



Ecole d'ingénieurs et d'architectes de Fribourg
Hochschule für Technik und Architektur Freiburg

BACHELOR THESIS

Bellrecords - Analysis of Historical Sound Recordings

Author:

Romain CRAUSAZ

Supervisor:

Carl HABER

Professors:

Ottar JOHNSEN

Frederic BAPST

Experts:

Noe LUTZ

Daniel FORCHELET

*A thesis submitted in fulfilment of the requirements
for the degree of Bachelor of Science*

realised in the

Lawrence Berkeley National Laboratory, California, USA

August 2013

Hes·so

Haute Ecole Spécialisée
de Suisse occidentale

Fachhochschule Westschweiz

University of Applied Sciences and Arts
Western Switzerland



Declaration of Authorship

I, Romain CRAUSAZ, declare that this thesis titled, 'Bellrecords - Analysis of Historical Sound Recordings' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Abstract

Bellrecords - Analysis of Historical Sound Recordings

by Romain CRAUSAZ

The Smithsonian Institution possesses a collection of records made by Alexander Graham Bell and Charles Sumner Tainter from the end of the 19th century at the Volta Laboratory. The records of this collection are in very bad shape and some are even broken, this precludes mechanical sound reading devices using needles. Hopefully systems developed by Carl Haber's team in the Lawrence Berkeley National Laboratory and by Ottar Johnsen in the College Of Engineering and Architecture of Fribourg Switzerland allow contactless 2D and 3D scanning to extract the sound from a record.

The aim of this Bachelor thesis was to develop new tools and features to improve the audio quality of the extracted file from this collection. Therefore data analysis was performed and new algorithms were developed. The results demonstrated these new implemented tools improved the quality. However further developments have to be proceed to obtain a sufficient audio quality.

Acknowledgements

I would like to thank Carl Haber and Earl Cornell for giving me the opportunity to work on my Bachelor project in their laboratory. Their knowledge, advice and ideas were very helpful to achieve the objectives of the project.

Thanks also to the professors Ottar Johnsen and Frederic Bapst for their availability, ideas and remarks about my work. They also gave me the opportunity to work on this thesis abroad through their contacts with the Lawrence Berkeley National Laboratory.

Thanks to my two experts, Noe Lutz and Daniel Forchelet, for their comments, remarks and support during this project.

The College Of Engineering and Architecture of Fribourg has also to be thanks to give the opportunity to their student to perform their Bachelor thesis in other countries.

Finally a special thanks goes to my friend Keegan Lane for the corrections he made to this documentation.

Contents

Declaration of Authorship	ii
Abstract	iii
Acknowledgements	iv
List of Figures	ix
List of Tables	xiii
1 Introduction	1
1.1 History	1
1.2 Context	2
1.3 Report's structure	3
2 Fundamentals	5
2.1 Discs & Cylinders	5
2.2 Extracting sound from the picture of a recording	7
2.3 Audio	8
2.3.1 Noise	8
2.3.2 Clicks	8
2.4 Blobs	9
3 Objectives	11
3.1 Objectives	11
3.2 Tasks	12
4 Analysis	15
4.1 PRISM Software	15
4.1.1 General information	15
4.1.2 Opening process of a recording	18
4.1.3 Blob Clean function	19
4.2 Convolution in image processing	24
4.2.1 General information	24
4.2.2 Normalization	24
4.2.3 Border pixels	25

4.3	Statistics	26
4.3.1	Standard deviation	26
4.4	Blobs Detection	27
4.4.1	Laplacian of Gaussian	27
4.4.2	Adaptive threshold	29
4.5	Interpolation	30
4.5.1	General information	30
4.5.2	Linear interpolation	30
4.5.3	Polynomial interpolation	31
4.5.4	Spline interpolation	31
4.6	Wav format	33
4.7	Recordings	33
4.7.1	Record 287700	34
4.7.2	Record 287701	34
4.7.3	Record 287881	35
5	Design and implementation	37
5.1	Blobs Clean	37
5.1.1	Blobs detection	37
5.1.2	Blobs correction	49
5.1.3	BlobClean v2 - Implementation	52
5.1.4	GUI	56
5.2	Replacement of muted part with "silence"	57
5.2.1	Algorithm	57
6	Tests and validation	59
6.1	Tests description	59
6.1.1	Replacement of muted part with "silence"	60
6.1.2	Audio extraction	60
6.1.3	Tracking	60
6.2	Results	61
6.2.1	Blobs detect algorithms	66
6.2.2	LoG algorithm	69
6.2.3	Adaptive threshold algorithms	71
6.2.4	Complete blobs cleaning algorithm	76
6.2.5	Replacement of muted part with "silence"	80
6.2.6	Audio extraction	81
6.2.7	Conclusion	87
7	Further developments	89
7.1	Blob detection	89
7.2	Tracking	89
7.3	Replacement of muted part with "silence"	90
7.4	Cracks	90
8	Conclusion	91

A Parameters of the new BlobClean function	93
Bibliography	95

List of Figures

1.1	Edison's Phonograph	1
2.1	Grooves and needle	5
2.2	Vertically cut record	6
2.3	Horizontally cut record	6
2.4	Horizontally and vertically cut records	7
2.5	Clicks	8
2.6	Blobs - dirt and dust particles	9
4.1	System schema and probe	15
4.2	PRISM software	16
4.3	Image structure	17
4.4	Opening process	18
4.5	Blob Clean function	19
4.6	Blob Clean Results	20
4.7	Blob Clean error	21
4.8	Blobs' searching	22
4.9	Convolution in image processing	24
4.10	Border pixels	25
4.11	Standard deviation with a normal distribution	26
4.12	Laplacian of Gaussian cross-section	28
4.13	Laplacian of Gaussian steps	28
4.14	Laplacian of Gaussian kernel	29
4.15	Linear interpolation	30
4.16	Derivative of linear function	31
4.17	Derivative of the spline interpolation	32
5.1	Blobs on record 287881	37
5.2	New blobs detection process	38
5.3	Brightness of lost of focus	39
5.4	Pixels size	40
5.5	LoG kernel with $\sigma_x = 1.6$ and $\sigma_y = 1.6$	41
5.6	LoG kernel with $\sigma_x = 1.6$ and $\sigma_y = 2.5$	41
5.7	Normalized LoG kernel with $\sigma_x = 1.6$ and $\sigma_y = 2.5$	42
5.8	Convolved picture	42
5.9	Enhanced blobs	43
5.10	Error of the current blobs detection algorithm	43
5.11	Groove information	44

5.12	High-pass kernel	44
5.13	Input of the high-pass filter	45
5.14	Output of the high-pass filter	45
5.15	Averaging kernel	46
5.16	HP-Filter - absolute values - averaging filter output	46
5.17	Pixels took into account for the adaptive threshold	47
5.18	Black & white image created by the blob detection algorithm	49
5.19	Results of the spline interpolation	50
5.20	Gap Class	50
5.21	Relation between the points and gap	51
5.22	Complete blob clean process	52
5.23	BlobDetect class	53
5.24	BlobClean class	53
5.25	Relation between figure 5.21 and cubic interpolation function	54
5.26	Interpolation - T1 parameter	55
5.27	BlobClean v2 GUI	56
5.28	BlobClean v2 - More parameters	56
5.29	Replacement of muted part with "silent" algorithm	57
5.30	Find the longest silent part algorithm	58
5.31	Audio correction - GUI	58
6.1	Interactive tracking	60
6.2	Results of LoG filter with $\sigma_x = 1.6$ and $\sigma_y = 2.8$	61
6.3	Results of LoG filter with $\sigma_x = 2.8$ and $\sigma_y = 1.6$	62
6.4	Results of LoG filter with $\sigma_x = 1.0$ and $\sigma_y = 1.0$	63
6.5	Results of LoG filter with $\sigma_x = 1.6$ and $\sigma_y = 1.6$	64
6.6	Results of LoG filter with $\sigma_x = 2.8$ and $\sigma_y = 2.8$	65
6.7	BRI Algorithm - lost of focus detection	66
6.8	BRI Algorithm - cracks detection	67
6.9	BRI Algorithm - too sensitive threshold	68
6.10	LoG Algorithm - Too sensitive threshold	69
6.11	LoG Algorithm - Threshold = 2.8	70
6.12	Adaptive threshold Algorithm (value)	71
6.13	Adaptive threshold Algorithm (value) - Cracks	72
6.14	Adaptive threshold Algorithm (value) - Lost of focus	73
6.15	Adaptive threshold Algorithm (value) - Computation time for 3 and 6 rings	73
6.16	Adaptive threshold Algorithm (derivative)	74
6.17	Adaptive threshold Algorithms	75
6.18	Blob Clean v2 result	76
6.19	Blob Clean v2 result 2	77
6.20	Blob Clean v2 result 2 - graphic	77
6.21	Blob Clean v2 result 3 - undetected blobs	78
6.22	Record 287881 - Replacement of muted part with "silence"	80
6.23	Records 287700, 287701, 287860.2, 287881	81
6.24	Record 287700 - cracks	82
6.25	Record 287700 - waveforms	82
6.26	Record 287700 - frequency spectrum	83

6.27	Record 287701 - waveforms	84
6.28	Record 287701 - frequencies' spectrum	84
6.29	Record 287881 - waveforms	85
6.30	Record 287881 - frequency spectrum	85
6.31	Record 287881 - zoom, remaining clicks	86
A.1	BlobClean v2 GUI	93
A.2	BlobClean v2 - More parameters	94

List of Tables

4.1	Blob Clean function's parameters	19
4.2	getTresh() parameters	21
4.3	setTresh() parameters	23
4.4	wav format - some definitions	33
4.5	Record 287700	34
4.6	Record 287701	34
4.7	Record 287881	35
6.1	Computation time	79
A.1	Blob Clean v2 function's parameters	94
A.2	Blob Clean v2 function's parameters 2	94

Chapter 1

Introduction

1.1 History

In 1860, Leon Scott, a French printer and bookseller, who lived in Paris, invented the first known sound recording device. His machine, called the “Phonautograph”, created a visual image of the sound on a paper sheet but didn’t have the ability to playback his recordings. He recorded himself singing “Au Clair De La Lune”. In 1877, Thomas Edison invented the “Phonograph” (see figure 1.1), the first device able to record and reproduce sound. He used cylinders and tinfoil sheets to record the sound. However, the “Phonograph” wasn’t efficient. The tinfoil tears easily and even with a properly adjusted stylus, the reproduction was distorted, squeaky and only good for a few playbacks. Thomas Edison couldn’t improve the quality of his phonograph, because he had to spend the next five years developing the New York City electric light and power system due to an agreement. However he had discovered the secret of sound recording.

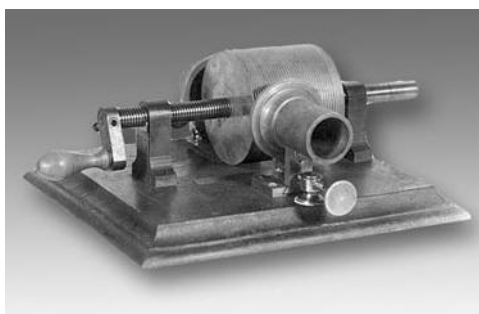


FIGURE 1.1: Edison’s Phonograph [1]

Gardiner Greene Hubbard, director of the Edison Speaking Phonograph & Co and owner of phonograph's patent, convinced Alexander Graham Bell, his son-in-law, to improve the Edison's invention because nobody wanted to buy a machine that wasn't efficient. In 1881, A. G. Bell and Charles Sumner Tainter invented the first version of the "Graphophone", an improvement of the Edison's phonograph, at their Laboratory in Georgetown (Washington D.C.), the Volta Laboratory and Bureau. Eventually they used wax-coated cardboard cylinders instead of tinfoil to record the sound. In the process they made a lot of other experiments with that machine leading to different prototypes. They were able to record sound on discs and cylinders using both lateral and "hill-and-dale" (vertical) cutting types. All the experimentation and recordings were given to the U.S. Smithsonian Institution [2, 3]

1.2 Context

The Smithsonian Institution contacted the Lawrence Berkeley National Laboratory to digitize the recordings made by Alexander Graham Bell at the Volta Laboratory. This collection, known as the oldest reproducible records' collection preserved in the world, contains more than 200 recordings, which have been created at the end of the 19th century. Most of them are in very bad shape or even broken. This precludes the mechanical sound reading devices using needles [4].

