapter.

3.1 From Spatial Domain to Frequency Domain

We can analyze an image in two contexts:

- Spatial domain: that represents the input image where each point is a pixel intensity
- Frequency domain: that represents the image after applying Fourier transform. The Fourier transform of the image is decomposed in its sine and cosine components. In the frequency domain, each point represents a particular *frequency* contained in the spatial domain image

Figure 3.1 [28] shows the space and frequency domains of an image. The space domain is the original image (up), and the frequency domain contains the frequencies of the pixel intensities in the original image (bottom).

For the input image given in Figure 3.1, we see that there are a few points in the frequency domain, disposed in a line perpendicular on the stripes orientation. That happens because the image intensity in the spatial domain changes the most if we go along it horizontally or vertically, respectively. The central point is a mark of the center (origin of axes), for zero frequencies, because we are representing symmetric frequencies (both positive and negative).

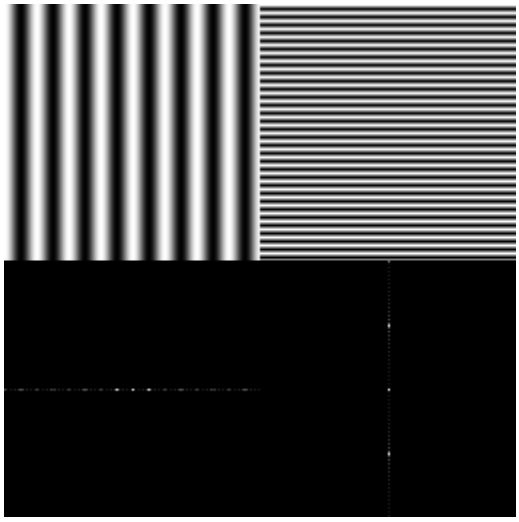
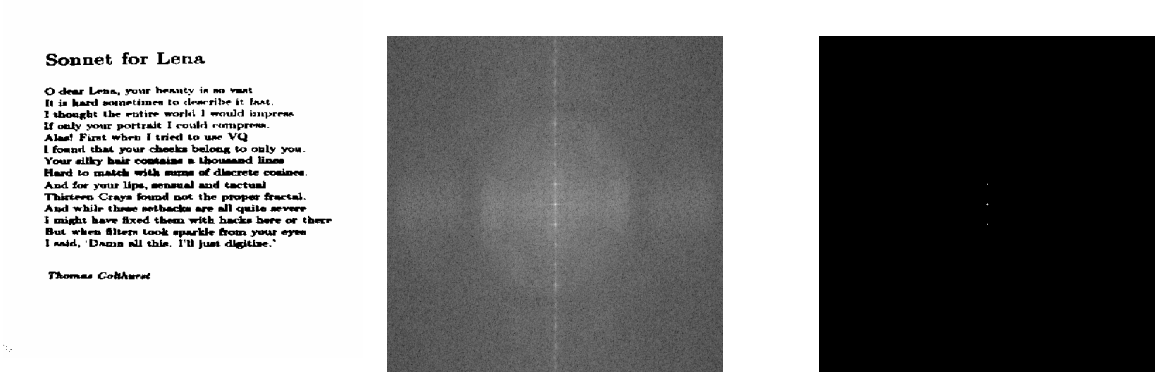


Figure 3.1 The space and frequency domains of images representing vertical and horizontal stripes

In Figure 3.1 and Figure 3.2 [29], the intensity values in the Fourier image that can not be displayed on the screen, appear as black. To overcome this issue, Figure 3.2 (center) displays the logarithm of the magnitude of the Fourier transform of the analyzed image.

The main frequency values (Figure 3.2, left) lie on a vertical line, indicating that the text lines in the input image are horizontal; therefore image intensity in the spatial domain changes the most if we go along it vertically.



Initial image, in space domain

The logarithm of the magnitude of the Fourier transform

The thresholded magnitude of the Fourier transform

Figure 3.2 The space and frequency domains of an image

3.2 Gaussian Filters

We will next analyze the effect of convolution with a Gaussian function, in the frequency domain of an image. For better understanding, we will start from the spatial domain, where the convolution can be written as:

```
input image  • Gaussian Function = output image
```

where the • sign signifies the convolution operator.

In the frequency domain, the Fourier transform of the convolution is given by the product of the Fourier transforms:

```
Fourier transform of input image • Fourier transform of Gaussian Function=
= Fourier transform of output image
```

Next, we will re-write the previous formula to focus on the Fourier transform of Gaussian Function:

```
Fourier transform of Gaussian Function =
```

= Fourier transform of output image / Fourier transform of input image

At this point, we use a mathematical result that says that Fourier transform of a Gaussian is also Gaussian in shape [30]. Therefore, applying a Gaussian has the same effect on frequencies as applying a Fourier transform of a Gaussian.

In order to see the effect of the Gaussian filter on different spatial frequencies, we need to look at the Fourier transform of the filter. The effect on the spatial frequencies is called *frequency response* of the filter.

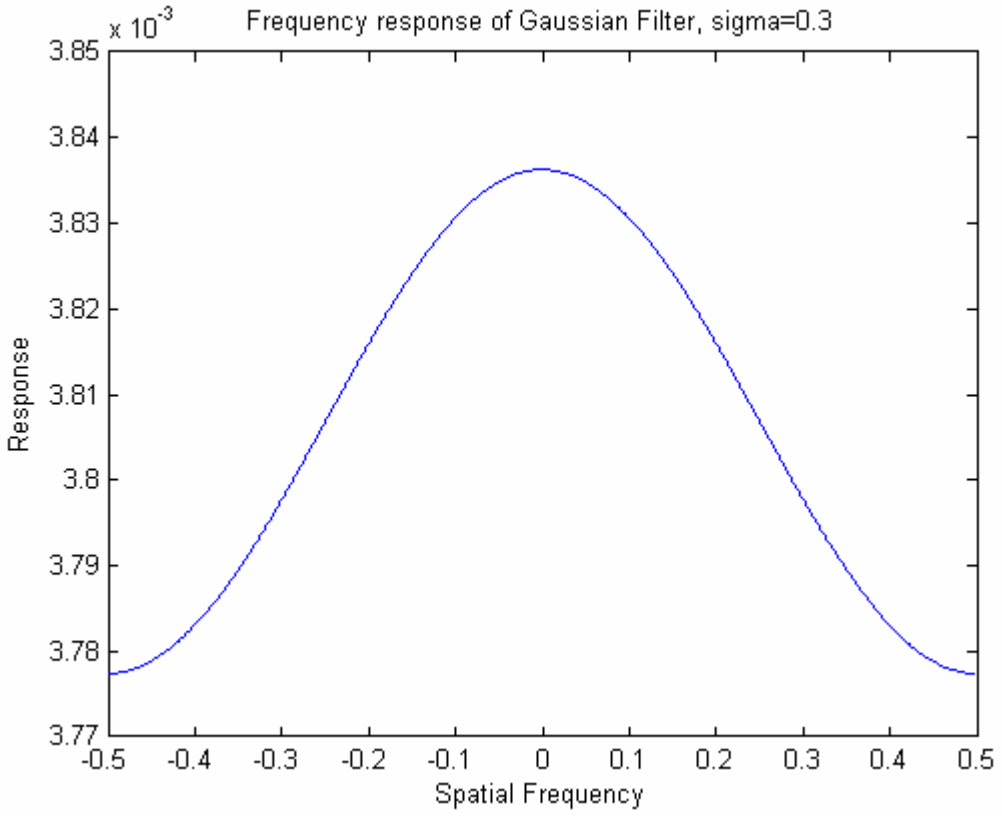


Figure 3.3 Frequency response of a Gaussian filter with σ = 0.3

In Figure 3.3, we show the frequency response of a Gaussian filter with σ = 0.3. We observe that the form of the curve is also Gaussian in shape, letting the low frequencies to pass and attenuating the high frequencies.

Therefore, the Gaussian filter is a *low-pass filter*. In general, there are two categories of filters: high-pass and low-pass filters (Figure 3.4 [31]). The high-pass filters attenuate low frequencies to a higher degree than high frequencies, while low-pass filter do the opposite.

Original image:

High-pass filter:

Low-pass filter (Gaussian blur):

Figure 3.4 Applying high-pass and low-pass filters on a image

The Gaussian filter is a low-pass filter, which means that:

- High frequency image information (detail) is blocked (more attenuated).
- Low frequency image information (little detail) is passed (less attenuated).

To understand the behavior of the Gaussian filter as a low-pass filter, we need to look at the Gaussian function:

$$G(r) = \frac{1}{(2\pi\sigma^2)^{N/2}} e^{-r^2/(2\sigma^2)}$$

where r is the blur radius and σ is the blurring coefficient.

The G(r) function generates a surface whose contours are concentric circles with a Gaussian distribution from the center point. Figure 3.5 shows G(r) for σ = 1.84. The surface is illustrated with a matrix, where the center is the highest value, and the values are disposed circular, having a fast decrease.

0.0037	0.0077	0.0120	0.0139	0.0120	0.0077	0.0037
0.0077	0.0161	0.0251	**0.0291**	0.0251	0.0161	0.0077
0.0120	0.0251	0.0391	0.0453	0.0391	0.0251	0.0120
0.0139	**0.0291**	0.0453	**0.0525**	0.0453	**0.0291**	0.0139
0.0120	0.0251	0.0391	0.0453	0.0391	0.0251	0.0120
0.0077	0.0161	0.0251	**0.0291**	0.0251	0.0161	0.0077
0.0037	0.0077	0.0120	0.0139	0.0120	0.0077	0.0037

Figure 3.5 Convolution matrix, G(r), for σ = 1.84

When applying a Gaussian filter, we first compute a convolution matrix. The matrix is then used to re-compute each pixel's value as a weighted average of that pixel's

neighborhood. The original pixel's value receives the heaviest weight, because it has the highest Gaussian value in the convolution matrix. Neighboring pixels receive smaller weights as their distance to the original pixel increases.

3.3 Signal Approximation with Finite Fourier Series

The Fourier series represent the decomposition of a signal in a sum of sine and cosine functions. Therefore, we can view the Fourier series of a signal as a superposition of simple sine and cosine waves.

The Fourier formula is given by an infinite sum:

$$f(x) = \frac{a_0}{2} + \sum_{1}^{\infty} \left[a_n \cos(nx) + b_n \sin(nx) \right]$$

where a_i and b_i represent the Fourier coefficient and are computes as:

$$a_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \cos(nx) dx$$

and

$$b_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \sin(nx) dx$$

The formula above represents the Fourier series of a periodic function, namely a function defined such that: $f(x + 2\pi) = f(x)$ for all real numbers x.

The previous representation is the original one that was introduced by Fourier, while modern versions use complex exponentials. The complex representation uses Euler's formula: $e^{inx} = \cos(nx) + i \sin(nx)$, where 'i' is the imaginary unit. The advantage of this representation is given by a more concise formulation. We can now represent the Fourier series as:

$$f(x) = \sum_{n=-\infty}^{\infty} c_n e^{inx}$$

and the Fourier coefficients as:

$$c_n = \frac{1}{2\pi} \int_{-\pi}^{\pi} f(x) e^{-inx} dx$$

We observe that with this representation, the index n also takes negative values. The relation between an, bn, cn coefficients is:

$a_n = c_n + c_{-n}$ for $n \geq 0$,

and

$b_n = i(c_n - c_{-n})$ for n > 0.

Since we used the first representation in our implementation, we will continue the discussion based on it. In practice, the convergence issue is a very important one. It is often necessary to represent the infinite sum with a finite one, which will approximate the original signal:

$$f(x) = \frac{a_0}{2} + \sum_{1}^{N} [a_n \cos(nx) + b_n \sin(nx)]$$

and we call it the N-harmonic Fourier series approximation of the signal. Increasing the N gives us a better approximation of the original signal.

We have studied how a signal is approximated with finite Fourier series with MATLAB. We have looked at a single row in a spiral image, and viewed its varying intensity as the amplitude of a periodic signal. Figure 3.6 shows the signal corresponding to a line in the image of a spiral.

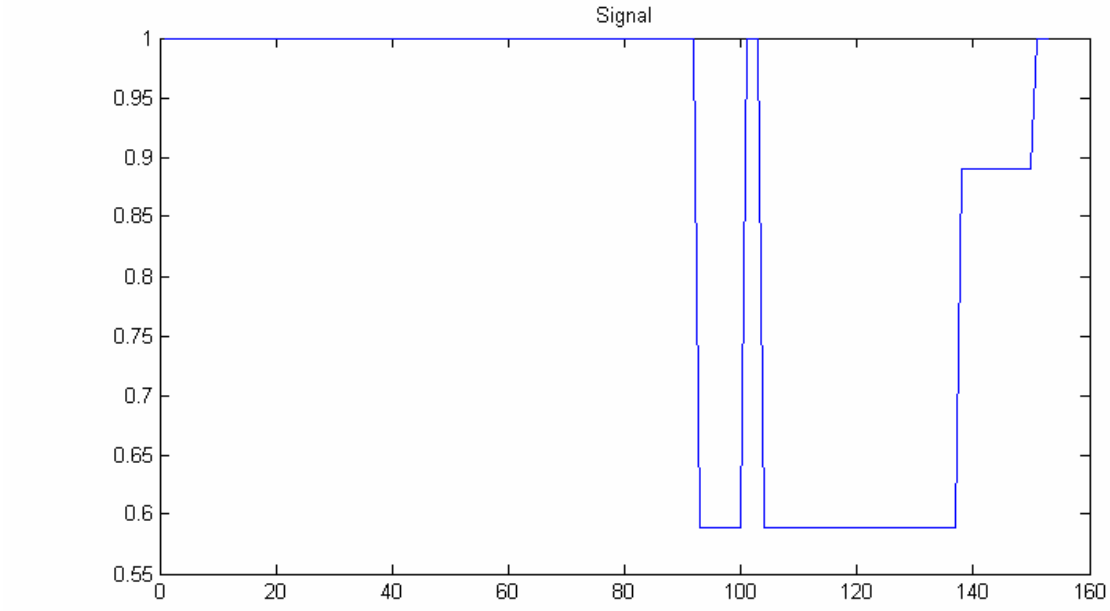


Figure 3.6 Signal modeling the intensity variation in a row from a spiral image

Next, we discuss how the original signal can be decomposed in sine and cosine waves. Figure 3.7 and Figure 3.8 show the signal approximation, for different harmonics. Figure 3.7 displays all the harmonics up to N = 40, and re 3.8 displays only 4 harmonics (N=2, N=12, N=22, N=32), together with the original signal, for better visualization. From the figure, it is clear that the original signal is better approximated with higher order harmonics. For N=2, the approximation is a simple cosine wave.