Dr Haber's team developed a system at Lawrence Berkeley National Laboratory to extract the sound from the recordings using contactless 2D and 3D scanning. They obtained very good results with a lot of different recordings. They were able to extract the sound from some recordings of the Alexander Graham Bell's collection, however the noise level is high due to the damage. The tools they developed aren't specifically adapted to this collection due to the bad shape of the recordings.

This diploma work aims to develop new analysis tools and features, which will be added to the existing project, to be able to extract the sound from this collection with a improved quality. To achieve this, 3 samples were taken from it. These samples will be analyzed and experiments will be run in order to develop these tools. The project will focus on the 3D system because it gives more information than the 2D and is more suited to this kind of bad shape recordings.

1.3 Report's structure

The documentation is structured in 8 chapters starting with an introduction about the early sound recordings. The second part presents the basic knowledge in order to understand this project. The third part is a presentation of the objectives and tasks that has to be reached. The fourth part regroups all the analysis of the subjects that were necessary during this project. The fifth part presents the design and implementation of the different developed algorithms. The sixth part regroups the tests' definition and results with analysis and comparison of these. The seventh part presents an analysis of possible further improvements and finally the last chapter is the conclusion of the thesis.

An appendix, explaining the new parameters, is at the end of this document. The audio files, weekly reports, meetings' invitations and minutes can be found in the CD or zip file attached to this document.

Chapter 2

Fundamentals

2.1 Discs & Cylinders

The discs, originally called Gramophone records are an analog sound storage medium. They were made in rubber, shellac and finally in vinyl. They replaced the phonograph cylinder and are now replaced by the digital media such as CD (Compact Disc). Both of discs and cylinders stores the sound information in the grooves. A needle follows the grooves (figure 2.1) while the disc or cylinder is turning. The movement of it is used to reproduce the recorded sound.

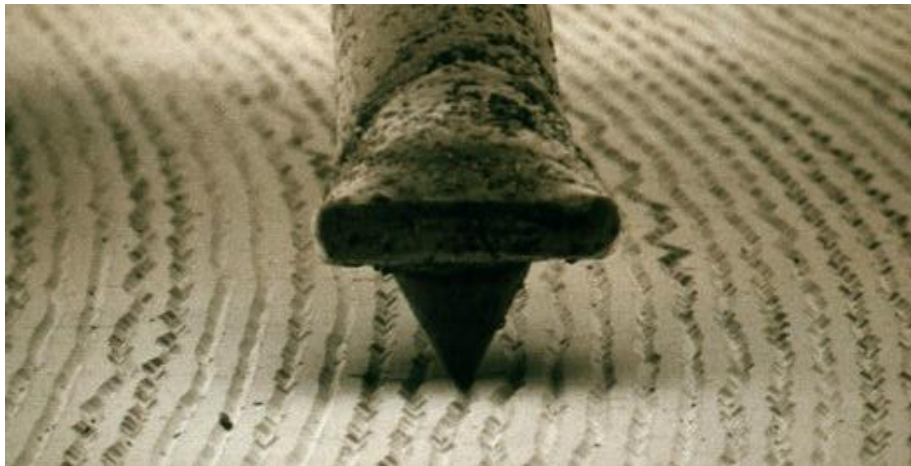


FIGURE 2.1: Grooves and needle [5]

It exists different kind of analog sound reading technique that depends on how the sound were recorded. The oldest one, used in the Edison's Phonograph and in some records of the Volta Laboratory, uses the vertical motion of the stylus to reproduce sound (figure 2.2).

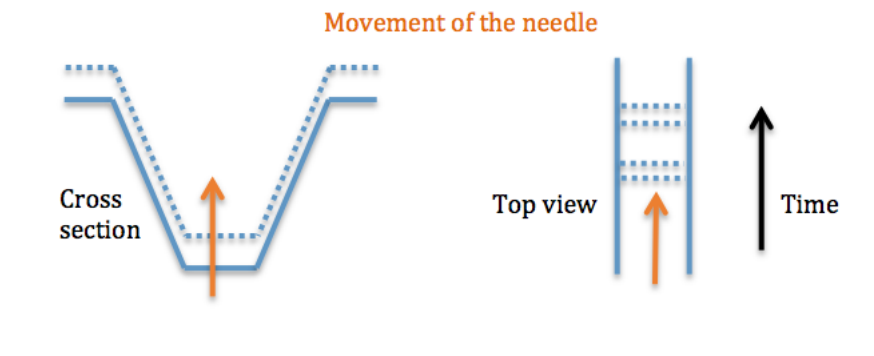


FIGURE 2.2: Vertically cut record

The other one, used in the Gramophone records and that replaced the previous technique, uses the horizontal motion of the stylus (figure 2.3).

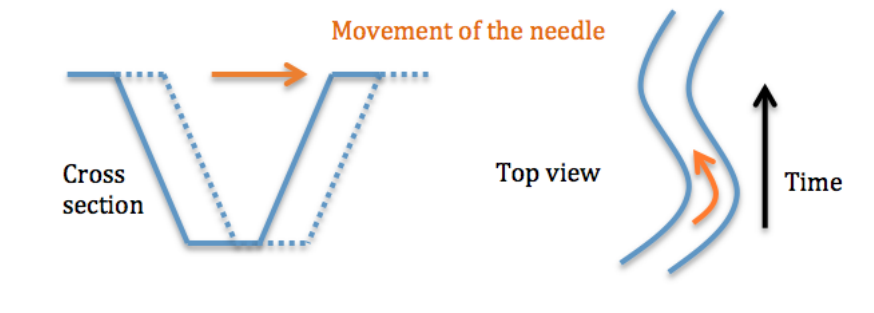


FIGURE 2.3: Horizontally cut record

The figure 2.4 shows two pictures of a record opened with PRISM (see chapter 4). The left one is a horizontally cut record and the right one is a vertically cut one. On those pictures, the gray-scale represents the depth. The deeper a pixel is, the darker it is displayed.

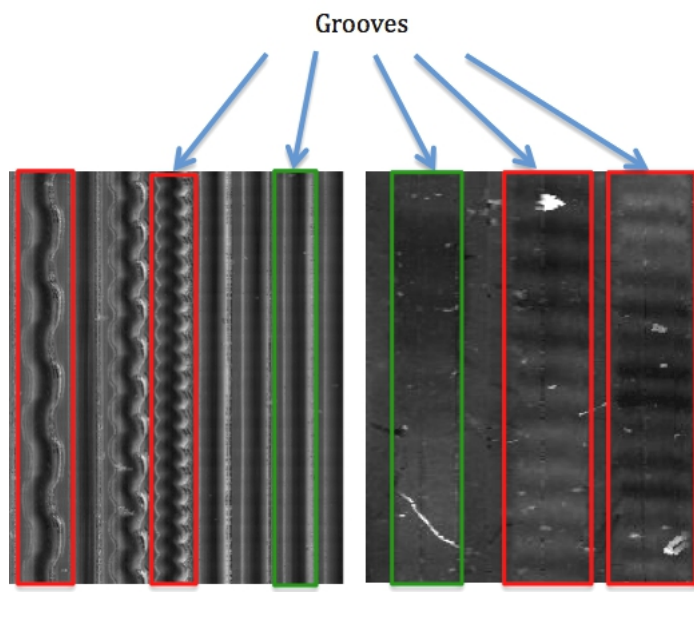


FIGURE 2.4: Horizontally and vertically cut records

In the horizontally cut record, we can see seven parts of grooves and three in the other one. The green squares represent silent part and the red ones represent sound at different frequencies.

2.2 Extracting sound from the picture of a recording

As explained before, the movement of the stylus represents the sound of the record. Knowing this, it is possible to extract the sound from a high resolution picture of a recording by simulating the movement of the needle with an algorithm. The first step consists of tracking the groove and the second one consists of following the tracked points and extracting the sound. Carl Haber's team developed a complete system to create the picture of the recordings and to process it to extract the sound. This project is called RENE 2D or 3D depending on the probe used. An equivalent project called VisualAudio has been realized in the College of Engineering and Architecture of Fribourg Switzerland by Dr. Ottar Johnsen.

This technology opened a lot of opportunities in sound restoration. Previously we were not able to play some of the oldest records of the world because of their condition. With this system it is now possible to play those records without using a mechanical sound reading devices using needles.

To extract the sound from the picture, the software takes the derivative of the groove's points.

2.3 Audio

2.3.1 Noise

The noise refers to the residual low level sound (usually hiss and hum) that can be heard. The noise can come from different device such as the speakers, the converter (A/D or D/A, quantification), or simply by the recording device (which is the case with the oldest recording devices) [6].

2.3.2 Clicks

The clicks refers to very short noise with high amplitude that appear in the sound. The figure 2.5 shows an audio waveform with and without clicks (the white square represents clicks).

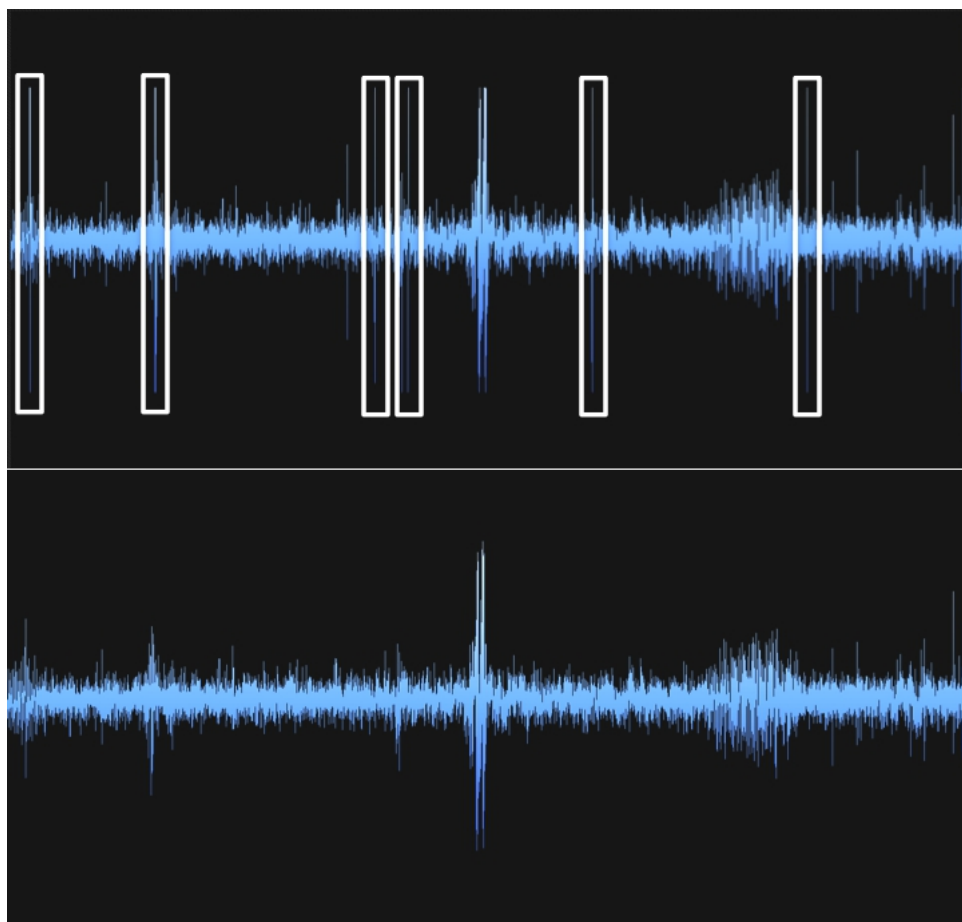


FIGURE 2.5: Clicks

2.4 Blobs

In image processing, the term blobs is used to represent a region of a digital image in which some properties are constant or vary within a prescribed range of values. A blob can represent a person, a face, an object in a picture. The blobs are used to determine region of interest for further processing [7].