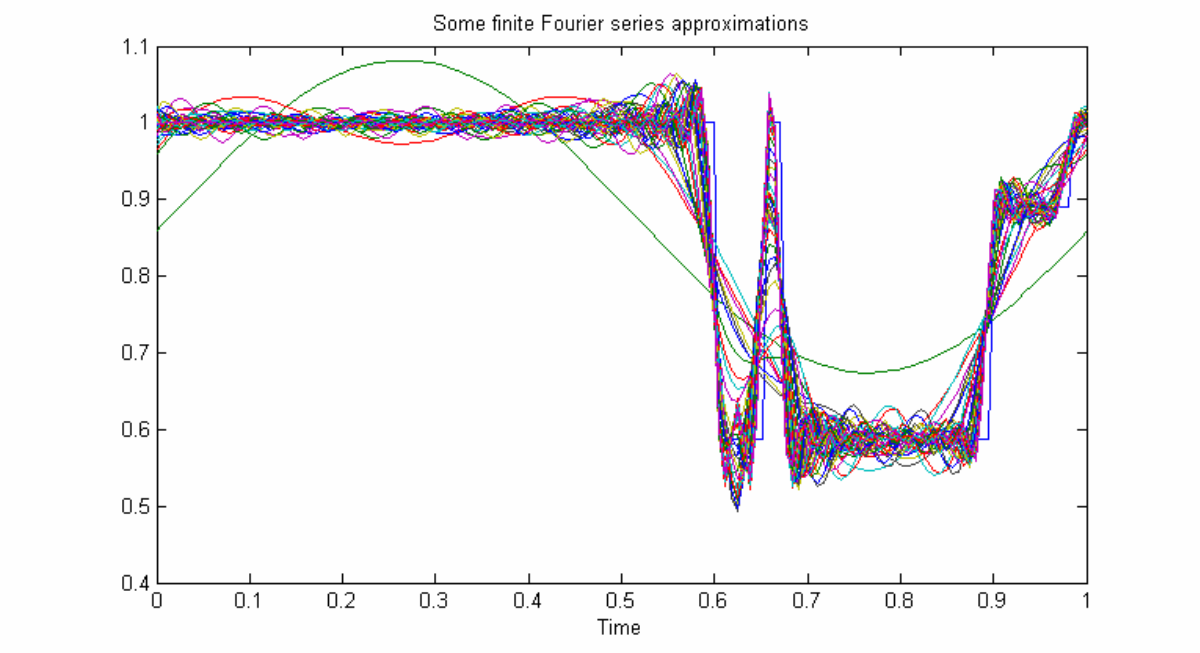


Figure 3.7 Signal approximation at various harmonics

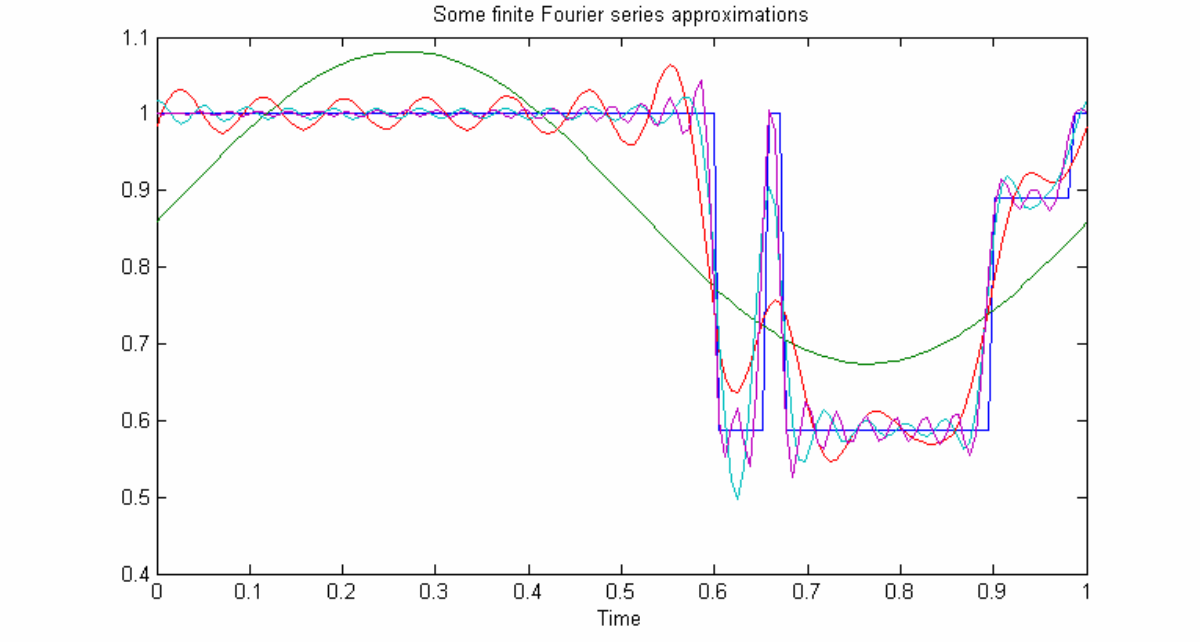


Figure 3.8 Signal approximation at various harmonics

It is also interesting to observe the magnitudes of Fourier coefficients, since they represent the amplitudes of the sine/cosine wave components (Figure 3.9). The highest coefficient is a_0, described by the middle vertical line. For higher frequency components (a1,

a2 ...), the coefficient values is smaller, getting closer to zero, This shows that signal approximation up to a particular harmonic is acceptable, since the influence of signal components at high frequencies is small.

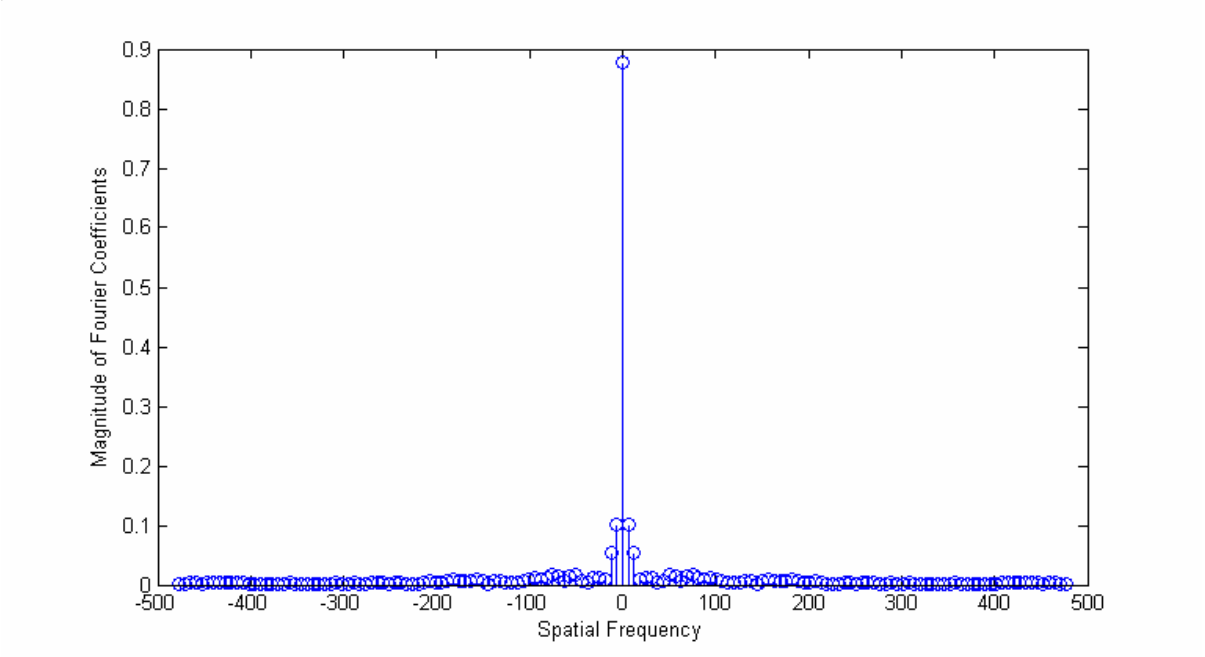


Figure 3.9 The magnitude of Fourier coefficients

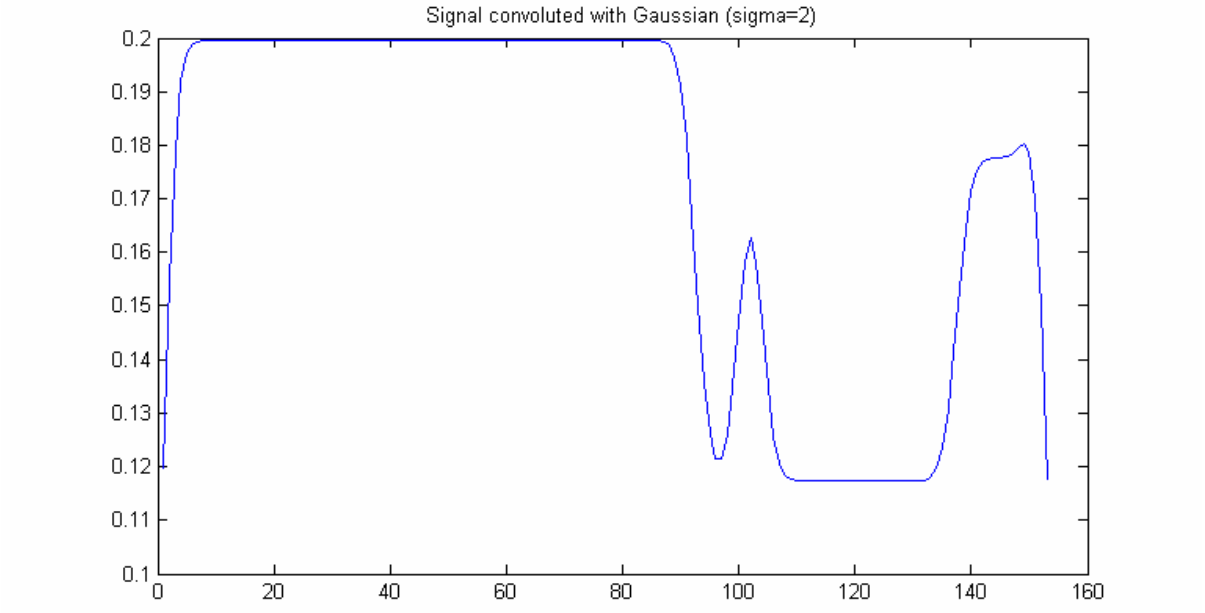


Figure 3.10 Signal convoluted with Gaussian (sigma=2)

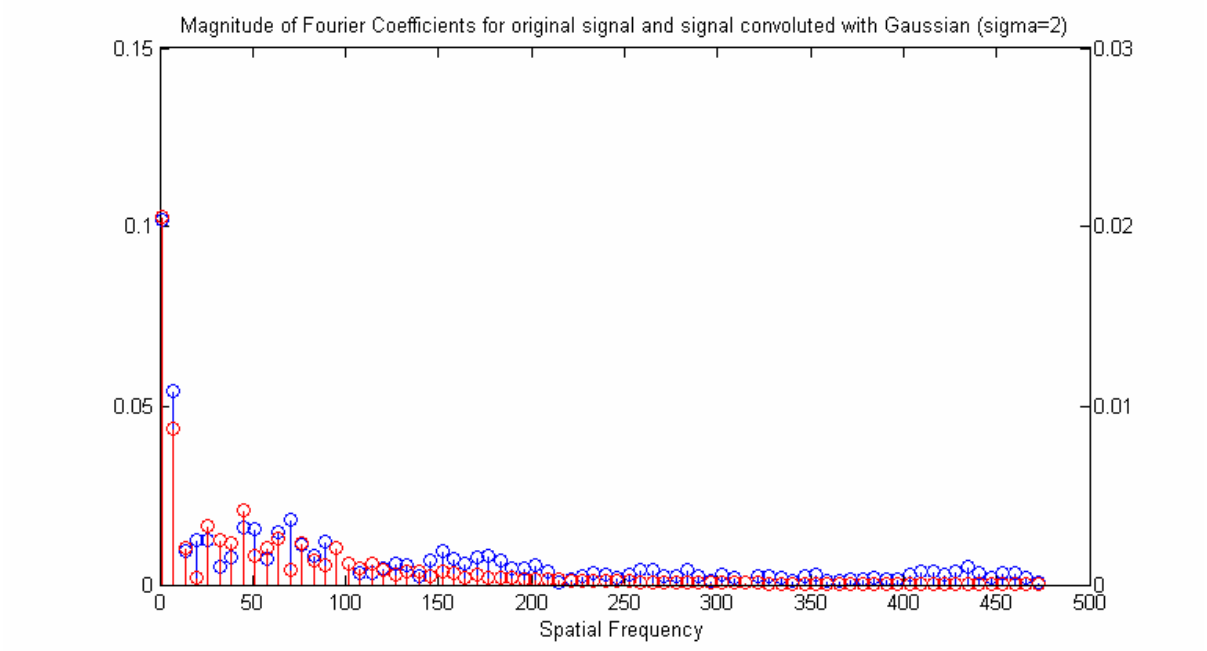


Figure 3.11 Magnitude of Fourier coefficients for original signal (blue lines) and for signal convoluted with Gaussian, sigma=2 (red lines)

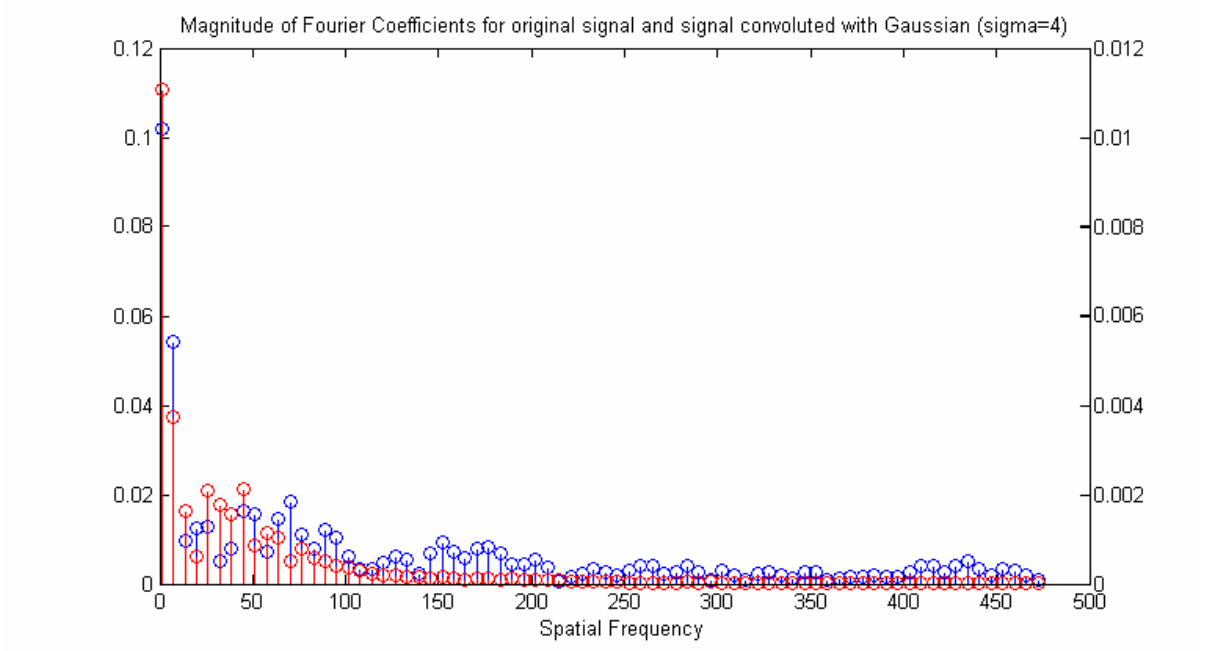


Figure 3.12 Fourier coefficients for signal convoluted with Gaussian, sigma=2 (blue lines) and signal convoluted with Gaussian, sigma=4 (red lines)

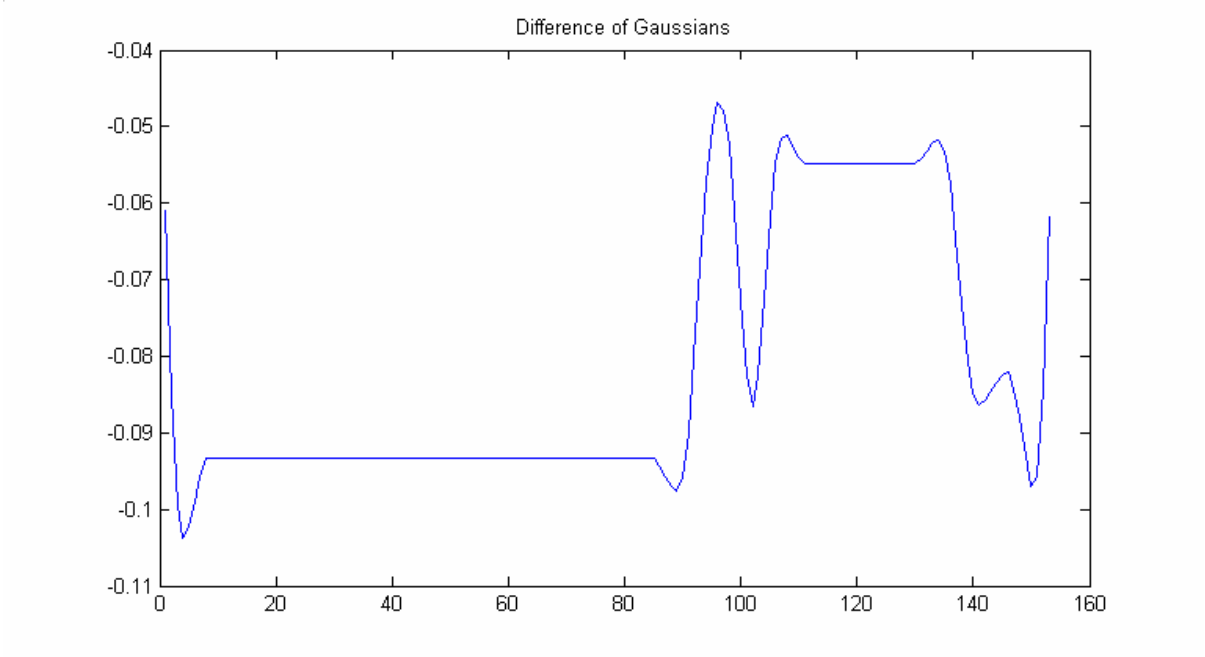


Figure 3.13 Difference of Gaussians (DoG) between Gaussians with sigma = 2 and sigma = 4 applied on the original signal

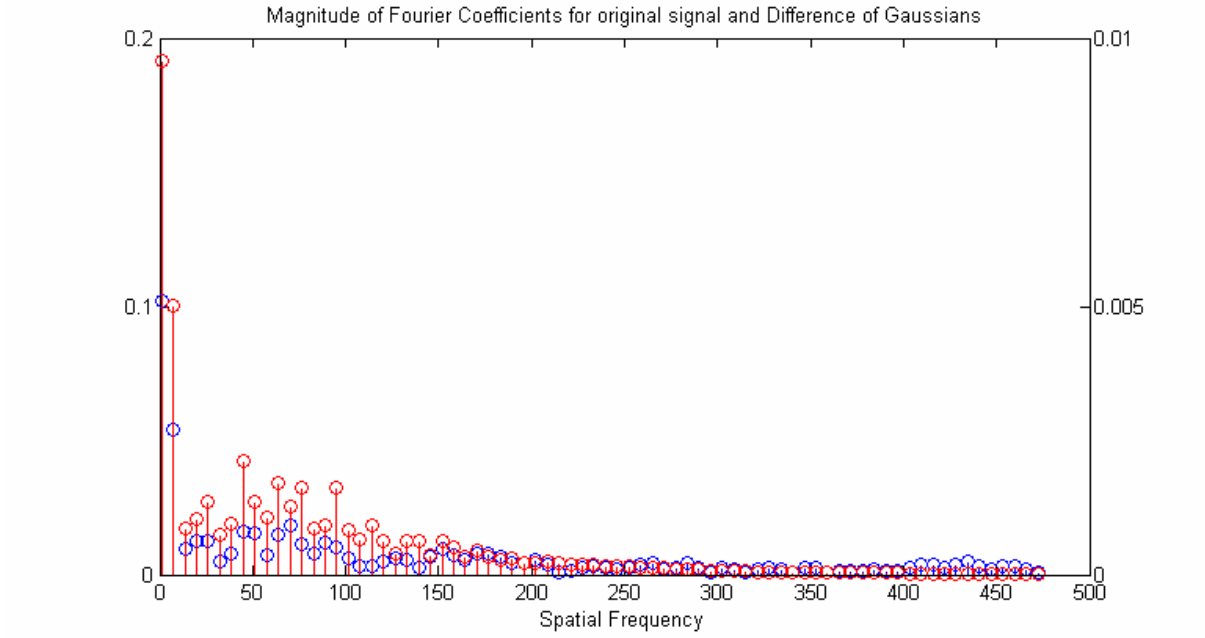


Figure 3.14 Magnitude of Fourier coefficients for original signal (blue lines) and DoG of the signal (red lines)

We also looked at the same signal blurred using a Gaussian filter. Figure 3.10 and

Figure 3.11 illustrate a Gaussian filter with sigma=2. Figure 3.10 shows the signal wave after applying the Gaussian filter. Figure 3.11 shows the comparison between the Fourier coefficients of the original signal (the blue lines) and for the convoluted signal (the red lines). We observe that the Fourier coefficients at high frequencies have decreased in the case of the convoluted signal compared to the initial signal, which supports the idea that the Gaussian filter is a low pass filter, attenuating high frequencies.

A further comparison between two different signal convolutions (sigma=2 and sigma=4) is presented in Figure 3.12. We see that increasing the sigma results in a stronger attenuation of high frequency components of the signal.

The next aspect that we study is the Difference of Gaussians (DoG). In Figure 3.13 we show the DoG between two blurred signals, one obtained by convoluting the original signal with a Gaussian filter having sigma = 2, and the other obtained by convoluting the original signal with a Gaussian filter having sigma = 4. An interesting observation comes from looking at the magnitude of Fourier coefficients for original signal (blue lines) and DoG of the signal (red lines) in Figure 3.14. We see that for low frequencies, the Fourier coefficients of DoG have higher values than the ones of the signal, while for high frequencies, the relation is inversed: the Fourier coefficients of DoG have lower values than the ones of the signal. Therefore, the DoG sine and cosine components are amplified at lower frequencies, and attenuated at higher frequencies.

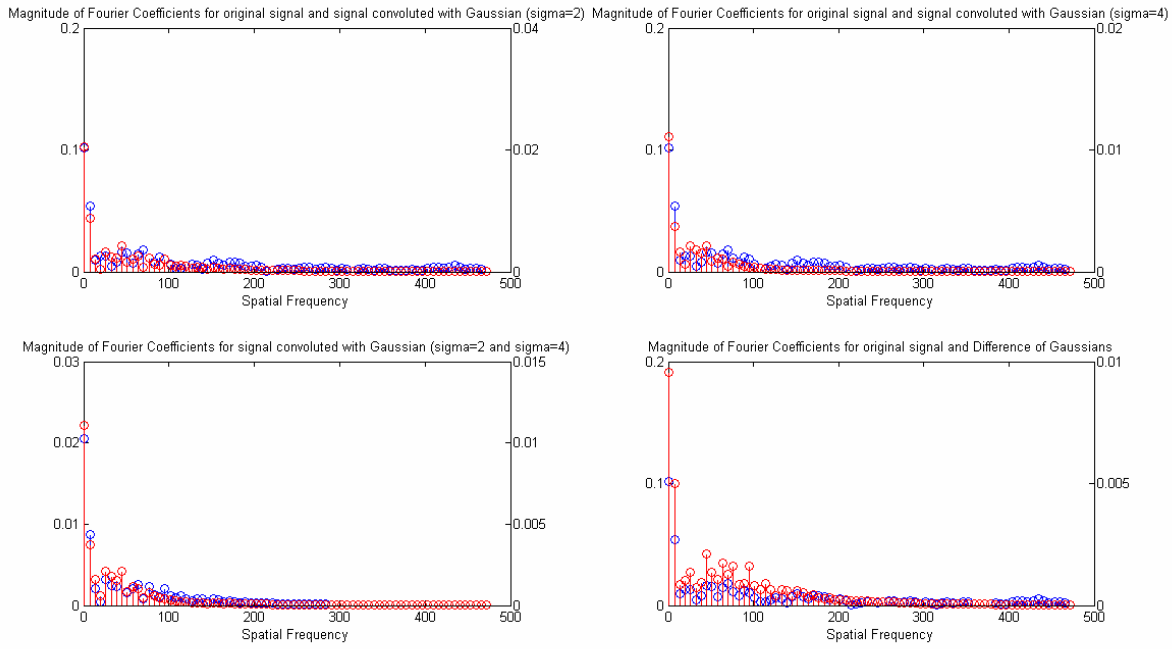


Figure 3.15 Synthesis: Magnitude of Fourier coefficients for original signal, Gaussian convolutions and DoG

Finally, Figure 3.15 is a synthesis of the aspects discussed before. It shows in parallel

the magnitude of the Fourier coefficients for the blurred signal with different sigma values, and for the Difference of Gaussians.

3.4 SIFT Algorithm

The Scale invariant feature transform algorithm – SIFT – is a computer vision algorithm used to detect and describe features in a given image. This algorithm was proposed first by David Lowe [23] in 1999. Due to its feature detection properties, SIFT is used in object detection and recognition. The SIFT features are local features, corresponding to regions in the image, and describe the picture that is analyzed using interesting points (keypoints).

The features have also some properties that need to be mentioned. They are invariant to scale and rotation, partial occlusion, partial distortion, partial change of point of view. Compared with other feature-detection algorithms, the features are region-based (local features), are robust and distinctive, and are therefore suited for image matching. For the same reasons, SIFT is also used in tracking and 3D scene reconstruction. Another important characteristic of SIFT is that it works only on grayscale images (.pgm), colored images have to be converted to grayscale.

3.4.1 Scale-space Extrema Detection with Location and Scale Invariance

The general outline of the algorithm is shown in Figure 3.16 [25]. The SIFT algorithm has three major stages with the purpose of obtaining key points in the image followed by another stage for correct key points localization.

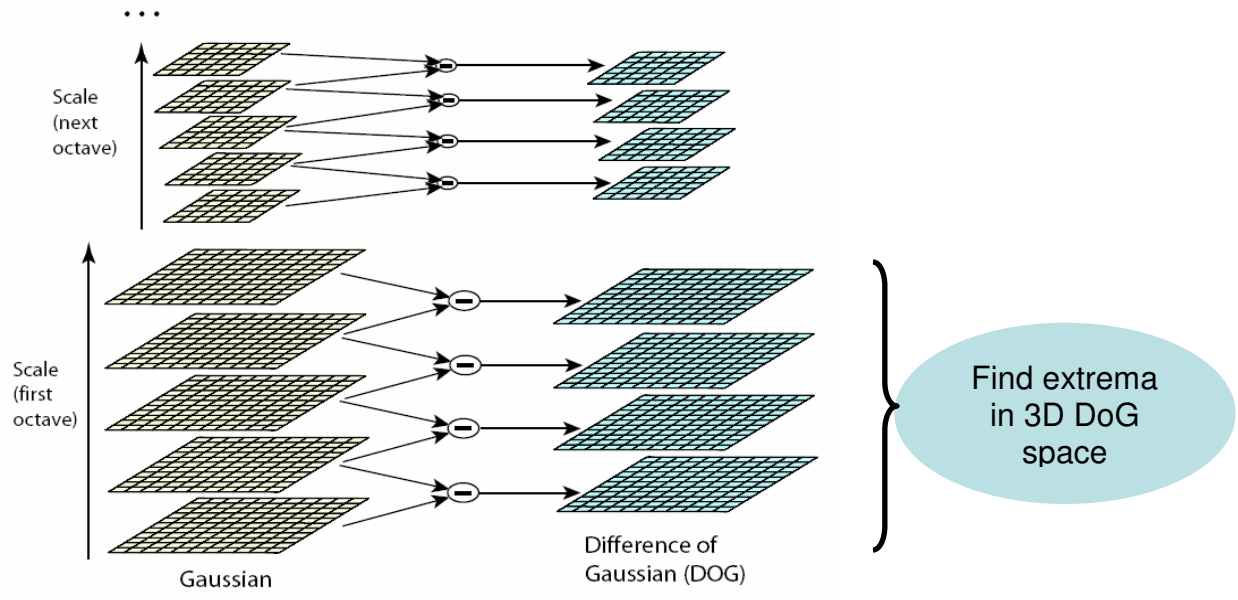


Figure 3.16 General presentation of the SIFT algorithm

The first stage consists in image blurring or convolving with the Gaussian low-pass filter at different scales (Gaussian with different σ). Mathematically this can be written as:

$L(x, y, k\sigma) = G(x, y, k\sigma) * I(x, y),$

where $I(x, y)$ is the original image and $G(x, y, k\sigma)$ is the Gaussian blur filter given by: $G(x, y, k\sigma) = e^{\frac{x^2+y^2}{2k\sigma^2}}$.

After this step the image is down-sampled in order to construct a new octave. The purpose of both smoothing and sub-sampling is the construction of a Gaussian (low-pass) pyramid. The sub-sampling is done by selecting each second pixel on row and column from the previous image. Another approach is to do re-sampling by averaging each four pixels in the image. For the new down-sampled image we redo the blurring as in the previous octave. In this approach, with more than one octave, all sizes must be examined to identify scale-invariant features.

The next step of the algorithm is to compute a Difference of Gaussian (DoG) pyramid. Each image in the pyramid is obtained by taking the difference between two adjacent Gaussian-blurred images which are in the same octave. Mathematically the DoG is given by the following formula:

$D(x, y, \sigma) = L(x, y, k_i\sigma) - L(x, y, k_j\sigma).$

The DoG of an image is viewed as an edge-enhancement filter. The purpose of the DoG is to increase the visibility of edges and other details present in a digital image. The most important feature of the DoG as an edge sharpening filter [32] is that it enhances high frequency detail at edges, while at the same time eliminating noise, unlike other filters that do not work well on noisy inputs.

The main reason for computing the DoG pyramid is to achieve scale invariance. Viewed as the difference of two multivariate normal distributions, the DoG approximates well a second derivate of Gaussian (Laplacian of Gaussian) [32], a well known filter used in blob detection. Blob detection refers to detection of points and/or regions in an image that are either brighter or darker than the surroundings. Figure 3.17 shows a comparison between the Laplacian and the DoG curves, generated with MATLAB. DoG was computed as a difference of sigma = 4.2 and sigma = 4 Gaussians, and Laplacian was computed for sigma = 4.

Mathematically, the DoG can be approximated with a difference of convolutions at nearby scales:

$$\nabla^2 L(x, y; t) = \frac{1}{2\Delta t}(L(x, y; t + \Delta t) - L(x, y; t - \Delta t)),$$

because the scale-space representation $L(x,y,t)$ satisfies the diffusion equation:

$\partial_t L = \frac{1}{2}\nabla^2 L$.