In this project, blobs represents all the dust and dirt particles, scratches and cracks present on the recording. The collection of the Volta Laboratory is very old and most of the records are in bad shape and the data taken by Carl Haber's team contains a lot of blobs that creates clicks and noise in the final audio file. The figure 2.6 shows parts of a recording from this collection. All the little white regions are blobs. One of the goal of this project is to develop a new blobs detection function in order to remove those clicks and enhance the audio quality. As explained before, those pictures represents the depth. The dirt and dust particles appear in white because they are over the record.

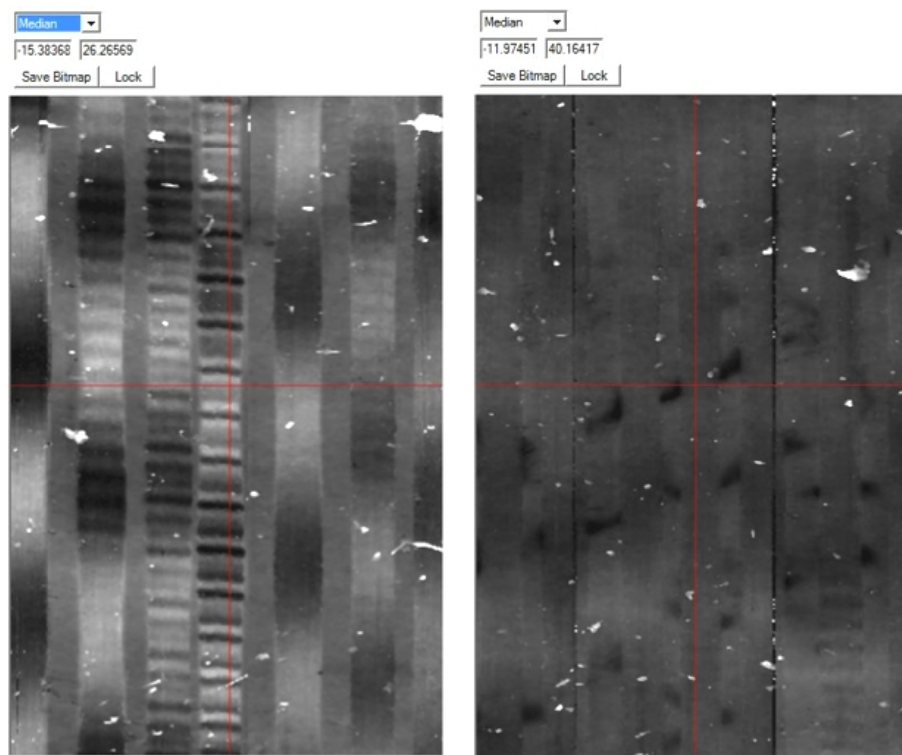


FIGURE 2.6: Blobs - dirt and dust particles

Chapter 3

Objectives

3.1 Objectives

The goal of this project is to develop new tools and features for the current software (PRISM). The first step will be to implement a function, which detects the blobs automatically and interpolates the missing points. As soon as this is functional, the major problem will be solved and, by analyzing the new results, we will be able to find new possible improvements in order to obtain a better sound quality on the recordings from the Volta Laboratory.

First part:

- Analyze early sound recordings.
- Analyze the recordings in order to get an overview of the errors that the present algorithm gives.
- Analyze the interpolation functions in audio reconstitution.
- Develop a new BlobClean function.
- Test this new function.

Second part:

- Replace the muted part with silence.
- Change the Laplacian of Gaussian function.
- Analyze the parameters of the new blob clean function.
- Evaluate the achieved results and compare them with the previous one.

3.2 Tasks

From these objectives, we can extract tasks in order to achieve them:

First part:

- Analyze early sound recordings.
 - Read the book Development of the Phonograph at Alexander Graham Bell's Volta Laboratory, by Leslie J. Newville.
 - Read the first part (Early steps in the history of sound recording) of the Journal Of The Society Of Motion Picture Engineers Vol 48, April 1947.
- Analyze the recordings in order to get an overview of the errors that the current algorithm gives.
- Analyze the interpolation functions in audio reconstitution.
 - Find and read general information about interpolation.
 - Read the book Digital Audio Restoration – A statistical Model Based Approach, by Simon J. Godsill and Peter J.W. Rayner.
- Develop a new BlobClean function.
 - Analyze PRISM's source code
 - Design a new blobs detection function.
 - Design a new interpolation function.
 - Implement these new functions.
- Test the new BlobClean function.

Second part:

- Replace the muted part with silence.
 - Detect a silent part in the audio file.
 - Replace the muted part with the silent values.
- Change the Laplacian of Gaussian function.
 - Create a new Laplacian of Gaussian function that takes two different sigma for x and y and adjust this parameter depending on the position on the recording.
- Analyze the parameters of the new blob clean function.
 - Try to find dependence between parameters in order to reduce the number of it for the user.
 - Analyze the effect of each parameters on the final result.
 - Create a user manual to simplify the use of the new blob clean function.
- Evaluate the achieved results and compare them with the previous one.
 - Create the tracking for the recordings.
 - Extract the audio from the recordings.
 - Compare the results.
 - * With no BlobClean function.
 - * With the previous BlobClean function.
 - * With the new BlobClean function.

Chapter 4

Analysis

4.1 PRISM Software

4.1.1 General information

The figure 4.1 demonstrates the system's concept. The capture system is controlled by the LabView program. The MPLS180 probe (for more information about this probe, refer to [8]) records the data and send them to the LabView program. It capture 180 pixels and use a white light that brakes up into it's spectral components with a lens that has a huge chromatic aberration. Only one wavelength λ is in focus. The depth is a function of λ , $f(\lambda)$. . The labView program generates the data files (.PRI and .BRI). The PRI file contains the depth information and the BRI the brightness information. However PRISM 2010 only uses the .PRI file to process the data and create the wav file. This software includes a lot of image processing, signal processing and tracking functions that help the user to obtain the best quality in the final audio file.

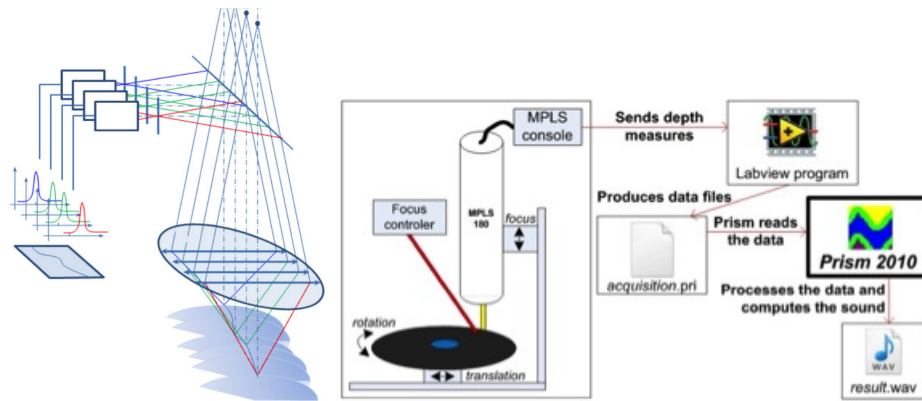


FIGURE 4.1: System schema [9] and probe

The figure 4.2 presents the PRISM software.

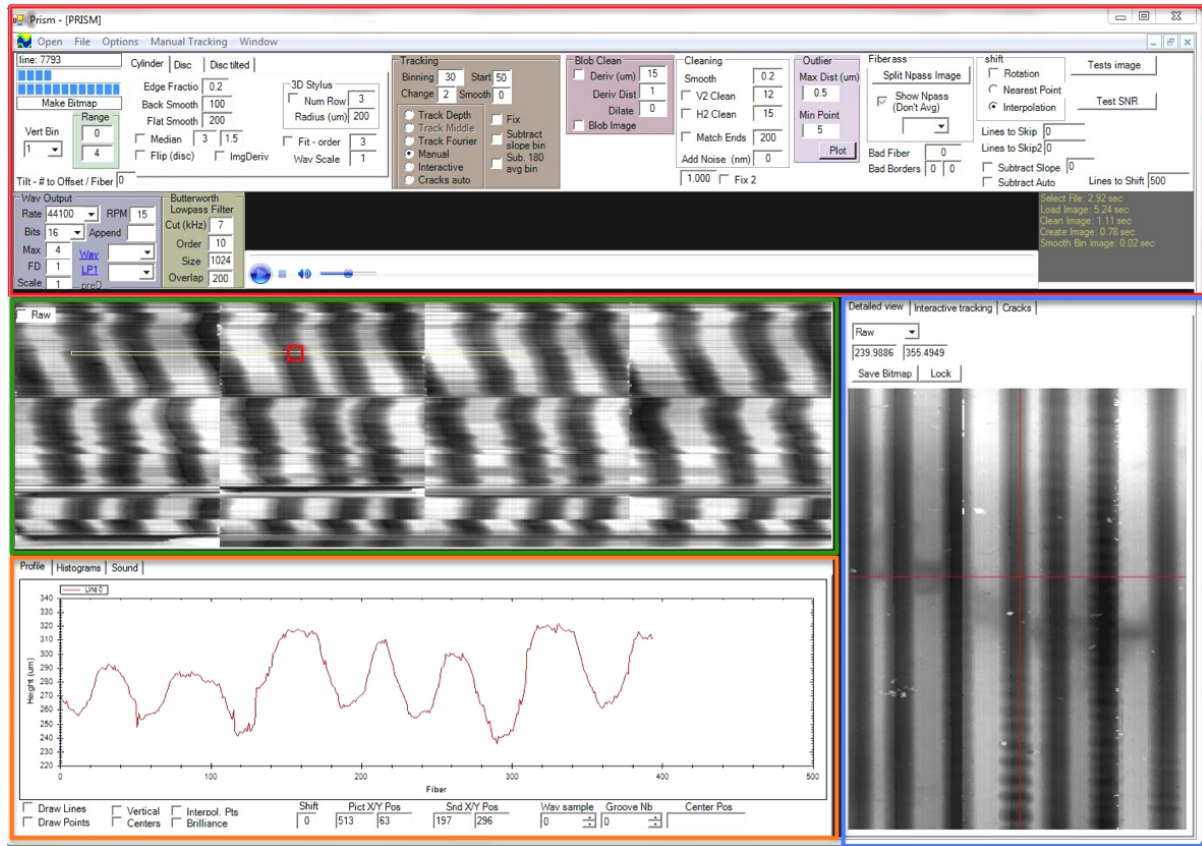


FIGURE 4.2: PRISM

- The top section (red) represents all the available tools including the recordings type (each type is processed differently), tracking options, cleaning functions, wav output parameters, etc...
- The middle section (green) displays a picture of the opened recording. The software converts the depth information into a gray-scale image. A black pixel represents a deep area in the recording.
- The left section (blue) displays a zoomed view of the yellow rectangle in the middle section. It also contains the cracks' parameters and interactive tracking tools.
- The bottom section (orange) displays the real values under the cursor in the detailed view. It is possible to display the horizontal or vertical pixels.

Images structure

Each step of the opening or cleaning process creates an image. Those images are stored in an array where the first index (i) represents the type of the picture (Bri, Raw, Flat, Sub Flat, Blob Clean, etc...) and the second index (x) represents the pixel position.

$$rawImages[i][x]$$

This structure implies that the pictures are stored in a one-dimensional array which makes the image processing a little tricky at first. The figure 4.3 shows an image stored in a two-dimensional array and its equivalent in the PRISM software.