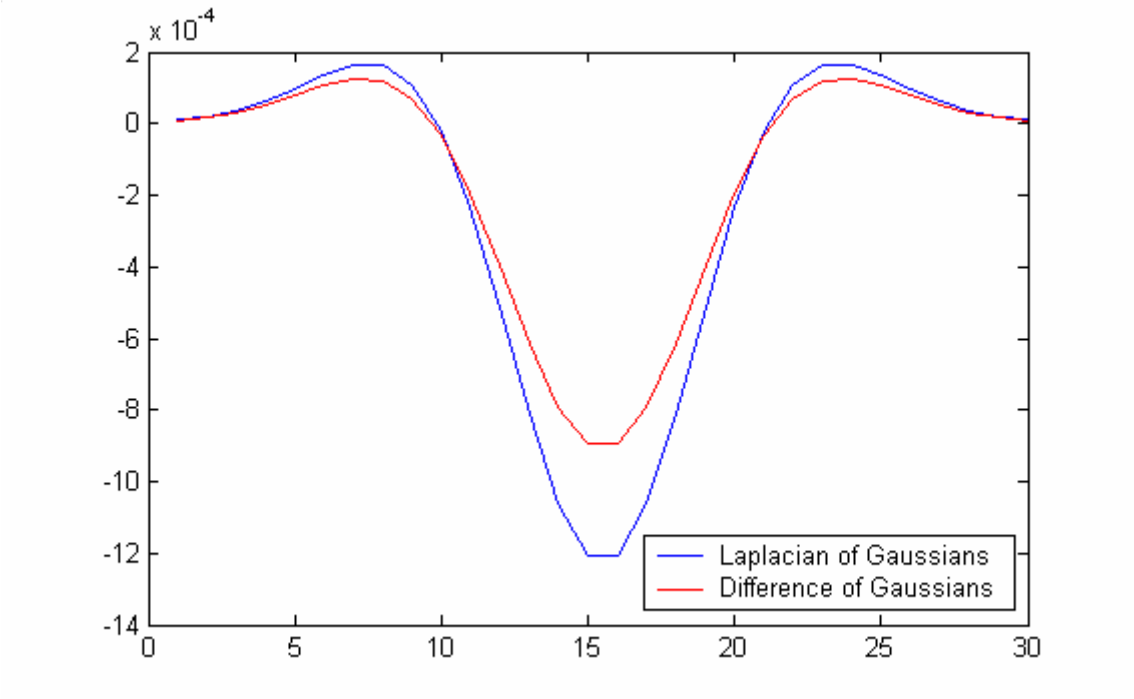


Figure 3.17 Laplacian and DoG filters comparison. Both kernels are invariant to scale and rotation

The next step of the SIFT algorithm is the selection of the spatial maxima and minima in the DoG pyramid. More specifically, each pixel is compared with its neighboring 8 pixels at the same scale and nine corresponding neighboring pixels in each of the neighboring scales. If the pixel value is the maxima or minima among all compared pixels, it is selected as a candidate keypoint (see Figure 3.18 [25]). The main reason behind this keypoint selection method is that maxima and minima are considered fundamental points for the Laplacian filter which is used in blob recognition.

3.4.2 Keypoint Localization

Until now the points of maximum interest have been selected. The problem that arises now is that the extrema detection method produces too many keypoint candidates, some of which are unstable. In order to fix this problem, the algorithm performs a detailed fit to the nearby data for accurate location, scale, and ratio of principal curvatures [25].

In this way, some keypoint candidates that have low contrast and are sensitive to noise or are poorly localized along an edge are rejected. During the best keypoints selection, the algorithm discards low-contrast points. In order to do this the value of the second-order Taylor expansion with a certain offset is used. If this value is less than a given threshold then the candidate keypoint is discarded.

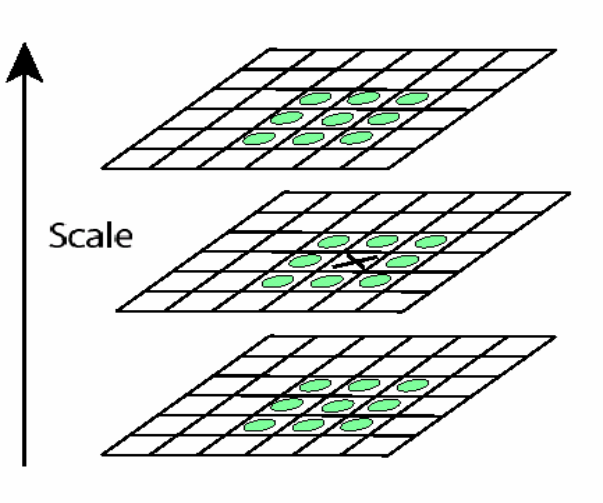


Figure 3.18 Spatial maxima and minima of the DoG selection in SIFT

The next step is the elimination of edge responses. In order to increase the stability of the chosen locations, we need to eliminate the candidates that have poorly determined positions, but have high edge responses. To do that, the SIFT algorithm uses the eigenvalues of the second-order Hessian matrix. The principle behind this approach is that for maxima and minima of the DoG function which are poorly defined, the principal curvature across the edge would be much larger than the principal curvature of those peaks. This method is similar to that of the Harris operator for corner detection [33].

3.4.3 Keypoint Orientation Assignment to Obtain Rotation Invariance

In addition to the construction and selection of the best keypoints, SIFT also tries to assign one or more orientations to the points that have been selected. This step is based on local image gradient directions and is essential in achieving rotation invariance. Each point is described relative to its orientation and therefore achieves invariance to image rotation. The algorithm starts by first Gaussian-smoothing the image $L(x, y, \sigma)$ at the keypoint's scale σ so that all computations are performed in a scale-invariant manner. For an image sample $L(x, y)$ at scale σ, the gradient magnitude, $m(x, y)$ orientation, $\theta(x, y)$, are precomputed using pixel differences [25]:

$$m(x, y) = \sqrt{(L(x+1, y) - L(x-1, y))^2 + (L(x, y+1) - L(x, y-1))^2}$$

$$\theta(x, y) = \tan^{-1}\left(\frac{L(x, y+1) - L(x, y-1)}{L(x+1, y) - L(x-1, y)}\right)$$

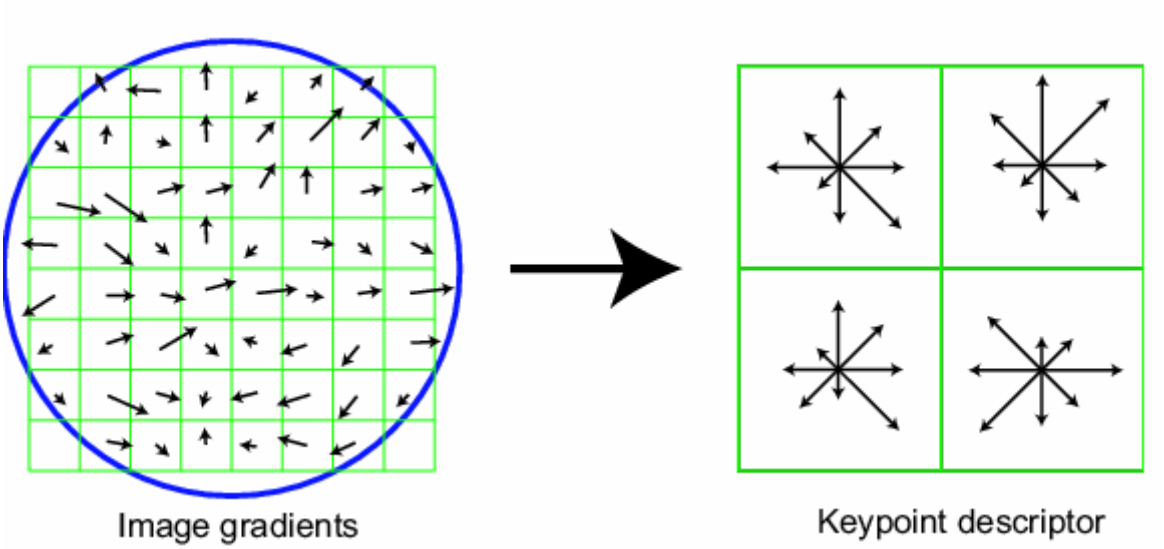


Figure 3.19 Computing the keypoint orientation descriptors based on image gradients

The magnitude and direction computations for the gradient are performed for every pixel in a neighboring region around the keypoint in the image convolved with the Gaussian filter, $L(x, y, \sigma)$. In order to compute the orientation, an orientation histogram with 36 bins is formed The orientation is computed at each 10 degrees, therefore 10 degrees/bin x 36 bins = 360 degrees. Each sample in the neighboring window added to a histogram bin is weighted by its gradient magnitude and by a Gaussian-weighted circular window with a σ that is 1.5 times that of the scale of the keypoint. The peaks in this histogram correspond to dominant orientations. Once the histogram is computed, the orientations corresponding to the highest peak and local peaks that are within 80% of the highest peaks are assigned to the keypoint. In the case of multiple orientations being assigned, an additional keypoint is created having the same location and scale as the original keypoint for each additional orientation (see Figure 3.19 [25] for visual details).

3.4.4 Keypoint Feature Descriptors

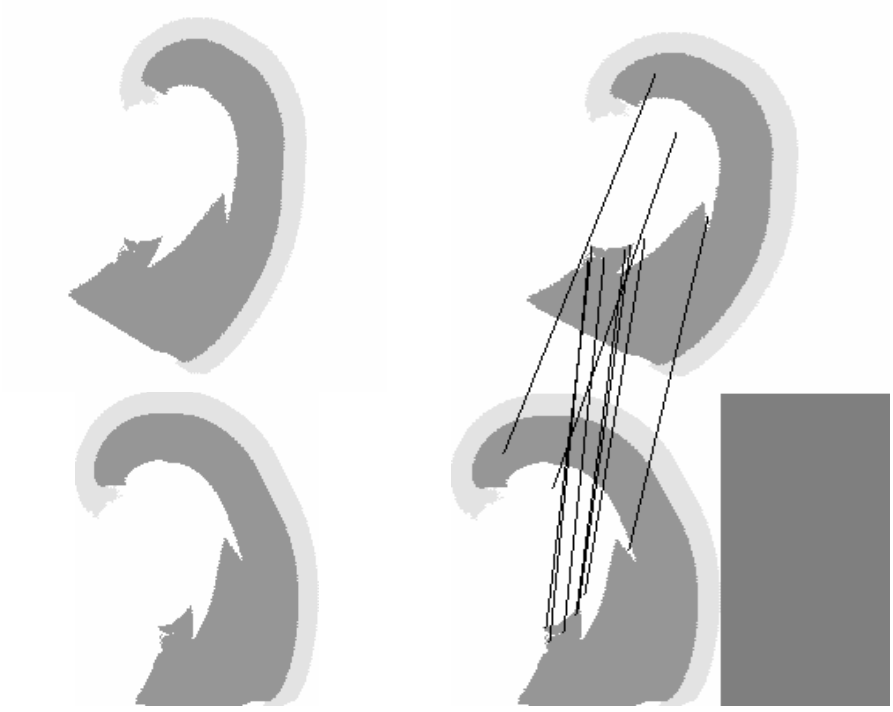
In order to ensure invariance to image location, scale and rotation, SIFT algorithm finds keypoint locations at particular scales and assigns orientations to them. Next, the algorithm tries to achieve partial invariance to the remaining variations, like illumination, 3D viewpoint, etc.

This is done by finding descriptor vectors for the keypoints such that the descriptors are highly distinctive and partially invariant. A feature vector is calculated similarly with the previous steps of the algorithm, as a set of orientation histograms on (4 x 4 = 16) pixel neighborhoods. The orientation histograms are relative to the keypoint orientation, where the orientation data is given by the Gaussian-blurred image that is the closest in scale to the keypoint's scale. For each of the 16 pixel neighborhoods, SIFT computes a histogram with 8 bins, which results in 16 x 8 = 128 elements in the feature vector. The feature vector is then normalized to increase invariance to changes in illumination.

The dimension of the descriptor has been chosen for performance purposes. If the dimension is less than 128, the matching accuracy decreases [25]. On the other hand, if we increase the dimension, the descriptor performance is enhanced, but not by much, and the danger of sensitivity to distortion and occlusion also increases. Furthermore, tests show that SIFT descriptors are invariant to minor affine changes (viewpoint changes up to 50 degrees). In addition, SIFT features are highly distinctive, and provide a high matching accuracy even for very large testing databases.

3.4.5 Object Recognition with SIFT

The best use of SIFT is for matching two different images (for example in Figure 3.20 the matching is done between and image and its rotated corresponding image).



a) b)

Figure 3.20 Image matching: a) the initial image and the image rotated with 30 degrees; b) the matching between the two different images

The matching between two different images is done with the following procedure:

1. Take each keypoint from the first image
2. Find the nearest keypoint from the second image

The metric used to find the best keypoint from the second image is the nearest neighbor metric which is given by the square distance between keypoints' invariant descriptors.

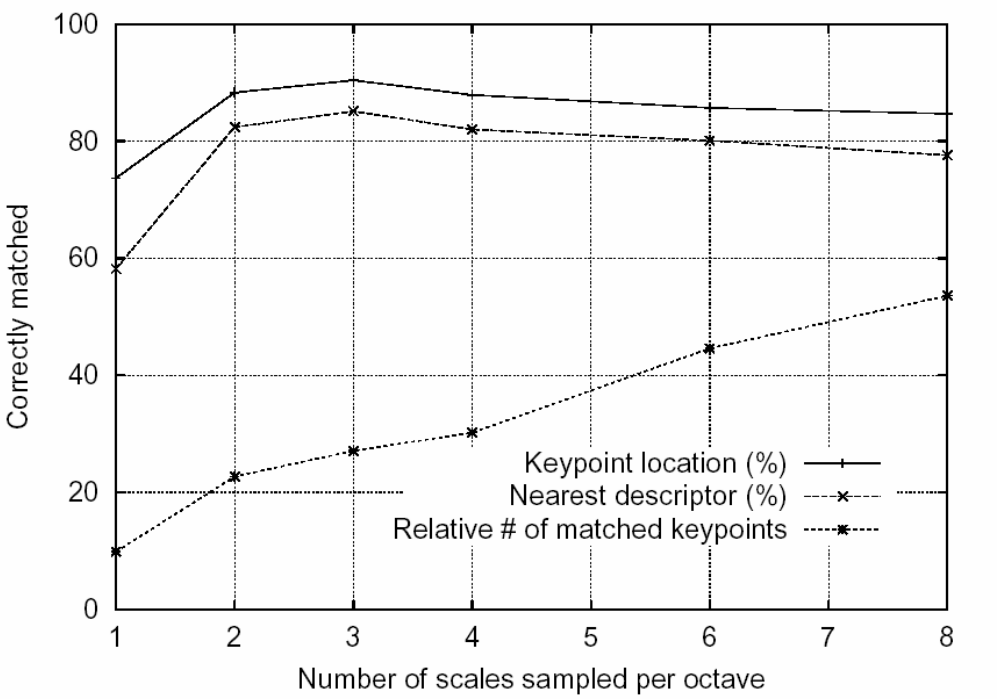


Figure 3.21 Number of scales and number of points correctly matched between two images

An empirical result shows that if the sampling frequency increases, more keypoints are found, but accuracy of matching decreases after 3 scales per octave. This is why usually the SIFT implementations use 3 DoG scales per octave (see Figure 3.21 [25] for details).

Chapter 4. Shape Extraction to Improve Identification

In computer vision, image segmentation refers to the process of partitioning the image in regions which can then be analyzed. In our case, we identify as meaningful regions each shape in the image, and subsequently analyze and classify the shapes to identify spirals or patterns. For images having various pixel intensities, we first apply edge enhancement. Next, we do shape detection and extraction to identify and output each shape in the image.

The problem that we try to solve is a subtype of content-based image recognition [8, 9]. One popular application of content-based image recognition refers to retrieving information about images from the World Wide Web, by improving the existing search engines for finding images [7]. The approaches that attempt to solve this problem apply image processing techniques to identify generic classes of objects and concepts, such as: vehicles, buildings, etc.

4.1 Ensuring Translation Invariance

The main benefit of shape extraction is that it ensures translation invariance. In this way, we look at a shape individually, ignoring its position in the original context. We will see in Chapter 6 that shape extraction improves the results obtained with SIFT, finding more keypoints that match between similar shapes. Also, outliers that previously matched with the surrounding context, do not appear if the shape is first extracting before matching.

4.2 Shape Extraction Algorithm

4.2.1 Edge Enhancement

In image processing, edge enhancement with thresholding is quite a common method for edge detection. One well-known technique based on this method is Canny [26].

We apply the edge enhancement algorithm to detect patterns produced by numerical simulations. Our edge enhancement algorithm uses a Euclidean Distance (ED) metric in the 3-dimensional vector space of the RGB color model.

The Euclidean Distance is defined in the N-dimensional vector space as:

$$D(\vec{v}_1, \vec{v}_2) = \left\| \vec{v}_1 - \vec{v}_2 \right\|.$$

where $\| \ \|$ is the vector norm.

For a 3-dimensional vector space, we can express the Euclidean Distance as:

$$D(\vec{v}_1, \vec{v}_2) = \sqrt{(v_{1,1} - v_{2,1})^2 + (v_{1,2} - v_{2,2})^2 + (v_{1,3} - v_{2,3})^2}.$$

where $\vec{v}_1 = [v_{1,1} v_{1,2} v_{1,3}]^T$ is a color triplet.

The ED metric is sensitive to variations in intensity, and less sensitive to variations in hue and saturation [2]. This characteristic makes it suited for our pattern extraction algorithm, since the patterns that we study have a limited number of hues, although the intensity of pixels can vary. For example, the patterns in Figure 4.1 contain various pixel intensities of the red, green, yellow, and blue colors.

Furthermore, thresholding methods are the basis of many edge segmentation schemes [3], and can be used to eliminate noise, that causes small intensity gradients, and which could mislead the edge tracing process.

For the images in Figure 4.1, we observe that the edges of the shapes are represented by various shades of green, and the edge pixels border a blue-shaded background. Our algorithm performs the edge enhancement by first computing the Euclidean distance between each pixel and the strongest intensity of the green and blue primary colors. Next, we determine if the computed difference on each component color (R, G, and B) falls below a threshold. In case it does, we enhance the pixels to the strongest intensity of their color. In this way, we obtain a uniform color intensity of the edge pixels and of the neighboring background pixels, which makes the edge detection and extraction process easier and less prone to errors.

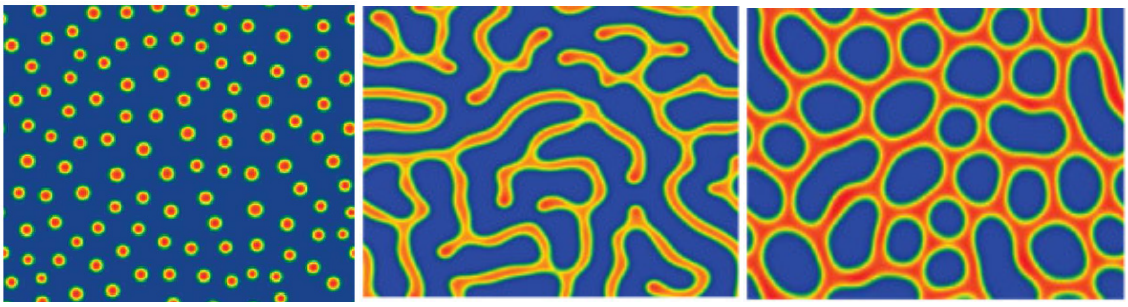


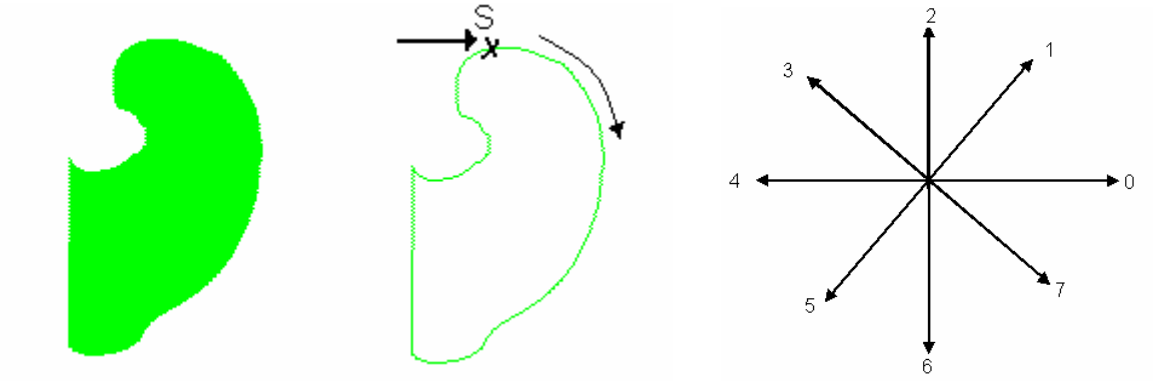
Figure 4.1 Images containing patterns

4.2.2 Contour-tracing with Directions

The contour extraction algorithm is based on the contour-tracing procedure proposed by Haig and Attikiouzel [4,5]. Our method builds upon the original procedure to extract a contour line. We enhance the original algorithm to extract the complete shapes, by identifying all the contour lines that belong to the same shape, not only exterior closed contours.

The core idea of the contour-tracing algorithm is to determine a potential start point and iteratively add new pixels to the contour, by looking at the 8 neighboring pixels [4,5] (Figure

4.2). Depending on which of the adjacent pixel is selected at the next step, we advance the contour in one of the 8 possible directions, as shown in Figure 4.2c.



a) shape b) contour c) next pixel direction

Figure 4.2 Contour-tracing algorithm based on directions

In this section, we describe in detail the implemented algorithm. The pseudo-code uses high-level specification of arrays [6]. An array expression is defined as:

array(i=iexp1..iexp2: exp),

where the elements have the values of the expression exp, and the start and end indexes are given by the values of iexp1 and iexp2.

In order to proceed to the specification of the contour extraction procedure, we will consider that a Contour array has been defined as a global variable for the program.

We do a top-down description of the algorithm, starting from the main function and then detailing the functions that it depends on. The ExtractEdges procedure (Figure 4.3) represents the main function for contour extraction. It begins from a potential start point, verifies that the start point is not isolated, and iteratively adds new points to the Contour array.

```
ExtractEdges(sy,sx)
        = if (P1 := IsolatedPoint(sy, sx)) = nil then return
        else    i := P1[3];  // get the position in the contour
                // get the coordinates and direction of the last pixel added to the contour
                // P is an array with 3 pieces of information: coordinates on x, and y,
                // and direction; first construct and initialize P with P1
                P := array(u1=0..2: P1[u1]);
                // iteratively add one point to the contour
                while   P[0] != sy Λ P[1] != sx do
                        P := EdgePoint(P);
                        AddPoint(P, i); i := i + 1;
```

Figure 4.3 High-level specification of the edge extraction procedure

We test if a point is isolated (Figure 4.4) by trying to find a next contour point which is distinct from the current point. If we end up in the same point, we change the direction hoping to find a new valid, unvisited point in the new direction. We store 3 pieces of information for each pixel in the Contour array, which motivates the "3" multiplication factor in the AddPoint procedure (e.g. Contour[3*i]), to get to the correct index position.

```
IsolatedPoint(s_y,s_x)
        // choose first direction = 6
        = P := array(u_1 = 0..2: if u1 = 0 then s_y else if u_1 = 1 then s_x else 6;
          AddPoint(P, 0); i := 1;                  // add start point to contour
          while i != 3 do
                P := EdgePoint(P);                 // find the next point
                AddPoint(P, i); i := i + 1;        // add point to contour
                // if it is different from the start point it means not isolated, we can stop
                if P[0] != s_y V P[1] != s_x then
                        A := array(u_1 = 0..3: if u_1 < 3 then P[u_1] else i);
                        return A;
         return nil;

AddPoint(P, i) = Contour[3*i] := P[0]; Contour[3*i+1] := P[1]; Contour[3*i+2] := P[2];
```

Figure 4.4 High-level specification for detecting isolated points and adding a new contour point

The EdgePoint procedure (Figure 4.5) looks for a point in a direction received as parameter, and if a contour point has not been found, it changes the direction for a next search attempt. It covers the direction spectrum by looking in 3 directions at a step: direction D-1, D, and D+1. The NextPoint function returns the coordinates of a neighboring pixel, by going one pixel in the specified direction.

```
EdgePoint(P) =
        p_y := P[0]; p_x := P[1]; D := P[2];
        if visited((next := next_point((D-1)%8,p_y,p_x)) = false then
                setVisited(next);
                return array(u_1 = 0..2: if u_1 < 2 then next[u_1] else (D-2)%8);
        else if visited((next := NextPoint(D%8,p_y,p_x)) = false then
                setVisited(next);
                return array(u_1 = 0..2: if u_1 < 2 then next[u_1] else D%8);
        else if visited((next := NextPoint((D+1)%8,p_y,p_x)) = false then
                setVisited(next);
                return array(u_1 = 0..2: if u_1 < 2 then next[u_1] else D%8);
        else return array(u_1 = 0..2: if u_1 < 2 then P[u_1] else (D+1)%8);
```

Figure 4.5 High-level specification for testing which of the 8 neighboring pixels is a valid contour point

4.2.3 Reconstructing the Shape from Contours

While some shapes are bordered by a single contour, such as in Figure 4.2a), others may be formed of several adjacent components, bordered by their own contours. The method to determine contours from the same shape is based on scan line tracing. For an extracted contour (e.g. the exterior contour in Figure 4.7a), we trace the scan lines originating in its pixels. We test the following pre-conditions:

- the Euclidean Distance between the pixels in the first contour and the pixels on the scan line is below an accepted threshold (or, for shapes of uniform color, the pixels in the first contour and the pixels on the scan line should have the same color)
- the scan line intersects another contour, and the Euclidean Distance between the pixels in the first contour and the pixels in the second detected contour is below an accepted threshold (or, for shapes of uniform color, the pixels in the first contour and the pixels in the second should have the same color)

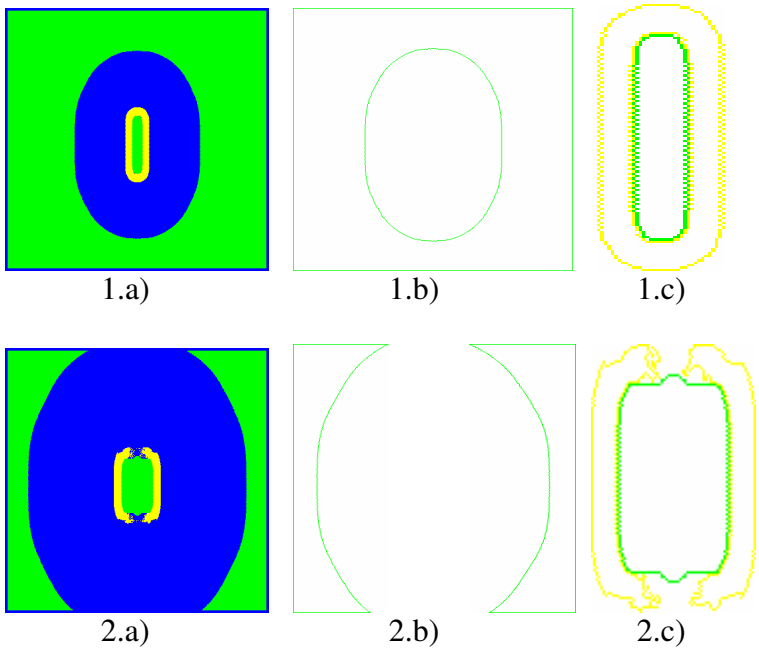
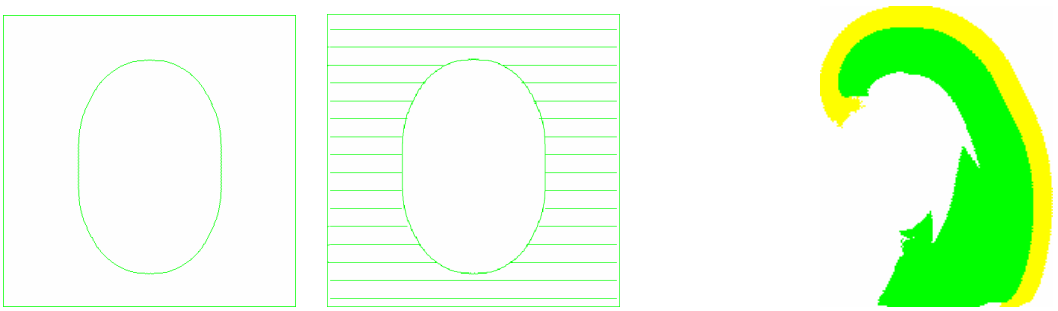


Figure 4.6 Identifying and extracting all the contours that belong to the same shape

Figure 4.6 shows 2 such cases, where a shape is formed of more than one contour. It can be a shape with a hole inside, as in Figure 4.6.1.b), or a shape with several adjacent components, such as image Figure 4.6.2.c).



a) b)

Figure 4.7 a) Scan line tracing; b) Green and yellow adjacent regions, components of the same spiral

In case both conditions are met, we consider that the two contours border the same shape. For pictures that might contain spirals, we determine all adjacent green as well as yellow components as being part of the same shape, since we know that a spiral may contain both green and yellow neighboring regions (Figure 4.7b)).

Chapter 5. Implementation Aspects

In this chapter, we present a few implementation aspects that were considered during the analysis with MATLAB, as well as in the implementation process of shape extraction.

5.1 MATLAB Image Processing Toolbox for Image Analysis, Transform and Display

Image Processing Toolbox™ [35] provides a series of graphical tools for image processing, analysis, visualization. In the following sections, we will describe the main implementation aspects that were used in our experiments. We will focus on image manipulation, the Gaussian filter construction, and the decomposition of signal with Fourier series.

5.1.1 Image Loading and Manipulation

Throughout the experiments, we have worked with grayscale image, in PGM format, which is one of the format types supported by MATLAB.

MATLAB stores the image in a matrix, where each value represents the intensity of the pixel at that particular position. Reading and writing images (from/to files) is done with the functions imread and imwrite. Below we show how to load an image from a file, and store it in a matrix:

```
[Spiral] = imread('out_3_snap0500.pgm');
```

In this way, we have stored the image in the matrix Spiral, where each pixel has a value between 0 and 255, and is stored in the format type uint8. The advantage of this type of data is that it requires little storage memory (1/8 compared to the “double” format). In out experiments, we preferred to work with values between 0 to 1, which required a conversion to double and a mapping in the [0,1] interval:

```
Sp = double(Spiral)/255;
```

Since the image is stored in a matrix, we can have easy access to rows/columns/values in the matrix. For example, when we look at a single row to plot the spatial frequency, we can access it with:

```
y=Sp(100,1:153);
```

which retrieves row 100 from the image. Also, we can display the information stored in a matrix or vector with the “plot” function, which gives a linear graph by default:

```
plot(y);
```

Another function used in displaying graphs is “stem”, which shows discrete values at equally spaced intervals:

```
stem(-M*Omega_o:Omega_o:M*Omega_o, abs(Xarray));
```

In the example above, the interval is “Omega” and the x values range between -M*Omega_o and M*Omega_o.

5.1.2 Filter Design, Gaussian Blurring, and Difference of Gaussians

In order to create a Gaussian filter, we need first to generate the convolution matrix, and then to convolve the image with the Gaussian.