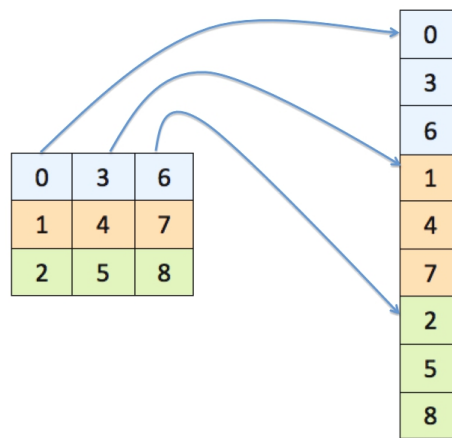


FIGURE 4.3: Image structure

The pixel at position (x,y) in image of type z can be accessed with the formula 4.1.

$$rawImages[z][x + y \cdot width] \tag{4.1}$$

4.1.2 Opening process of a recording

When we open a .PRI file in PRISM, some operations are automatically performed. We are going to focus on the operations for the cylinder because even if the recordings aren't cylinder, the algorithm used for this type suits the Volta Laboratory's recordings.

The figure 4.4 describes the opening process. The orange rectangles correspond to the image that are added to the image structure. Those image are available from the detailed view. The green circles represent operations performed by the software.

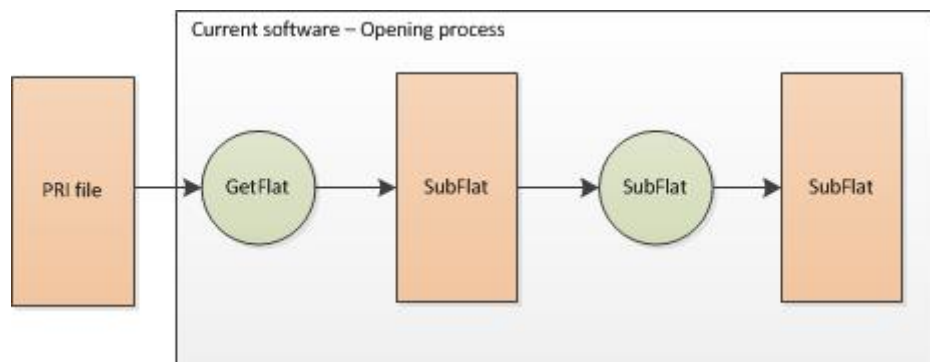


FIGURE 4.4: Opening process

For each pixel, the software computes an average of the number of pixels given (Flat Smooth parameter in the cylinder tab). Then it creates the subflat image by subtracting the flat image to the .PRI. This operation is performed to suppress the continuous component and obtain the depth information on the record. Then the cleaning process starts and uses the subflat image as an input.

4.1.3 Blob Clean function

The Blob Clean function deletes the dirt and dust particles present on the recordings. The figure 4.5 shows the different parameters of this function.

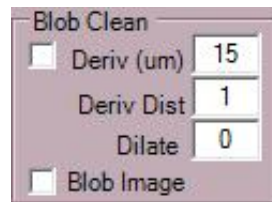


FIGURE 4.5: Blob Clean

Parameter	Description
Deriv (μm)	Sensitivity of the blob's detector. Big value \rightarrow less sensitive, low value \rightarrow more sensitive
Deriv Dist	Focus distance value to detect blobs. Big value can detect blob's gradient edge.
Dilate	Dilatation factor. For each edge point found, set a square of '1' at this place where the size is proportional to the dilatation factor.
Blob Image	Create an image with the thresholded blobs.

TABLE 4.1: Blob Clean function's parameters

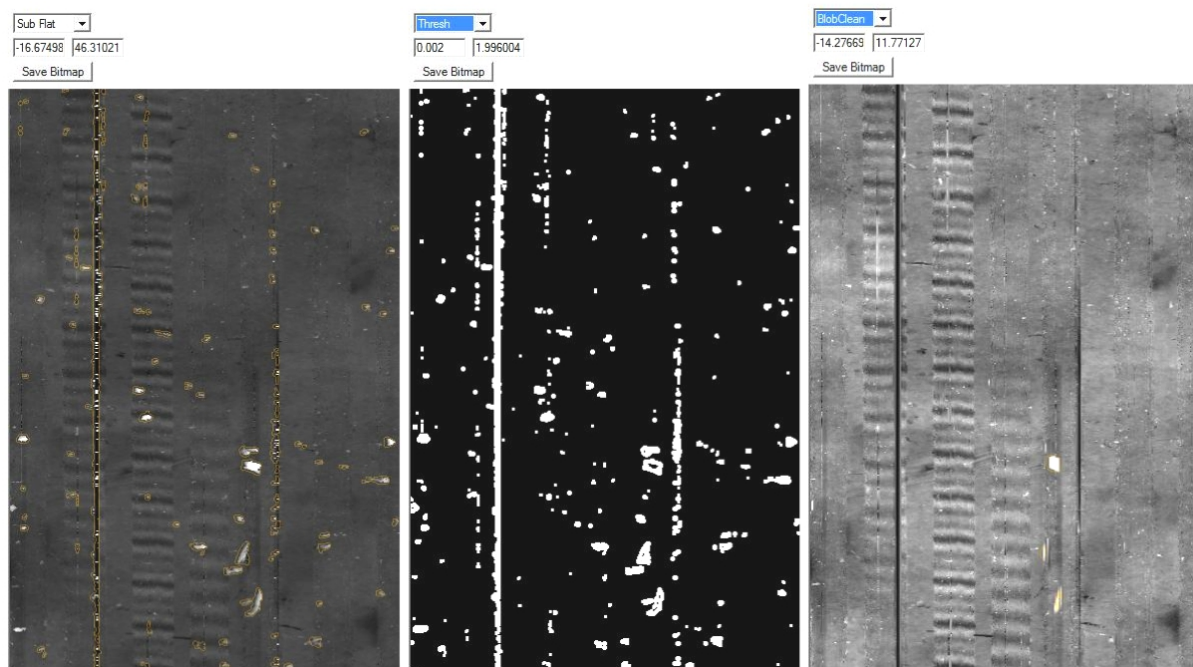


FIGURE 4.6: Blob Clean results

The figure 4.6 shows an example of the Blob Clean function. On the left, we can see the sub flat image of the record (input of the Blob Clean function) with the detected blobs in orange. The middle picture is the black and white image of the blobs and the right one is the result after the interpolation of the detected blobs.

The actual cleaning function uses a linear interpolation to delete the blobs (it reads the threshold image from top to bottom, column by column and draws a line between the previous and the next good points when a blob is present). This kind of interpolation is not well adapted to this type of file. We might think that for small blobs, a linear interpolation is the best choice, however as explained before, we use the derivative of the groove points to extract the sound. The derivative of a linear function is a constant, this creates jumps in the audio file resulting in audible clicks. Moreover, the blobs detection algorithm misses a lot of blobs or doesn't detect the whole blob as shown in the figure 4.7.

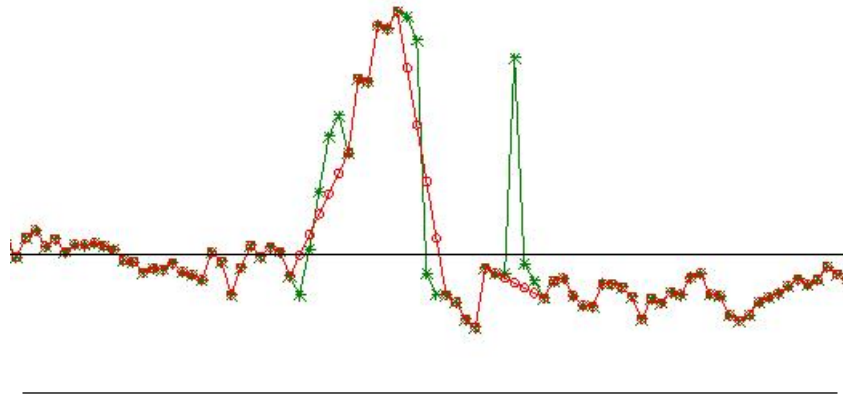


FIGURE 4.7: Blob Clean error

This figure represents a part of the recording's depth values. The big peak is a blob, the green line is the data before the Blob Clean function and the red one is the resulting data. We can see that the current algorithm doesn't always detect the whole blob.

getTresh(...) function

This function contains the blob detection algorithm. The table 4.2 explains the different parameters of this function.

Parameter	Description
int inv	Index of the image to process
int i1	Set to 0 in the code (magic number)
int j1	Set to 0 in the code (magic number)
int wd	Width of the image
int ht	Height of the image
float thresh	Deriv (μm) parameter
float vtresh	Deriv (μm) parameter
int deFocus	Deriv dist parameter
ref short[][] thr	Reference to an array that will contain the blobs
ref short[][] thr2	Reference to an array that will contain the blobs
float[][] v1	rawImage structure
int width	Width of the image
int height	Height of the image
TextBox txtDilate	Text box containing the dilate parameter

TABLE 4.2: getTresh() parameters

thr and thr2 are arrays that contains 1 if the pixel is a blob or 0 if it is not.

The function scans the entire image searching for blobs. It also takes into account the dilate and deFocus parameters (see figure 4.5 and table 4.2). Here are some explanation about the blobs detection process. In the figure 4.8, the current pixel is in blue:

		Y1		
		Y2		
X1	X2		X3	X4
		Y3		
		Y4		

FIGURE 4.8: Blobs's searching

The function decides if the current pixel (i,j) is a blob with the formula 4.2, the formulas 4.3 and 4.4 represents the influence of the deFocus parameter.

$$\frac{A[i - deFocus][j] - A[i + deFocus][j]}{thresh} + \frac{A[i][j - deFocus] - A[i][j + deFocus]}{thresh} > 1 \quad (4.2)$$

Where A is a two-dimensional array representing the image.

If deFocus = 1:

$$\frac{X2 - X3}{thresh} + \frac{Y2 - Y3}{thresh} > 1 \quad (4.3)$$

If deFocus = 2:

$$\frac{X1 - X4}{thresh} + \frac{Y1 - Y4}{thresh} > 1 \quad (4.4)$$

If the current pixel is detected as a blob, the function calls the setTresh(..) function.

setTresh(...) function

This function sets the current pixel as a blob. It sets a 3x3 square (time the dilate factor) of '1' around the pixel. The table 4.3 explains the different parameters of this function.

Parameter	Description
int i	X position of the pixel
int j	Y position of the pixel
int dilate	Dilate factor
ref short[] thr	Reference to an array that will contain the blobs

TABLE 4.3: setTresh() parameters

Conclusion

After the analysis and test of the present Blob Clean function, it appears that it fails to detect some blobs and to clean them correctly. The main goal of this project will be to design new tools to detect and clean the blobs in order to reduce the clicks that appear in the audio file and improve the audio quality.

4.2.3 Border pixels

There are different way to deal with the border pixels when processing a convolution.

- The value of the border pixel is repeated
- Mirroring the image
- Repeat the image
- Crop (the output image is smaller than the input image)
- Set the outside pixel to zero

For this diploma work, we repeated the image for the y-axis because the recordings' images are circular and we repeat the value of the border pixel for the x-axis as shown in the figure 4.10. The green parts represent the repeated pixel and the red parts represents the repeated image.

1	1	1	2	2	2
3	3	3	4	4	4
1	1	1	2	2	2
3	3	3	4	4	4
1	1	1	2	2	2
3	3	3	4	4	4

FIGURE 4.10: Border pixels

4.3 Statistics

4.3.1 Standard deviation

The standard deviation is represented by the Greek letter sigma σ and shows how the population (data set) varies from the average. If the standard deviation is low, it indicates that the data points are very close to the mean. If it is high, it indicates that the data points are spread out over a large range of values.

The figure 4.11 shows how the values are spread from the mean μ with a normal distribution.

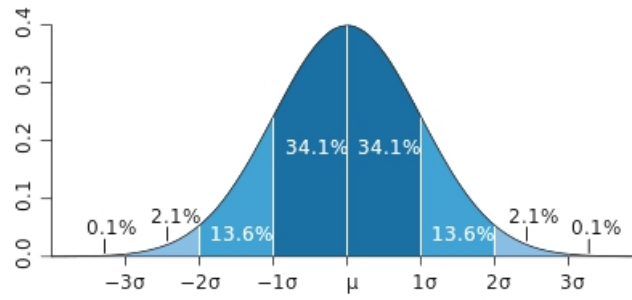


FIGURE 4.11: Standard deviation with a normal distribution

The equation 4.5 shows how to compute the standard deviation.

$$\sigma = \sqrt{\frac{1}{N-1} \sum_{i=1}^N (x_i - \bar{x})^2} \quad (4.5)$$

Where N is the number of samples, x_i the samples values and \bar{x} the mean of the samples.