MATLAB offers a special function to construct a convolution matrix:

```
g = fspecial('gaussian',15,2);
```

If the first parameter is “Gaussian”, we obtain a Gaussian convolution matrix. The second parameter specifies the size of the convolution matrix (here is a square matrix with 15 rows and 15 columns), and the third parameter represents the sigma for the filter. The function “conv2” computes the two-dimensional convolution of matrices:

```
gsp = conv2(y,g,'same');
```

where the “same” parameter specifies that conv2 returns the central part of the convolution, of the same size as y.

The Difference of Gaussians is simply computed as the difference of two convolutions:

```
dog = gsp01-gsp0;
```

The two convolutions were obtained by convolving the original image with the Gaussian.

5.1.3 Fourier Series Decomposition with FFT Transform

In order to compute the Fourier transform of a vector X, we use the ‘fft’ function that applies a fast Fourier transform (FFT) algorithm to determine the discrete Fourier transform (DFT) of X:

```
Y = fft(X)
```

Also, we compute the Fourier series approximation for a particular harmonic N1 in a loop like:

```
for i = 2:N1,
```

in order to construct the sum previously defined as:

$$f(x) = \frac{a_0}{2} + \sum_{1}^{N} \left[a_n \cos(nx) + b_n \sin(nx) \right]$$

In the MATLAB code, a_n is the real part of the Fourier transform, and b_n is the imaginary part. An iteration in the loop that computes the sum becomes:

```
fx = fx + (real(2*tmp(i))*cos(Omega_o*(i-1).*x)) -
      (imag(2*tmp(i))*sin(Omega_o*(i-1).*x));
```

where Omega_o represents the fundamental frequency.

5.2 Image Processing Algorithms with Java

The following sections present a few implementation considerations for the Java code. We mainly discuss data structures, and image manipulation aspects determined by image types.

5.2.1 Image Loading and Manipulation

Our implementation loads images in the PPM and PGM formats. A specification of these formats can be found at [36]. Basically, both formats contain a header part followed by the actual pixel information. The PPM files contain color images, where each of the 3 color components (red, green, blue) has a value between 0 and 255. The PGM files can only contain greyscale information, with one value per pixel, instead of 3.On the first line of the file there is a “magic number”, correlated with the image format. For PPM files the magic number can be “P3” for ASCII or “P6” for binary representation, while for PGM images it takes the values “P2” (ASCII) or “P5” (binary). An example of a PPM header is given below:

```
P6
1024 788
# Comment
255
```

Based on the above format specifications, we have implemented processing functions for reading and writing the image files. For binary files, we process the data at byte level. When storing the image pixels as bytes, we might need to do a logic AND to obtain an int value in the interval [0...255]. For example, we can compare two pixel values with the following test:

```
if ((im2&0xff) > (im3&0xff))
```

5.2.2 Data structures

For efficiency reasons, we prefer to use array data structures, rather than Java objects (e.g. Vector). With arrays, the access to elements is faster. For storage of contour pixels (here we refer to position and direction, not intensity), we use an array of integers:

```
int contour_positions[] = new int[height * width * 3];
```

While the contour_positions array described above stores the contour positions, the image (pixel intensities) is stored in a byte array:

```
byte image1[] = new byte[height * width * 3];
```

We store the image in a one-dimensional array, although a 2-dimensional array (matrix) would have worked equally well. The "3" factor in the array size is necessary here to store all the 3 color components (r,g,b) for PPM images. A comprehensive description of the algorithms for shape extraction was presented in Chapter 4.

Chapter 6. Testing and Experimental Results

6.1 Feature-based Identification Algorithms

During the experiments, we analyzed how the original SIFT algorithm works for shape identification. In parallel, we tested how the SIFT algorithm preceded by the shape extraction step works for the same shapes, with the same shape orientation (i.e. rotation angle, scale).

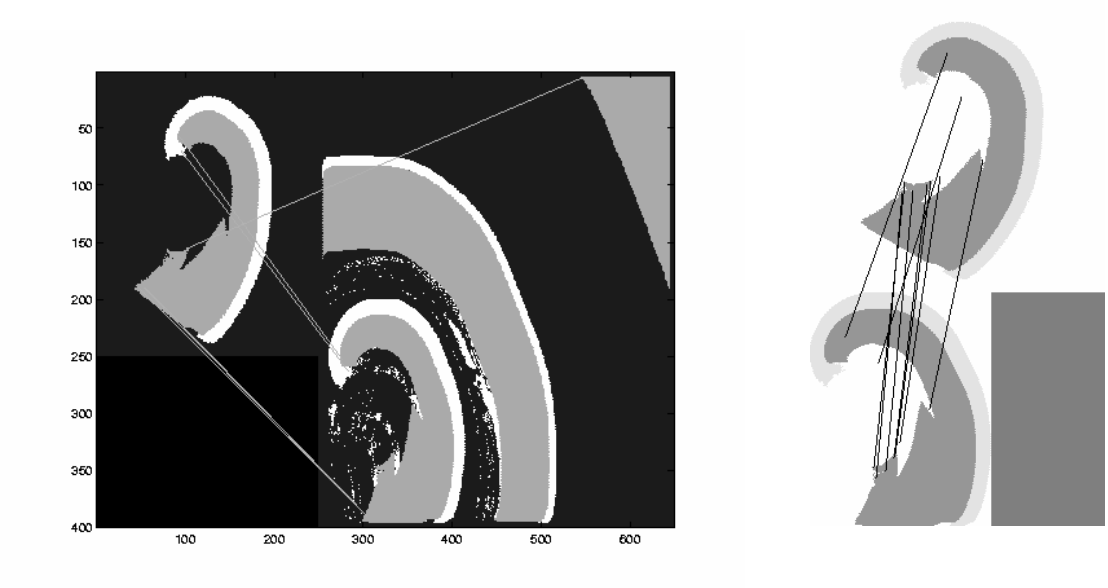


Figure 6.1 Shape identification: SIFT (left) / SIFT with shape extraction (right)

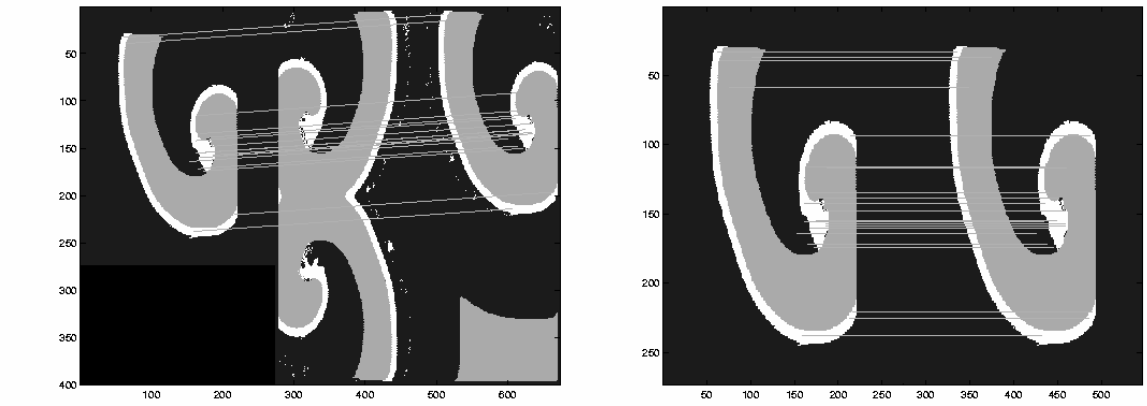


Figure 6.2 Shape identification: SIFT (left) / SIFT with shape extraction (right)

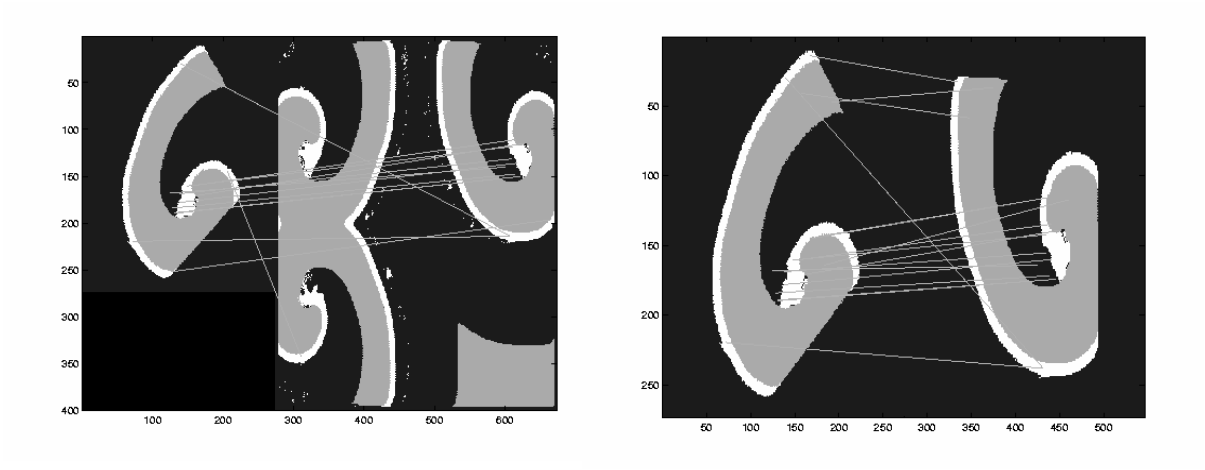


Figure 6.3 Shape identification: SIFT (left) / SIFT with shape extraction (right)

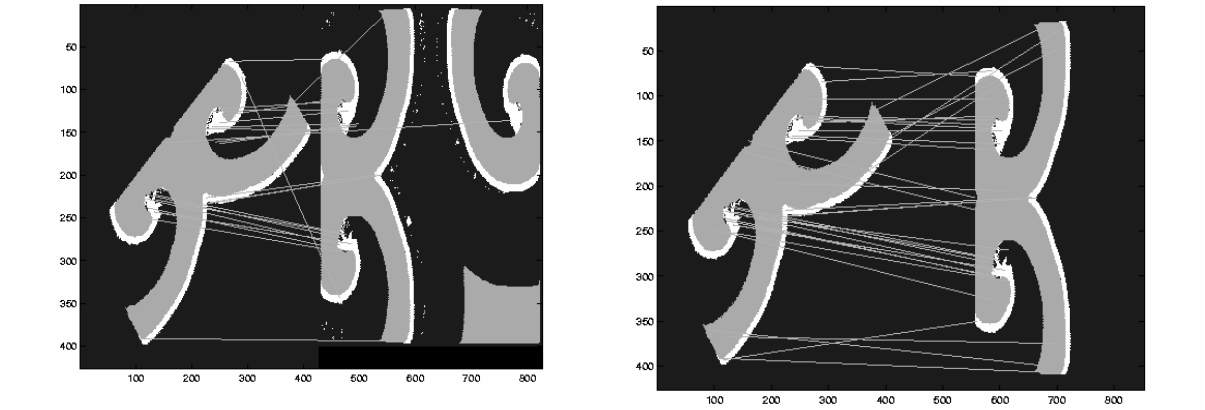


Figure 6.4 Shape identification: SIFT (left) / SIFT with shape extraction (right)

Figure 6.1, Figure 6.2, Figure 6.3, Figure 6.4, and Figure 6.5 show the shape identification results when the shape to be identified is rotated with an angle. As a general result, we see that the number of keypoints found is larger for the second method (SIFT with shape extraction) compared to the first (SIFT).

Furthermore, some of the outliers displayed by SIFT as keypoints do not exist in the second method. We can see such an example, where outliers are eliminated, in Figure 6.1 and Figure 6.4.

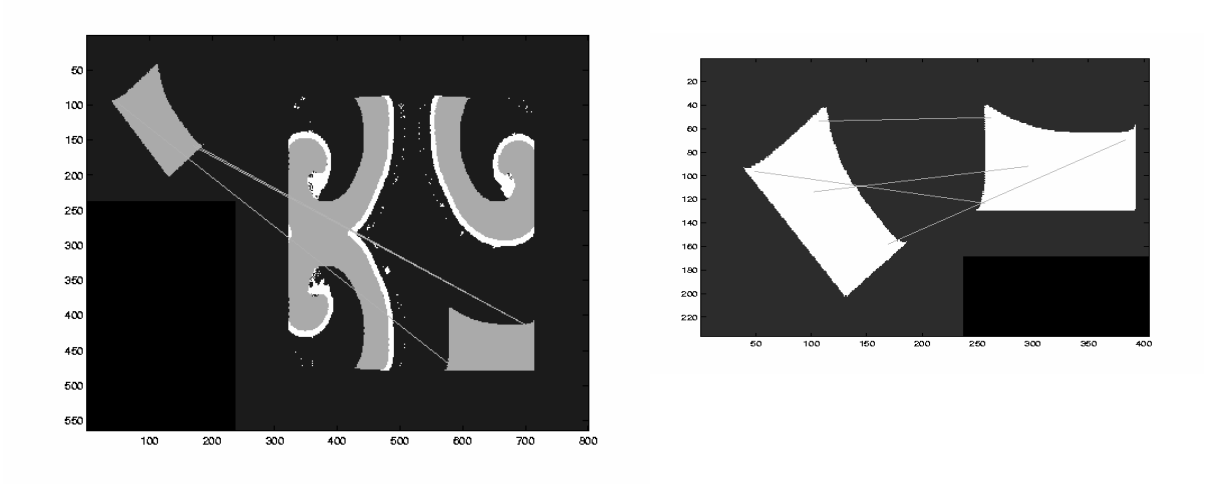


Figure 6.5 Shape identification: SIFT (left) / SIFT with shape extraction (right)

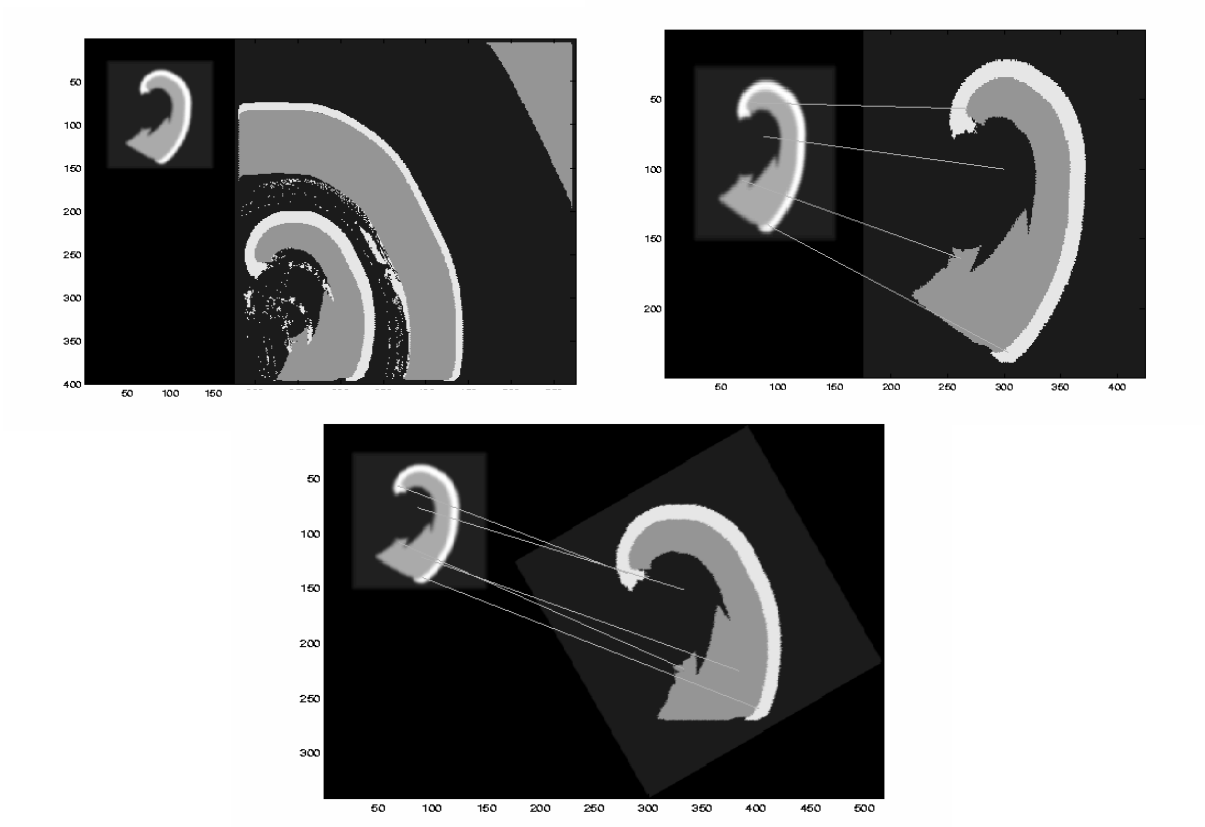


Figure 6.6 Shape identification: SIFT (1st image) / SIFT with shape extraction (2nd and 3rd images)

Figure 6.6 shows how the identification works for a scaled image. In this experiment, we have previously scaled the spiral image to half the original image on width and height. We see that shape extraction improves matching and helps identify some keypoints, while the

original SIFT algorithm does not detect any keypoints.

The next experiment attempts to answer the following question: can we develop a method that uses SIFT as the core algorithm, and a set of labeled images, and is able to recognize a particular class from a set of unlabeled images? In this way, we would have a SIFT-based classifier.

Figure 6.7 illustrates the difficulty of constructing such a classifier (SIFT-based). We test how matching would work between two images, both containing spirals, but which do not belong to the same original spiral shape. SIFT identifies few matching keypoints (2-3 keypoints), most of them being outliers. We observe that there is only one correct match between the two spirals in Figure 6.7 (both left and right images), which is quite insufficient to provide robustness assurances for a classifier. While SIFT responds well to geometric transformations of the same shape, it is not appropriate for matching different shapes, even if they belong to the same class of objects.

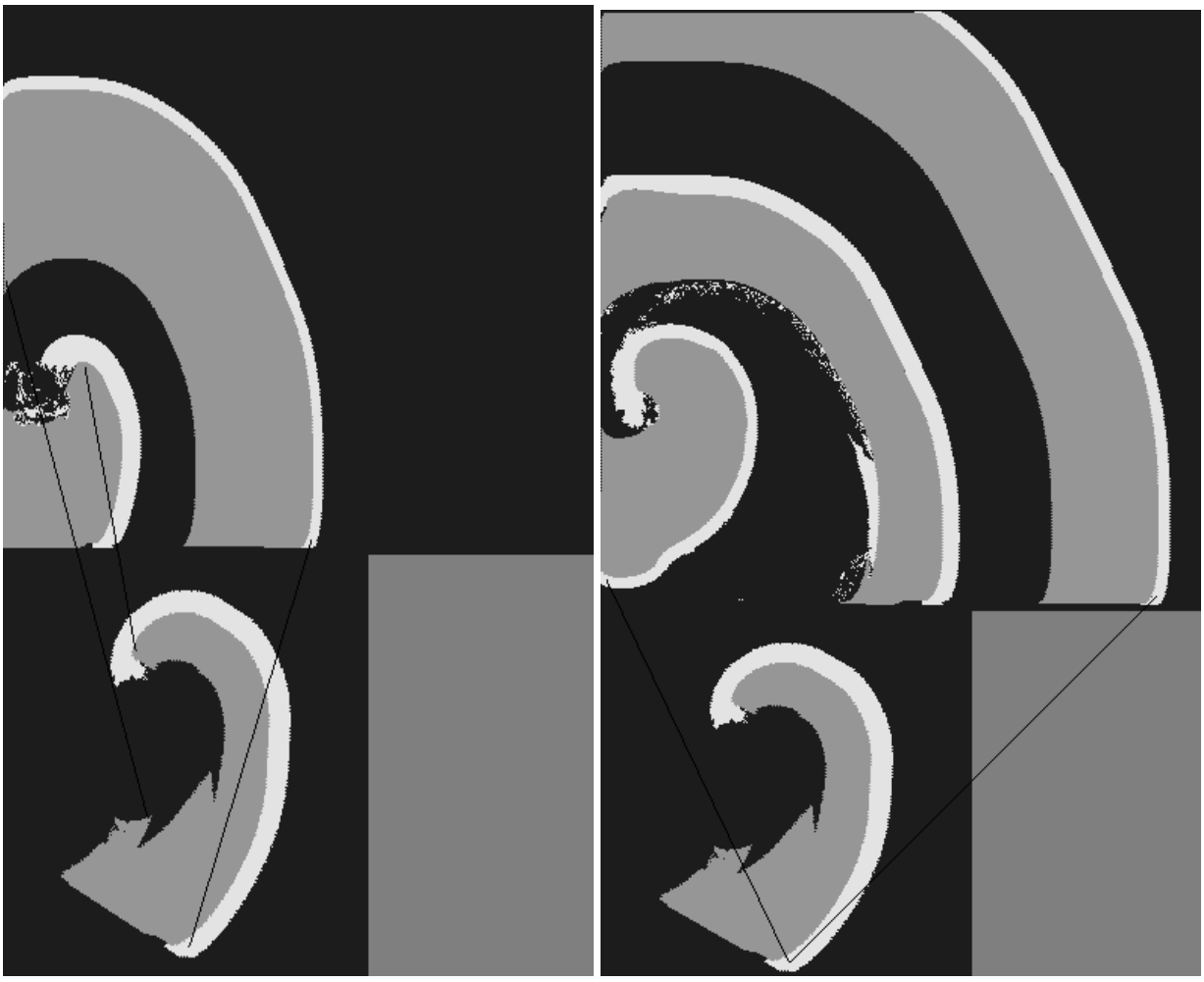


Figure 6.7 SIFT-based matching between different shapes belonging to the same class (spiral class)

6.2 Shape Classification with EMERALD

The following experiments test the EMERALD tool for spiral detection. With EMERALD, we focus on one target class only and classify the images in containing spirals or not. The training phase uses the Weka's decision-tree algorithm to learn a classification rule. The attributes used in classification correspond to the nodes along a path in a superposition quad-tree (SQT). The attribute values represent a mass probability as defined in [1]. The full analysis is carried out for the simulated mode (yellow color).

Using Weka, the following classifier has been learned, for spirals:

```
if (a1 <= 0.061218){
      return false;
}else{
      if (a2 <= 0.325928){
            if (a1 <= 0.156067){
                  if (a4 <= 0.539063){
                        if (a2 <= 0.216309 ){
                              return true;
                        }else{
                              return false;
                        }
                  }else{
                        return true;
                  }
              }else {
                  if (a3 <= 0.499023){
                        if (a0 <= 0.065735){
                              return false;
                        }else{
                              return true;
                        }
                  }else{
                        return true;
                  }
            }
      }else{
            if (a3 <= 0.439453){
                  return false;
            }else{
                  return true;
            }
      }
}
```

Applying the classification rule described above, we compare in Figure 6.8 and Figure 6.9 how the spiral classification works on a full image and on an image where the spiral shape was extracted from the context. The core point is determined with a user click. The left panel in Figure 6.8 presents the attribute values resulted after the quad-tree construction.

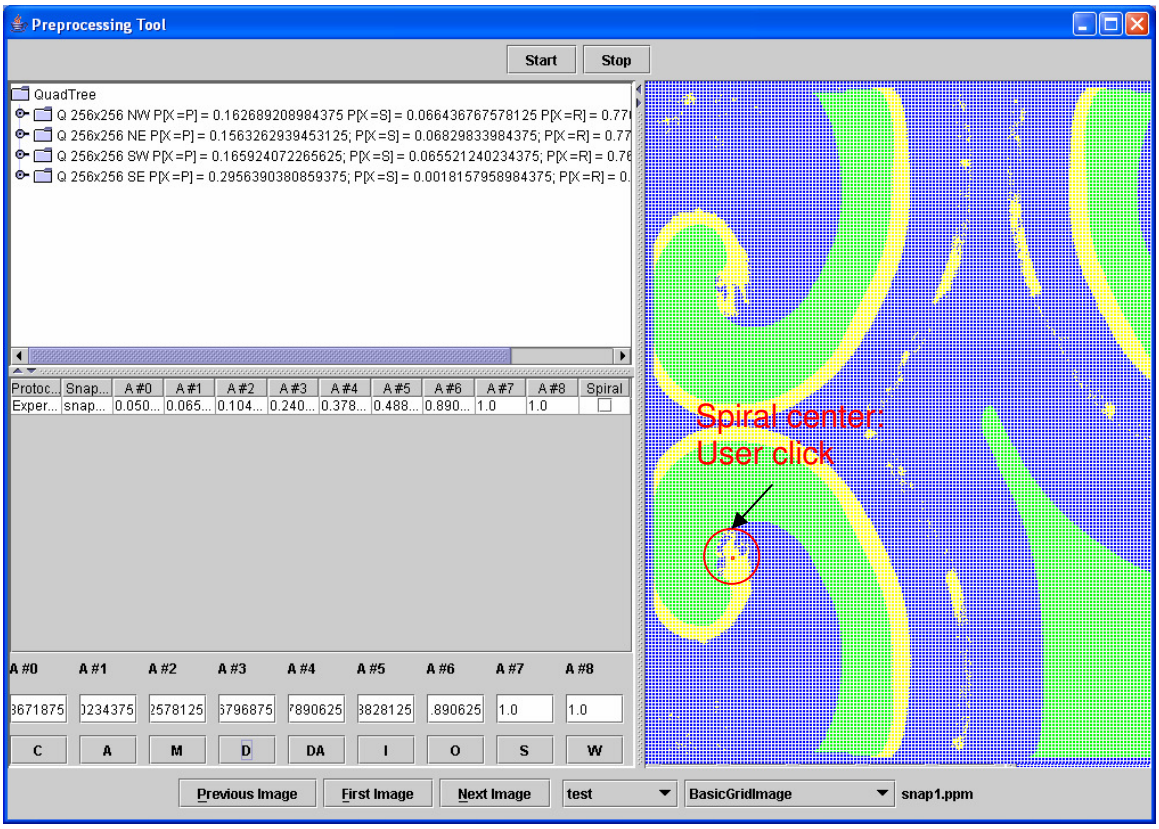


Figure 6.8 Spiral classification for a full image (non-extracted shapes): center selected by user

The correspondence with the attributes in the classification rule above is as follows:

a0 = 0.065521240234375
a1 = 0.1046142578125
a2 = 0.240966796875
a3 = 0.37890625
a4 = 0.48828125
a5 = 0.890625
a6 = 1.0
a7 = 1.0

Based on the classification rule and on the attributes values, the result of the classification for the image in Figure 6.8 is “not spiral”. The result is not correct. We will see that extracting the shapes from the image will improve the classification (Figure 6.9).

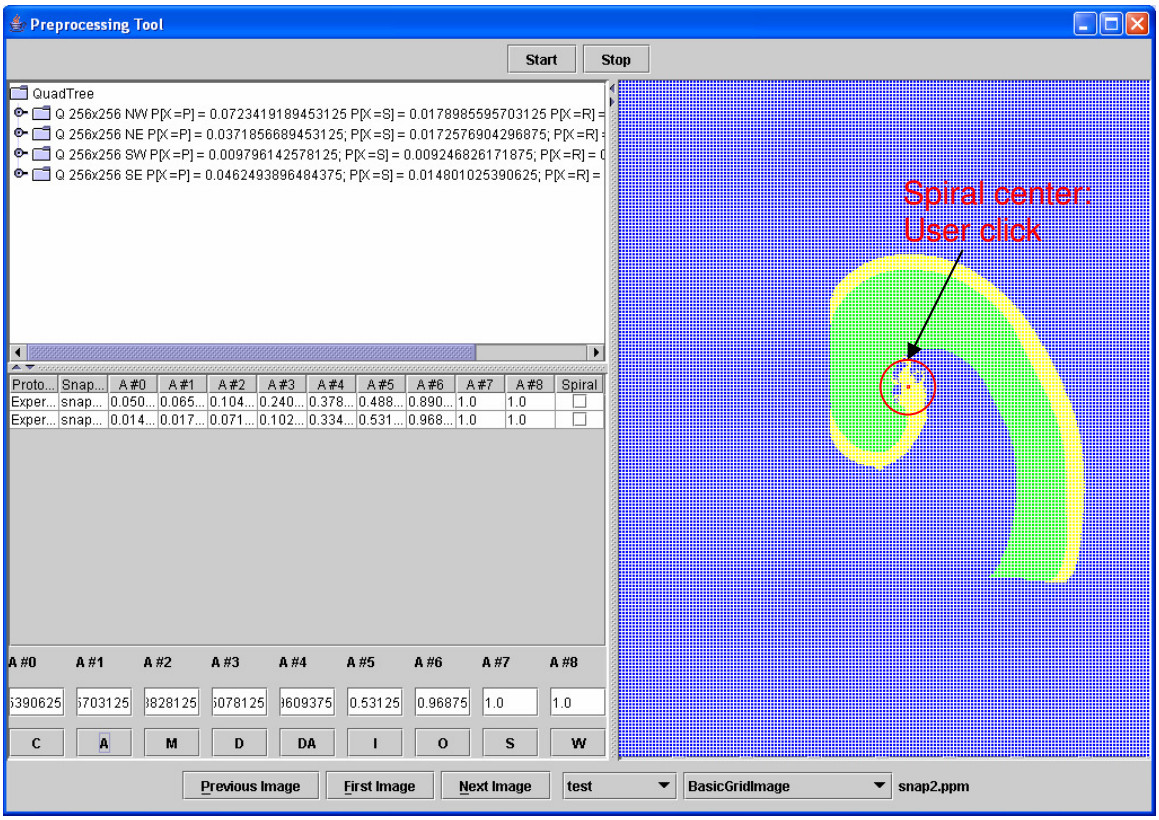


Figure 6.9 Spiral classification for an extracted shape: center selected by user

The attributes determined for the extracted shape in Figure 6.9 have the following values:

a0 = 0.0178985595703125
a1 = 0.07159423828125
a2 = 0.10205078125
a3 = 0.3349609375
a4 = 0.53125
a5 = 0.96875
a6 = 1.0
a7 = 1.0

The classification rule applied on these attributes gives the class “spiral” for the image in Figure 6.9. We observe that in both Figure 6.8 and Figure 6.9 the user clicked on the exact same point belonging to the same shape. However, the classification performed better after the shape had been extracted.

The next experiments (Figure 6.10 and Figure 6.11) eliminate the user effort of selecting a core point. The attributes are determined automatically, as corresponding to the maximum density path for stimulated mode. The maximum density path is determined as follows: we start from the root node (the entire image) and divide it in quads. We select the quad-child with highest density for stimulated mode (yellow color), and recursively divide it

using the same principle.