During this project, we used the standard deviation for the blob detection algorithm and we had to compute the average for every pixel which took more time than expected. The equation 4.6 is another way to compute the standard deviation which takes much less time because it lets us use a sliding sum to compute it [12].

$$\sigma = \sqrt{\left(\frac{1}{N} \sum_{i=1}^N x_i^2 \right) - \left(\frac{1}{N} \sum_{i=1}^N x_i \right)^2} \quad (4.6)$$

4.4 Blobs Detection

In image processing, blob detection refers to mathematical methods that are aimed at detecting regions in a digital image that differ in properties, such as brightness or color, compared to areas surrounding those regions. As explained in chapter 2, for this project, the blobs represent the dirt and dust particles, cracks and scratches present on the recording. The aim of the blob detection function is to find those particles in order to delete them and to clean the image. In this chapter we will explore different techniques and algorithms to do so[7].

4.4.1 Laplacian of Gaussian

The most commonly used function for the blob detection is the Laplacian of Gaussian (LoG). This function uses a Gaussian filter to reduce the noise (because the Laplacian filter is a derivative filter and the derivative filters are very sensitive to noise) and then the Laplacian operator to detect edge. The Laplacian operator is defined by equation 4.7 and the Gaussian operator by equation 4.8 [13, 14].

$$L(x, y) = \nabla^2 f(x, y) = \frac{\partial^2 f(x, y)}{\partial x^2} + \frac{\partial^2 f(x, y)}{\partial y^2} \quad (4.7)$$

$$g(x, y) = \frac{1}{2\pi\sigma^2} \cdot \exp -\frac{x^2 + y^2}{2\sigma^2} \quad (4.8)$$

Where x and y are the distance from the origin $(0,0)$, and σ is the standard deviation of the Gaussian distribution.

Combining those two equations give us the Laplacian of Gaussian equation 4.9 which will be used to create the kernel.

$$LoG(x, y) = -\frac{1}{\pi\sigma^4} \left[1 - \frac{x^2 + y^2}{2\sigma^2} \right] \exp -\frac{x^2 + y^2}{2\sigma^2} \quad (4.9)$$

The LoG filter takes the second derivative of the image. Where the image is uniform, the result will be zero. When an edge is detected, the LoG will give a positive response on the darker side and a negative response on the lighter side. The figure 4.12 represents the LoG function and how the σ influences it [13]. The σ controls the width of the "bell". This affects the blur effect of the Gaussian low-pass filter.

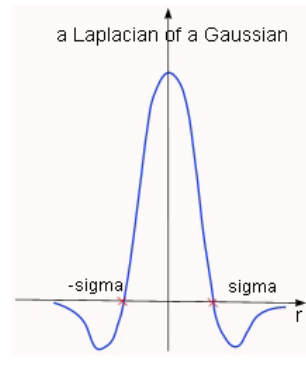


FIGURE 4.12: Laplacian of Gaussian cross section [15]

The figure 4.13 represents the different step of the LoG operator on a picture (intensity profile corresponds to the result of the Gaussian filtering).

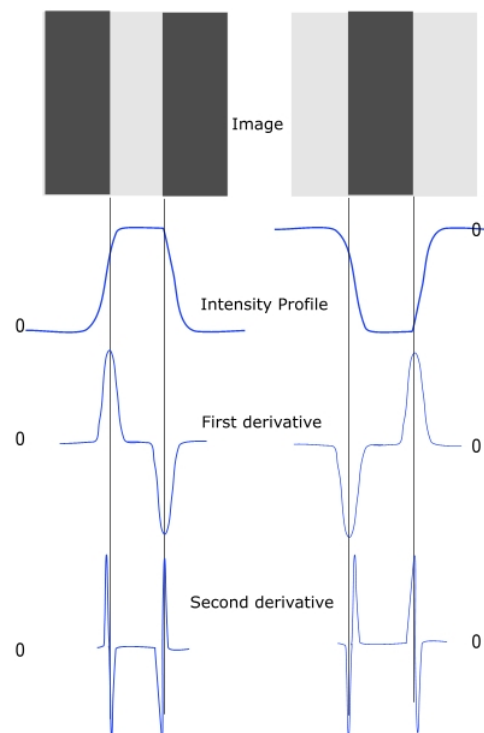


FIGURE 4.13: Laplacian of Gaussian steps [15]

We apply this filter by convolving the image with a kernel created with equation 4.9. The figure 4.14 represents a 9x9 kernel with $\sigma = 1.6$.

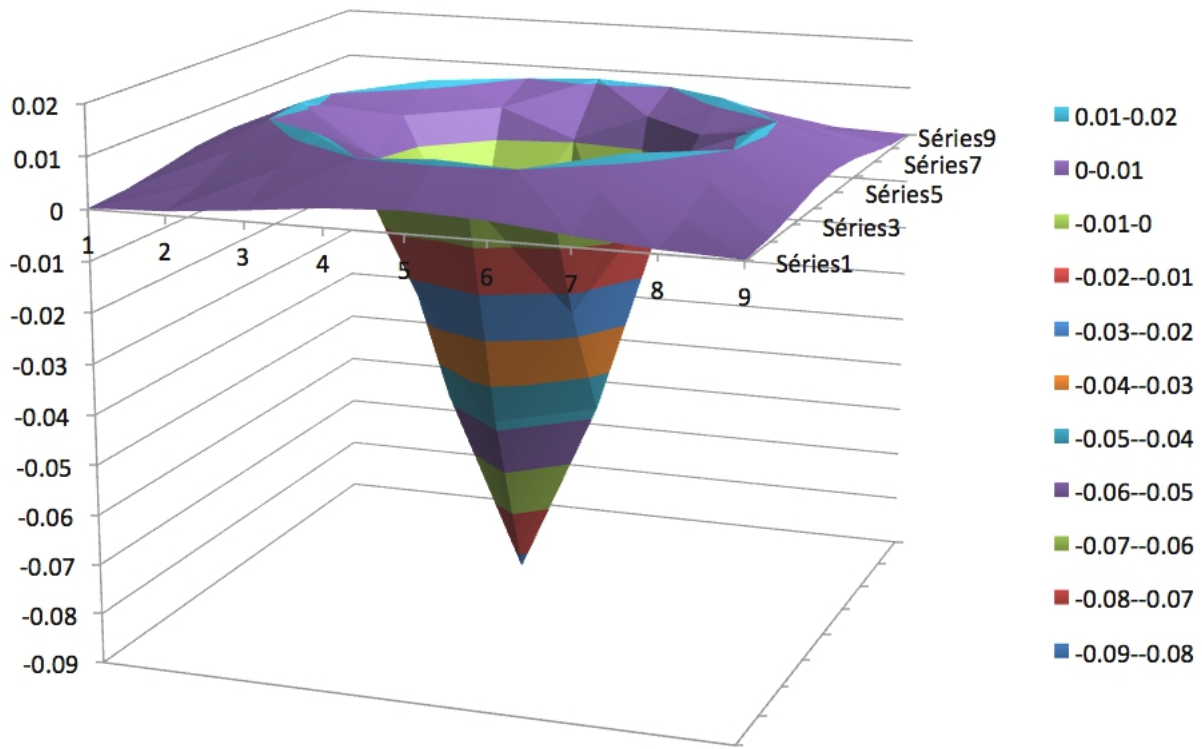


FIGURE 4.14: Laplacian of Gaussian 9x9 kernel with $\sigma = 1.6$

It is common to use a kernel size of $5 \cdot \sigma$ [16].

4.4.2 Adaptive threshold

Thresholding is used to separate the foreground and the background of an image. Whereas the standard thresholding operator uses a single threshold for all pixels, the adaptive thresholding changes the threshold dynamically. With this kind of threshold, it is possible to take into account the changing lighting conditions [17]. This type of algorithm can be used to extract/detect the blobs of a picture.

During this project, diverse adaptive threshold were used. For example by comparing the deviation between a pixel and the mean with the standard deviation of the pixels around or by comparing the pixel value with the mean of the pixels around.

4.5 Interpolation

4.5.1 General information

In mathematics, the interpolation is the principle of constructing new data points within the range of a discrete set of known points. In engineering, given a number of data points (obtained by sampling, measurement and experimentation) representing the value of a function, the interpolation is an estimation of the values of this function for unknown points. It is generally achieved with curve fitting or regression analysis.

4.5.2 Linear interpolation

The linear interpolation is the current method used in the algorithm. The value between each known points is given by a 1st order function [4.10](#).

$$y = a \cdot x + b \quad (4.10)$$

With the known points (x_a, y_a) and (x_b, y_b) , the y value at x can be calculated with equation [4.11](#).

$$y = y_a + (y_b - y_a) \cdot \frac{x - x_a}{x_b - x_a} \quad (4.11)$$

The figure [4.15](#) represents this kind of interpolation.

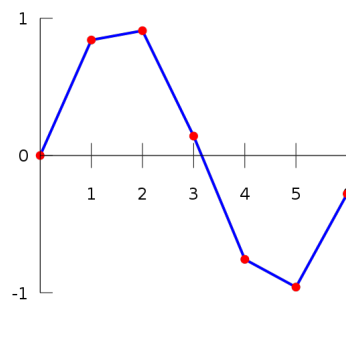


FIGURE 4.15: Linear interpolation [\[18\]](#)

We might think that linear interpolation is appropriate for small gaps/blobs, however as explained in chapter 2, PRISM takes the derivative of the groove points to extract the sound from the picture. The derivative of a linear function is a constant, this will create jumps in the audio file resulting in audible clicks. The figure 4.16 shows a sinus with a linear interpolation and its derivative.

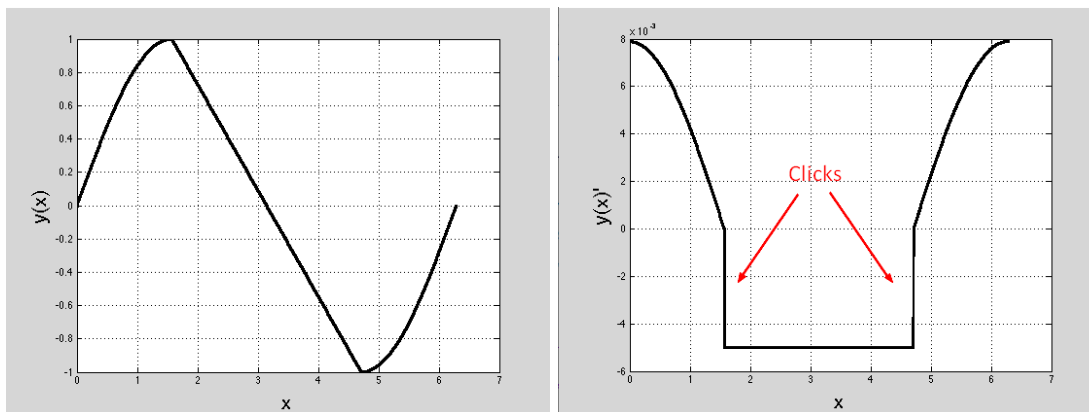


FIGURE 4.16: Derivative of linear function

4.5.3 Polynomial interpolation

The principle of polynomial interpolation is to fit one polynomial of degree $n-1$ going through all the n data points. This method works fine if the degree of the polynomial isn't too high. Otherwise, this method is computationally expensive, oscillatory artifacts might appear and the shape of the curve might be contrary to common sense [18]. We didn't analyze this kind of interpolation because it can create artifacts in the end points (Runge's phenomenon [19]) and we want something smooth to replace the blobs in order to avoid unnecessary clicks in the audio file.

4.5.4 Spline interpolation

The spline interpolation is a special type of piecewise polynomial [20] called a spline. This interpolation avoids Runge's phenomenon and is preferred over the polynomial interpolation. The spline will take a shape that minimizes the bending making the first and second derivative of the points will be continuous everywhere. This is very interesting for this project because the audio file will be continuous and no clicks will appear because of the interpolation.

Let $q_i(x_i)$ be a polynomial that represent the function between two data points. To achieve the previous statement, both equations 4.12 and 4.13 must be satisfied for all i , $1 \leq i \leq n - 1$. We need a polynomial of degree 3 or higher to achieve this [21].

$$q'_i(x_i) = q'_{i+1}(x_i) \quad (4.12)$$

$$q''_i(x_i) = q''_{i+1}(x_i) \quad (4.13)$$

The complete algorithm to find the interpolating cubic spline can be found in [21].

The figure 4.17 shows the same sinus that in figure 4.16 but with a spline interpolation and its derivative. We can see that there is not jump in the derivative.

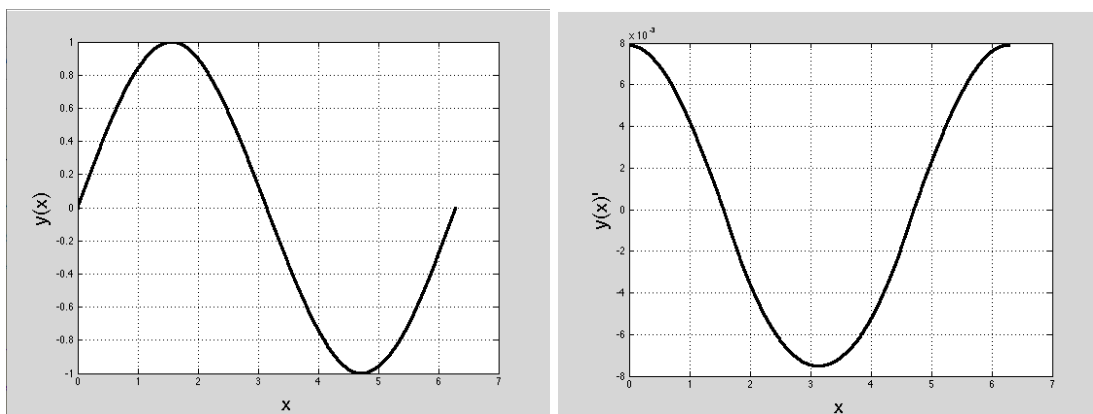


FIGURE 4.17: Derivative of the spline interpolation

4.6 Wav format

For the second part of the project, two objectives were to detect the background noise of an audio file and to replace the muted part with it. Even if a good knowledge of the wav format is unnecessary, we decided to present the basics of it.

The wav format is used to store audio. It contains a header and the raw audio data in time format. The table 4.4 explains some definition that will be used during the design of those functions [22].

Name	Description
Bit size	Number of bit per sample (16 in PRISM)
Sample	Represent the raw audio information
Sample rate	Number of samples per second (44100 in PRISM)
Channels	Number of channels (1 = mono, 2 = stereo, mono in PRISM)
Data	Represents a sample
Header	Used to provide information about the wav file, the header is 44 bytes long

TABLE 4.4: wav format - some definitions

4.7 Recordings

During this project, a set of the Volta Laboratory's records were taken. This section presents the different information we have about those records. The collection contains 465 records (189 cylinders, 275 discs or other flat media and 1 tape reel) created by Alexander Graham Bell, Charles Sumner Tainter and Chichester Bell at the end of the 19th century. P. Feaster created a discography of those records [4], however we don't know a lot about them. We calculated the sample frequency of these records with equation 4.14.

$$F_{sample} = \frac{height_{img} \cdot RPM}{60sec} \quad (4.14)$$

4.7.1 Record 287700

File name	287700_ 105_ 1800_ 31_ 18000_ 400_ 1.pri
Real picture	
Opening options	Vertical flip
RPM	15
Content	unknown
Inscription	Japan
Fs	4.5kHz
Other	Recorded in 1885, vertically cut

TABLE 4.5: Record 287700

4.7.2 Record 287701

File name	287701_ 110_ 1800_ 36_ 18000_ 400_ 1.pri
Real picture	
Opening options	Vertical flip
RPM	45
Content	unknown
Inscription	-
Fs	9.9kHz
Other	Recorded in 1885, vertically cut

TABLE 4.6: Record 287701

4.7.3 Record 287881


File name	287881-FULL1_110_1750_24_36000_400_1.pri
Real picture	
Opening options	Vertical flip, no 180 flip
RPM	15
Content	Alexander Graham Bell counting
Inscription	Record made April 15 1885, AGB and CAB, to test reproduction of numbers, disk AGB No. 1
Fs	9kHz
Other	Recorded 15 April 1885, vertically cut

TABLE 4.7: Record 287881

Chapter 5

Design and implementation

5.1 Blobs Clean

5.1.1 Blobs detection

As explain in chapter 4, the present function makes mistakes and takes parameters that aren't easy to set due to the large variety of blobs present on the Volta Laboratory's recordings. The figure 5.1 shows two parts of the record 287881 with a lot of different kind of blobs.

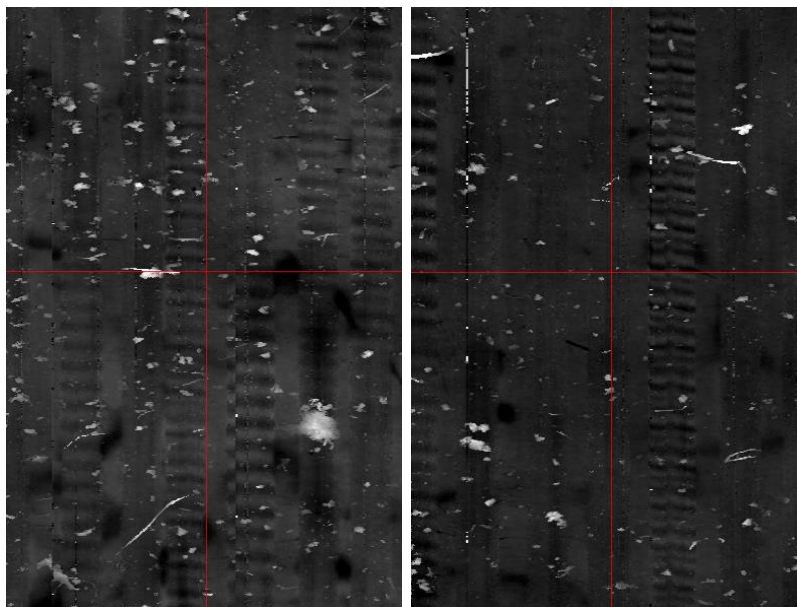


FIGURE 5.1: Blobs on record 287881

The aim of the new function is to detect blobs and to create a black and white image representing the blobs. This image will be used by the interpolation function to clean the recording.

General idea

After a lot of test and experimentation, it appears that it was not possible to detect properly all the blobs and cracks of the recordings with only one straightforward function because of their variety. As presented in the objectives (chapter 3), we had to design a function using no parameter. This is not possible because a value has to be set to help the algorithm to decide if the pixel is part of a blob. We tried to implement functions that are not simple thresholds in order to obtain the best results with most of the recordings.

We decided to take the advantages of all the experimentation we made and to combine the results to detect most of the blobs, cracks and probe's lost of focus. The blob detection process starts after the opening process explained in chapter 4. The figure 5.2 demonstrates this.

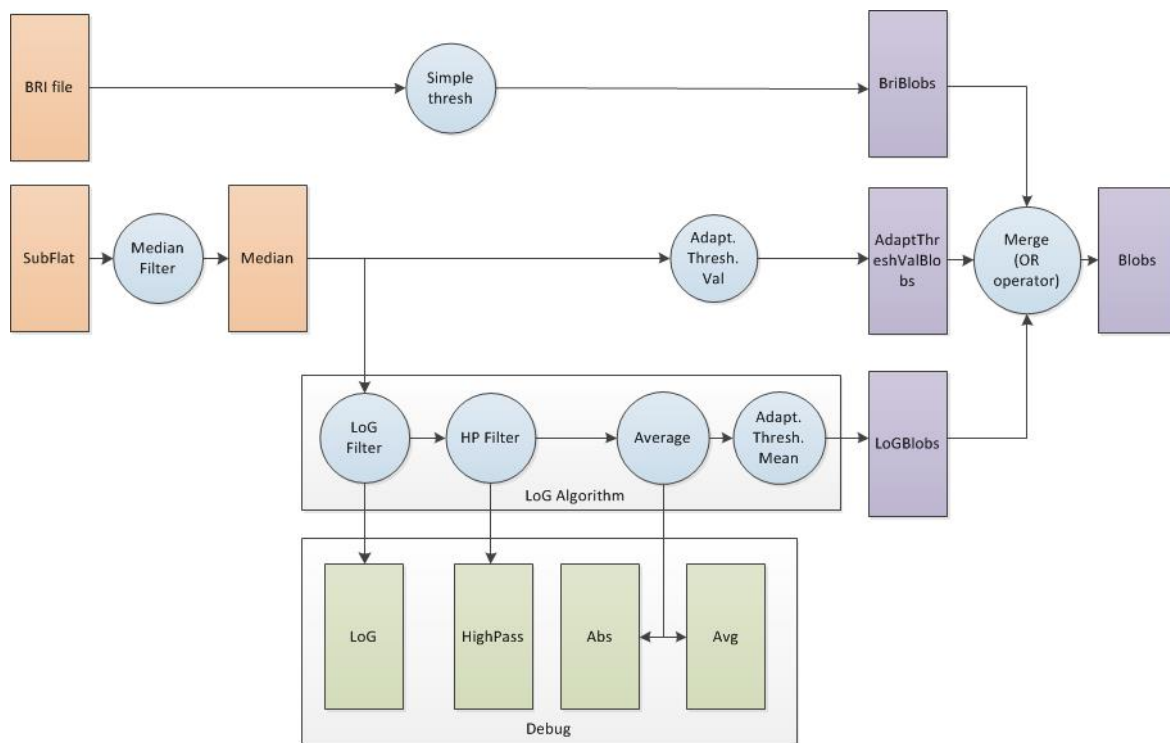


FIGURE 5.2: New blobs detection process

Orange rectangles represent the files that are always available in the detailed view.

Green rectangles represent the files that are added to the detailed view if the debug mode is selected in the view (see chapter 6 for more information).

Violet rectangles represent the files that are added when the blob image is selected (see chapter 6 for more information).

Blue circles represent functions.

BRI algorithm

As explained in the chapter 4, the BRI file represents the brightness of the points. At the current time, this file is not used in the PRISM Software. Carl Haber's team tried to use it to optimize the different algorithms without success. After some analysis, we saw that when the probe lost focus, the brightness of those points were very low compared to the normal points (figure 5.3). The detailed view shows the subflat image and the right one shows the values of the points under the vertical red line in the BRI file. We decided to use this information to detect those errors. This is performed by a simple threshold (equation 5.1).