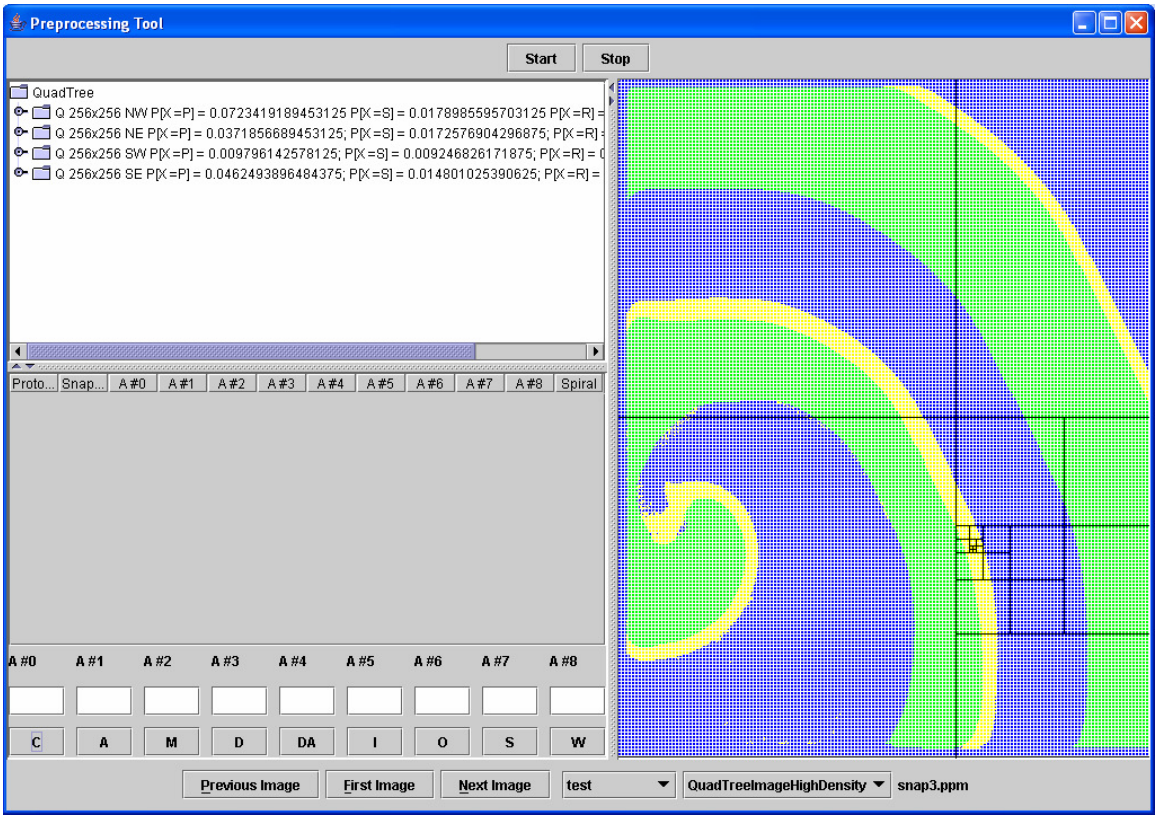


Figure 6.10 Spiral classification for a full image (non-extracted shapes): path selected automatically (highest density path in stimulated mode)

The following attributes values correspond to the path in the quad-tree in Figure 6.10, which gives the class “not spiral”.

a0 = 0.046691895
a1 = 0.05895996
a2 = 0.17285156
a3 = 0.36328125
a4 = 0.7421875
a5 = 0.96875
a6 = 1.0
a7 = 1.0

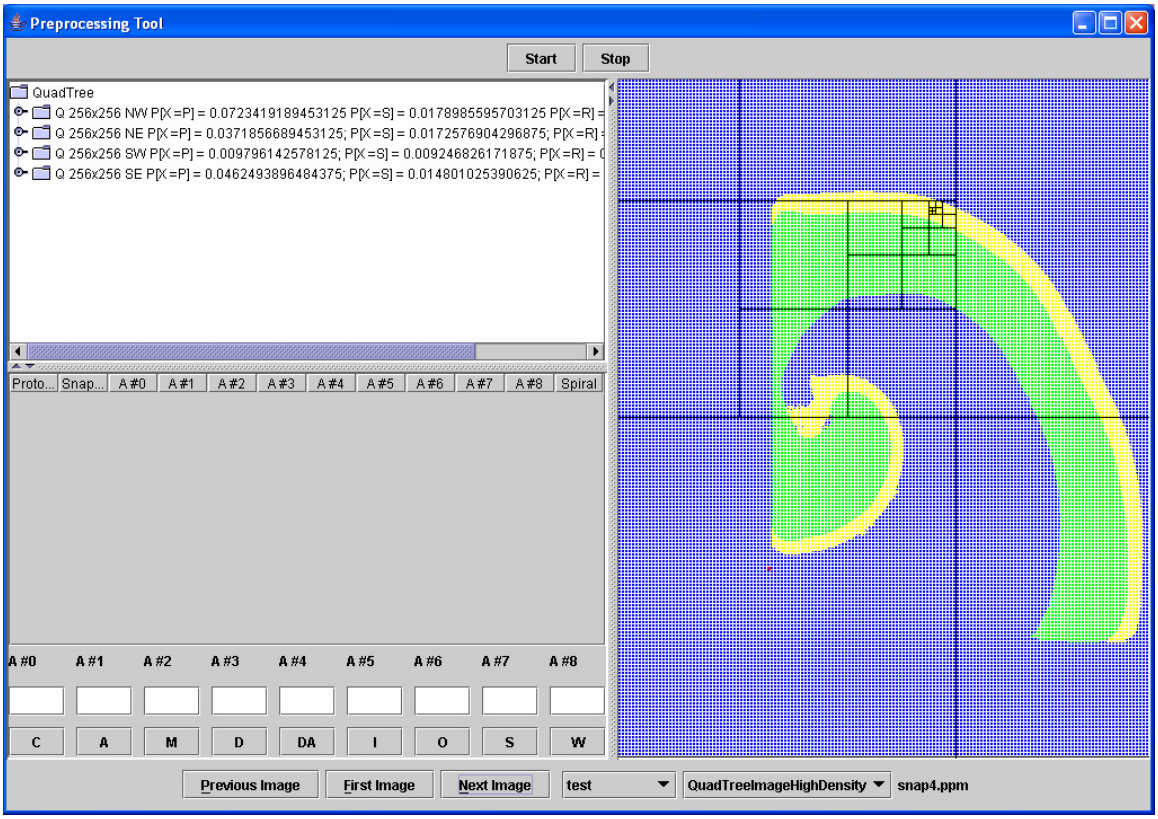


Figure 6.11 Spiral classification for an extracted shape: path selected automatically (highest density path in stimulated mode)

The following attributes values correspond to the path in the quad-tree in Figure 6.11, which gives the class “spiral”.

a0 = 0.03251648
a1 = 0.1048584
a2 = 0.14355469
a3 = 0.34960938
a4 = 0.80859375
a5 = 0.984375
a6 = 1.0
a7 = 1.0

We observe that the strategy of extracting the shapes and performing the classification on independent shapes outperforms the classification on complete images, even when the path in the quad tree is selected automatically, with no user input.

The next test uses a larger batch of test images and preserves the same experimental method: we use highest density path to determine the attributes values and the class of the shape. We use a set of 62 images that were incorrectly classified by the EMERALD tool. We extract the shapes from each of them, and obtain 124 images with single shapes. Next, we run the classifier on the extracted shapes. If we consider the component shapes of an original

image together, as a unit (‘compound virtual image’), 39 of the original 62 incorrectly classified images are now assigned the correct class, which represents approximately 63%. To establish the class of the ‘compound virtual image’, we proceeded as follows: if at least one of the component shapes was classified as ‘spiral’, then we consider the compound image to also have the class ‘spiral’; otherwise (all component shapes were classified as ‘not spiral’), the final class is ‘not spiral’. If we consider each component shape as an independent image, 73 of the 124 independent shapes are classified correctly, which represents 59%. As a result, for our test set, shape extraction eliminates roughly 60% of the incorrectly classified images.

Chapter 7. Conclusions

This thesis addresses the problem of shape identification in medical images. The focus is on identifying shapes that belong to a particular class, specifically spirals, which would indicate fibrillations in the cardiac tissues. We analyze two methods for shape identification: the Scale-Invariant Feature Transform algorithm, which detects and matches interesting features in images while preserving invariance constraints for scaling, translation, rotation, and a learning-based method with the EMERALD tool, which uses superposition quad-trees to detect the onset of spirals.

Each of the two methods has its particular characteristics. The SIFT algorithm is similar to a black box: we use it to discover interesting features in images, without knowing what the features are. Moreover, we apply it on general images without knowing any helpful characteristics of the images. On the other hand, our learning strategy with EMERALD is like a white box when talking about shape identification. We know what we are looking for (we have information about the shape that we are trying to detect in an image, e.g. a spiral). Also, we know some helpful characteristics of the images: they show a continuous evolution of our system, in time and space, until the point where we can have an image with a spiral.

We also observe that a preprocessing phase in which the shapes are extracted from the original images, before the actual identification phase brings a significant improvement for both the feature-based identification method (SIFT) and the learning-based method (EMERALD).

Moreover, our results indicate that SIFT performs poorly when trying to identify shapes belonging to the same class. SIFT responds well to rigid transformations of the *same* shape. However, if we try to match two *different* spirals, insufficient matching keypoints are found. For our particular problem, where the shape of the spiral changes significantly in time, SIFT is less appropriate than the EMERALD learning-based method.

Bibliography

[1] Radu Grosu, Ezio Bartocci, Flavio Corradini, Emilia Entcheva, Scott A. Smolka, Anita Wasilewska. Learning and Detecting Emergent Behavior in Networks of Cardiac Myocytes, *11th International Conference on Hybrid Systems: Computation and Control (HSCC'08)*, LNCS, 2008.

[2] R.D. Dony and S. Wesolkowski, Edge Detection on Color Images Using RGB Vector Angle, Proc. IEEE CCECE'99, Edmonton, Canada.

[3] S.S. Iyengar, Yuyan Wu, Hla Min, Efficient edge extraction of images by directional tracing, *Third International Conference on High-Performance Computing (HiPC '96)*, p. 233, 1996

[4] T. D. Haig and Y. Attikiouzel, An Improved Algorithm for Border Following of Binary Images, *IEE European Conference on Circuit Theory and Design*, 1989, 118-122.

[5] T. D. Haig, Y. Attikiouzel, and M. D. Alder, Border Following: New Definition Gives Improved Borders, *IEE Proceedings-I,* 139(1992) 206-211.

[6] Yanhong A. Liu Scott D. Stoller Ning Li Tom Rothamel, Optimizing Aggregate Array Computations in Loops, *ACM Transactions on Programming Languages and Systems (TOPLAS),* Vol. 27 , Issue 1, pp. 91 - 125, 2005.

[7] Y. Li and L. G. Shapiro, "Object Recognition for Content-Based Image Retrieval," in *Lecture Notes in Computer Science*, Springer-Verlag, 2004.

[8] Stefan Rolfes, Maria- Joao Rendas, Shape Recognition: a Fuzzy Approach, *Proceedings of the 1998 IEEE International Conference on Robotics & Automation*, Leuven, Belgium, May 1998

[9] Beatrice Lazzerini and Francesco Marcelloni, A Fuzzy Approach to 2-D Shape Recognition, *IEEE Tranactions on fuzzy systems,* vol. 9, no. 1, February 2001.

[10] R.C. Veltkamp and M. Hagedoorn, State of the Art in Shape Matching, Technical Report UU-CS-1999-27, Utrecht, 1999.

[11] M. Hagedoorn, Pattern Matching Using Similarity Measures, PhD Thesis, Universiteit Utrecht, 2000.

[12] T. Cootes, D. Cooper, C. Taylor, J. Graham, Active Shape Models – Their Training and Application, *Computer Vison and Image Understanding (CVIU)*, vol. 61, no. 1, pp. 38-59, 1995.

[13] M. Lades, C. Vorbuggen, J. Buhmann, J. Lange, C. von der Malsburg, R. Wurtz, and W. Konen, Distorsion Invariant Object Recognition in the Dynamic Link Archiecture, *IEEE Trans. Computers,* vol. 42, no. 3, pp. 300-311, 1993.

[14] T. Vetter, M.J. Jones, and T. Poggio, A Bootstraping Algorithm for Learning Linear Models of Object Classes, *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* pp. 40-46, 1997.

[15] A. Torsello and E. R. Hancock, A skeletal measure of 2D shape similarity, *Computer Vision and Image Understanding,* vol. 95, Issue 1, pp. 1 – 29, 2004.

[16] K. Siddiqi, S. Bouix, A. Tannenbaum, S. Zucker, The Hamilton-Jacobi Skeleton, *International Conference on Computer Vision (ICCV'99)*, 1999.

[17] B.B. Kimia, A.R. Tannenbaum, S.W. Zucker, Shapes, shocks, and deformations I, Int. *J. Comput. Vision*, vol. 15, pp. 189–224, 1995.

[18] M. Turk and A. Pentland, Eigenfaces for Recognition, *J. Cognitive Neuroscience*, vol. 3, no.1, pp. 71-96, 1991.

[19] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, Gradient-Based Learning Applied to Document Recognition, *Proc. IEEE*, vol. 86, no. 11, pp. 2278-2324, 1998.

[20] C. Cortes and V. Vapnik, Support Vector Networks, *Machine Learning*, vol. 20, pp. 273-297, 1995.

[21] Zhifeng Li, Xiaoou Tang, Bayesian Face Recognition Using Support Vector Machine and Face Clustering, *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'04)*, vol. 2, pp. 374-380, 2004.

[22] Serge Belongie, Jitendra Malik, Jan Puzicha, Shape Matching and Object Recognition Using Shape Contexts, *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 24, Issue 4, pp. 509-522, 2002.

[23] David G. Lowe, Object Recognition from Local Scale-Invariant Features, *International Conference on Computer Vision (ICCV 1999),* pp. 1150-1157, 1999.

[24] Jim Mutch, David G. Lowe, Multiclass Object Recognition with Sparse, Localized Features, *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2006),* pp. 11-18, 2006.

[25] David G. Lowe, Distinctive Image Features from Scale-Invariant Keypoints, *International Journal of Computer Vision,* vol. 60, Issue 2, pp. 91-110, 2004.

[26] J. Canny, A computational approach to edge detection, *IEEE Transactions on Pattern Analysis and Machine Intelligence,* 8(9):679–698, 1986.

[27] E. Bartocci, F. Corradini, E. Entcheva, R. Grosu, and S. A. Smolka, CellExcite: A tool for simulating in-silico excitable cells, *BMC Bioinformatics*, 2007.

[28] Introduction to Fourier Transform, http://www.cs.unm.edu/~brayer/vision/fourier.html, Accessed July 2008.

[29] Image Transforms – Fourier Transform, http://homepages.inf.ed.ac.uk/rbf/HIPR2/fourier.htm, Accessed July 2008.

[30] Weisstein, Eric W. "Fourier Transform--Gaussian." From *MathWorld*--A Wolfram Web Resource. http://mathworld.wolfram.com/FourierTransformGaussian.html, Accessed July 2008.

[31] Photoshop Elements/High Pass & Gaussian Blur Filters Compared with Step Wedges, http://www.photokaboom.com/photography/learn/Photoshop_Elements/filters/1_high_pass_Gaussian_blur_compared_with_step_wedges.htm, Accessed July 2008.

[32] Difference of Gaussians, Wikipedia, http://en.wikipedia.org/wiki/Difference_of_Gaussians, Accessed July 2008.

[33] C. Harris, M. Stephens, A Combined Corner and Edge Detector, *Alvey88*, pp. 147–152, 1988.

[34] T. Lindeberg. Scale-space theory in computer vision. *Kluwer*, 1994.

[35] Image Processing Toolbox – MATLAB, http://www.mathworks.com/products/image/, Accessed July 2008.

[36] Paul Bourke, PPM / PGM / PBM image files, 1997, Web location: http://local.wasp.uwa.edu.au/~pbourke/dataformats/ppm/, Accessed July 2008.