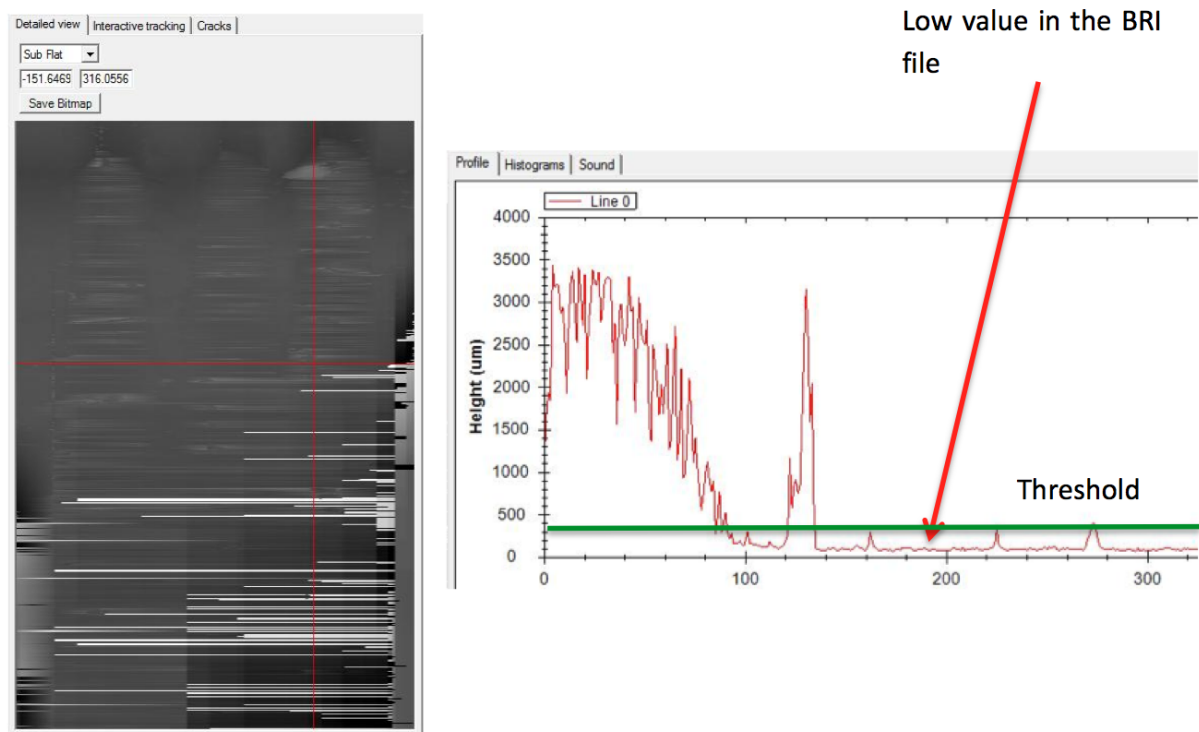


FIGURE 5.3: Points' value when lost of focus

$$BRIvalueOfCurrentPixel < Threshold \quad (5.1)$$

To avoid false detection of blobs, we decided to process the morphological closing operation (erosion following by a dilatation). This operation removes the small detected blobs that aren't real blobs.

LoG algorithm

We decided to design a new version of the LoG filter taking two different σ for the kernel's creation because the pixel doesn't represent the same size in μm on the record if we are in the border or in the center of the disc. This is due to the capture system. It captures rings over the record with the same resolution (number of pixels). As the border ring is longer on the record than the center ring but not in the "record's picture", one pixel doesn't represent the same size in μm . The figure 5.4 demonstrate this. The X value is a parameter that can be set before the capture to adjust the sampling frequency.

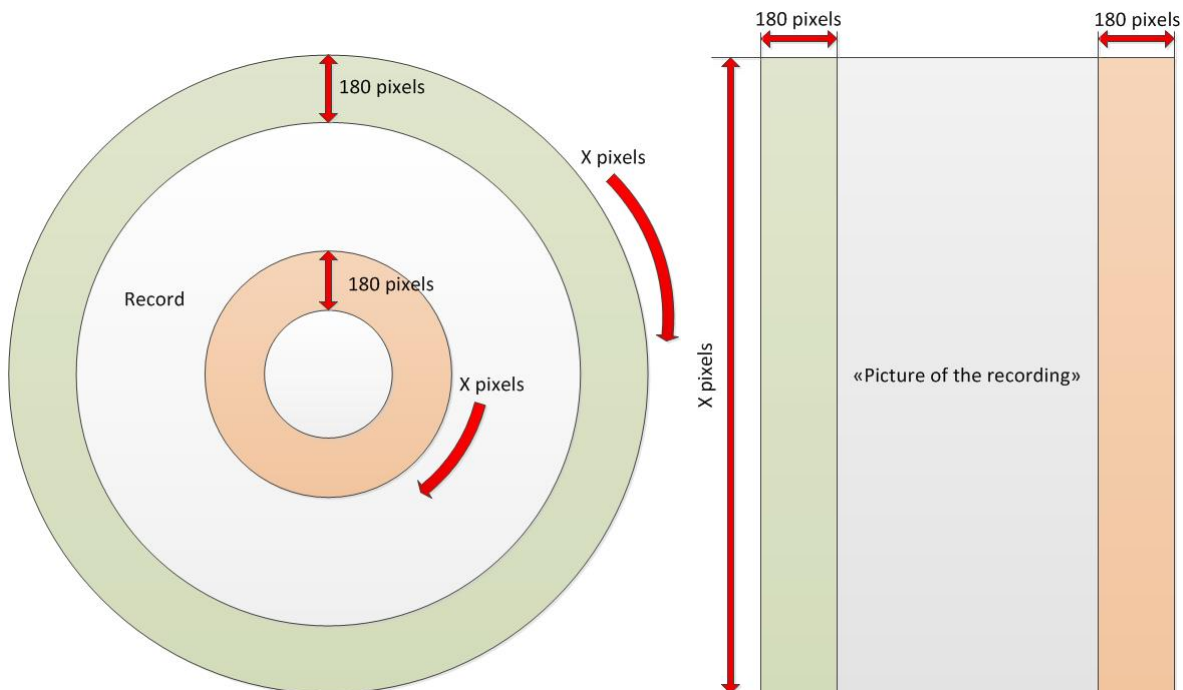


FIGURE 5.4: Pixels size

The Laplacian's equation (4.7) takes the second derivative of the function. Knowing this, it is possible to create a new LoG equation taking two different σ as parameter by computing the second derivative of the equation 5.2. This equation is an adaptation of equation 4.8.

$$g(x, y) = \frac{1}{2\pi\sigma^2} \cdot \exp - \left(\frac{x^2}{2\sigma_x^2} + \frac{y^2}{2\sigma_y^2} \right) \quad (5.2)$$

The term $\frac{1}{2\pi\sigma^2}$ is a normalization factor that can be omitted because we will normalize the kernel later.

$$LoG(x, y) = \nabla^2 g(x, y) = \frac{\partial^2}{\partial x^2} \left(\exp - \left(\frac{x^2}{2\sigma_x^2} + \frac{y^2}{2\sigma_y^2} \right) \right) + \frac{\partial^2}{\partial y^2} \left(\exp - \left(\frac{x^2}{2\sigma_x^2} + \frac{y^2}{2\sigma_y^2} \right) \right) \quad (5.3)$$

$$LoG(x, y) = \frac{1}{\sigma_x^4} \cdot (x^2 - \sigma_x^2) \cdot \exp - \left(\frac{x^2}{2\sigma_x^2} + \frac{y^2}{2\sigma_y^2} \right) + \frac{1}{\sigma_y^4} \cdot (y^2 - \sigma_y^2) \cdot \exp - \left(\frac{x^2}{2\sigma_x^2} + \frac{y^2}{2\sigma_y^2} \right) \quad (5.4)$$

$$LoG(x, y) = \left(\frac{x^2 - \sigma_x^2}{\sigma_x^4} + \frac{y^2 - \sigma_y^2}{\sigma_y^4} \right) \cdot \exp - \left(\frac{x^2}{2\sigma_x^2} + \frac{y^2}{2\sigma_y^2} \right) \quad (5.5)$$

From equation 5.5, we created two different kernels figure 5.5 ($\sigma_x = 1.6$ and $\sigma_y = 1.6$) and 5.6 ($\sigma_x = 1.6$ and $\sigma_y = 2.5$) to see how this new formula influence the kernel.

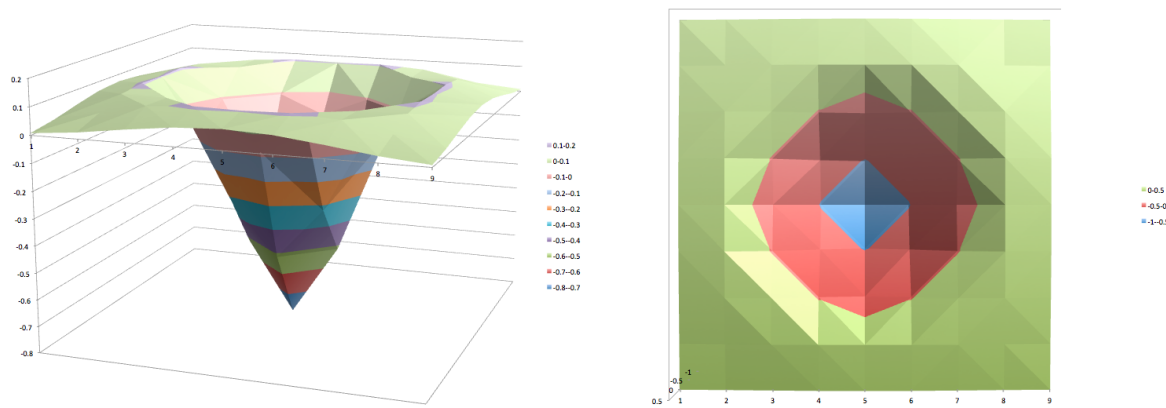


FIGURE 5.5: LoG kernel with $\sigma_x = 1.6$ and $\sigma_y = 1.6$, right is the top view

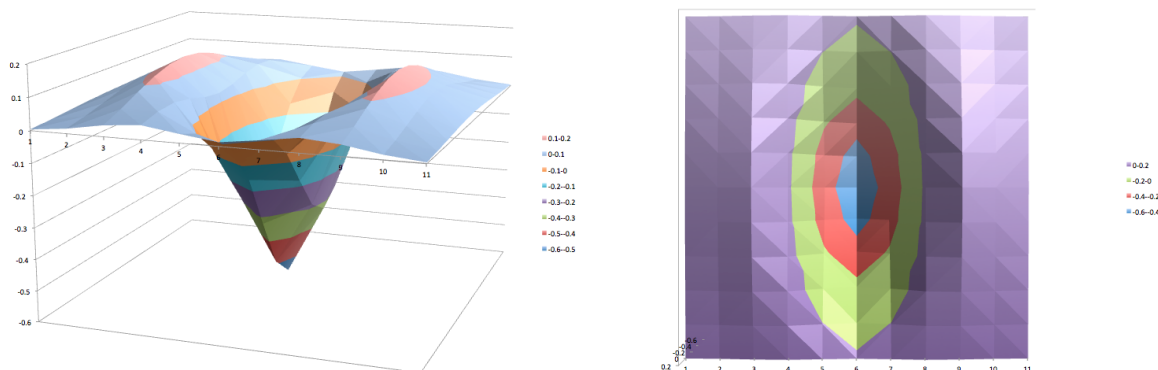


FIGURE 5.6: LoG kernel with $\sigma_x = 1.6$ and $\sigma_y = 2.5$, right is the top view

The figure 5.7 represents the two previous kernels after normalization. With this kind of blob detection algorithm, the second step is to detect the zero-crossing points and then use a threshold on the derivative to decide if it is a blob or the background. However, it is not possible with the pictures of the recordings because the luminosity isn't constant.

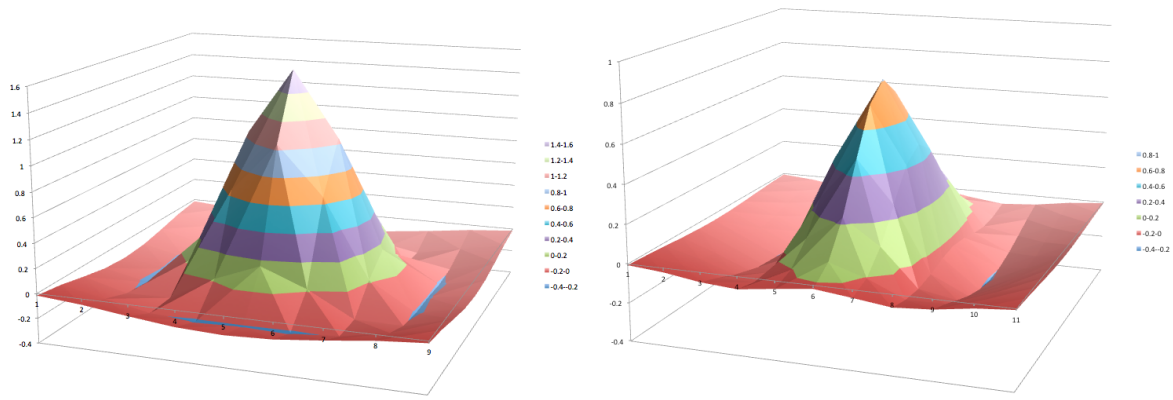


FIGURE 5.7: Normalized LoG kernel with $\sigma_x = 1.6$ and $\sigma_y = 2.5$, right is the top view

The figure 5.8 shows the convoluted picture of a record and a LoG kernel. The right picture represent the pixel values under the vertical red line in the left picture. We can see that it is not possible to detect the zero-crossing points because we will miss some blobs (or detect them but not as big as they really are).

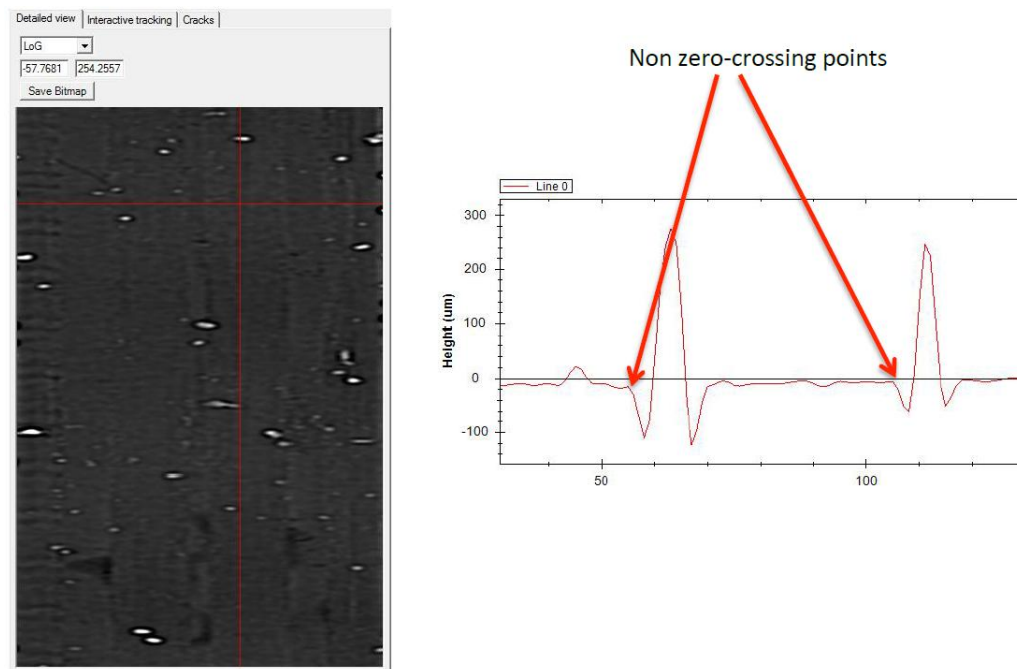


FIGURE 5.8: Convoluted picture

However, this filter is useful because it enhances the blobs (figure 5.9). We can use it with other filters and image-processing kernel to extract the blobs.

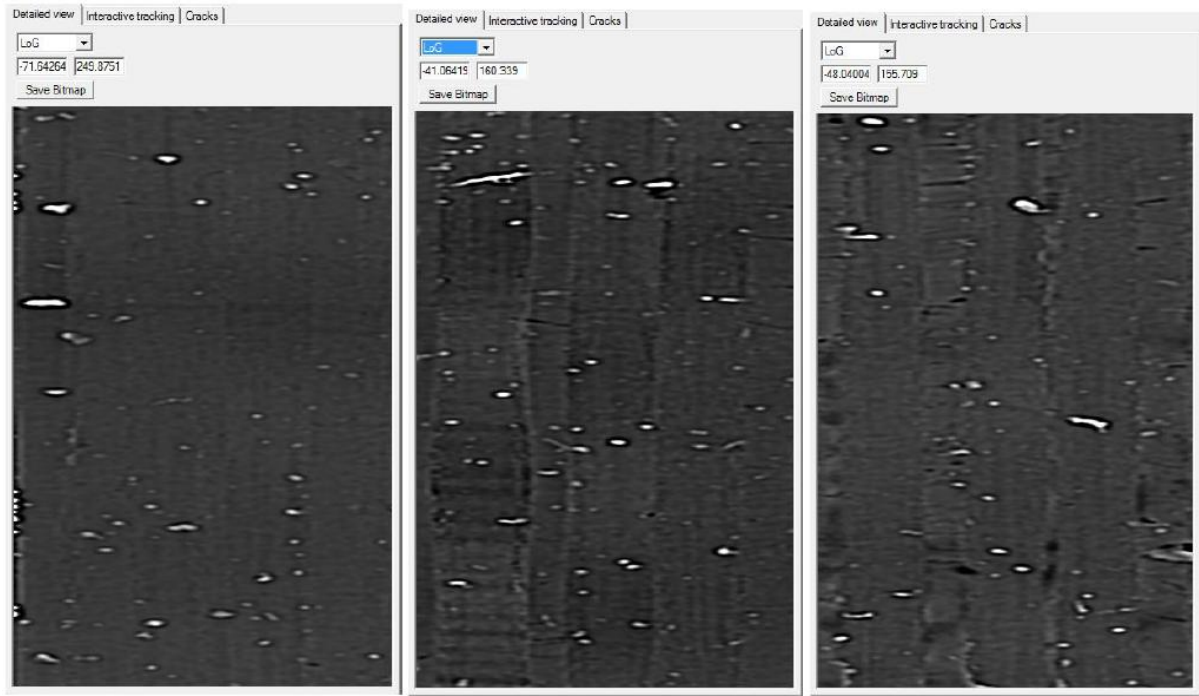


FIGURE 5.9: Enhanced blobs

It has the advantage to give us the size and position of the blobs. The current blobs detection function fails when it comes to detect all the blobs and their size as shown on figure 5.10.

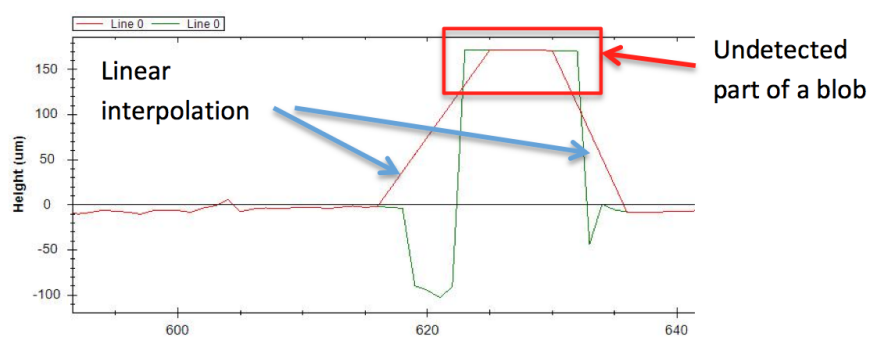


FIGURE 5.10: Error of the current blobs detection algorithm

The green line represents pixels' values of a recording before the blob detection and cleaning function. The red line represents the results of the current algorithm and the red square represent a blob. We can see that it doesn't detect the complete blob.

As shown in the figure 5.2, we use an high-pass filter after the LoG operator to reduce the low frequency fluctuation on the records. This includes the grooves information and the recording undulation. The figure 5.11 shows the grooves information after the LoG filter.

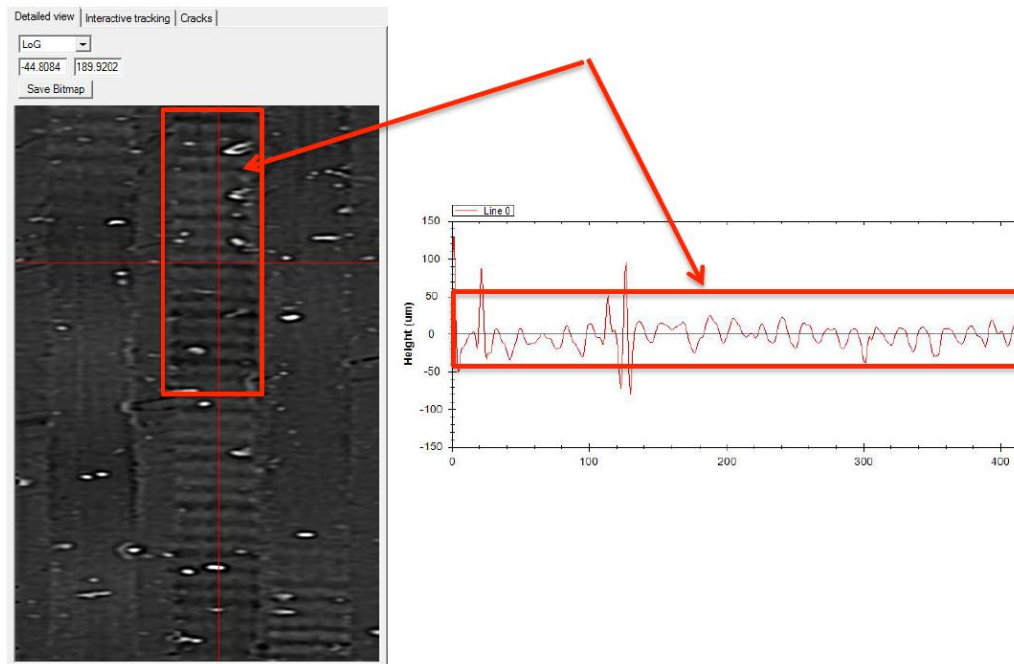


FIGURE 5.11: Groove information

We use a simple high-pass kernel (figure 5.12) and compute it with the image in order to reduce the low frequency.

-2	-2	-2
-2	8	-2
-2	-2	-2

FIGURE 5.12: High-pass kernel

The figure 5.13 shows the input and the figure 5.14 shows the output of the high-pass filter.

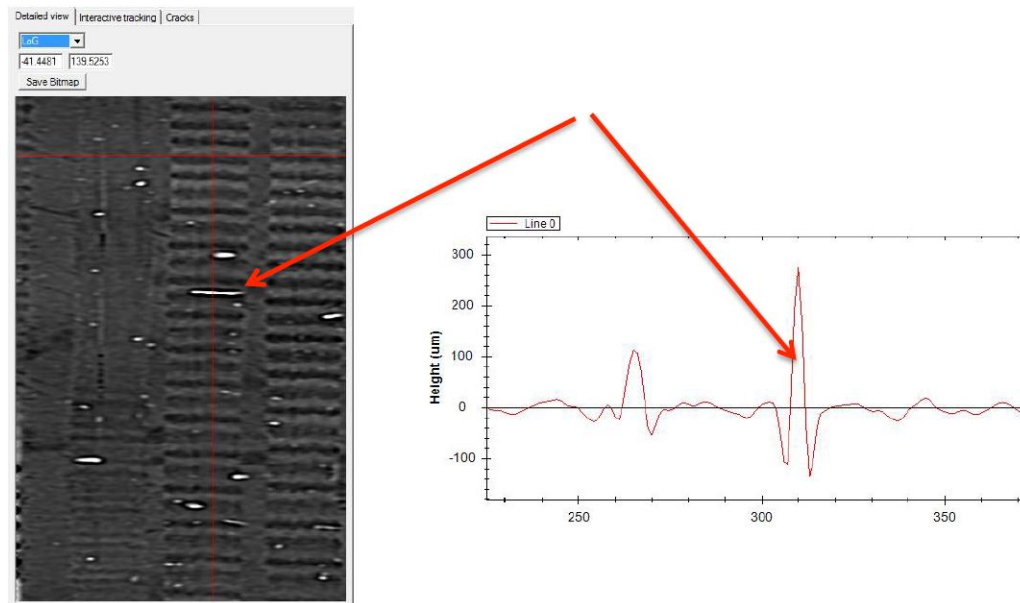


FIGURE 5.13: Input of the high-pass filter

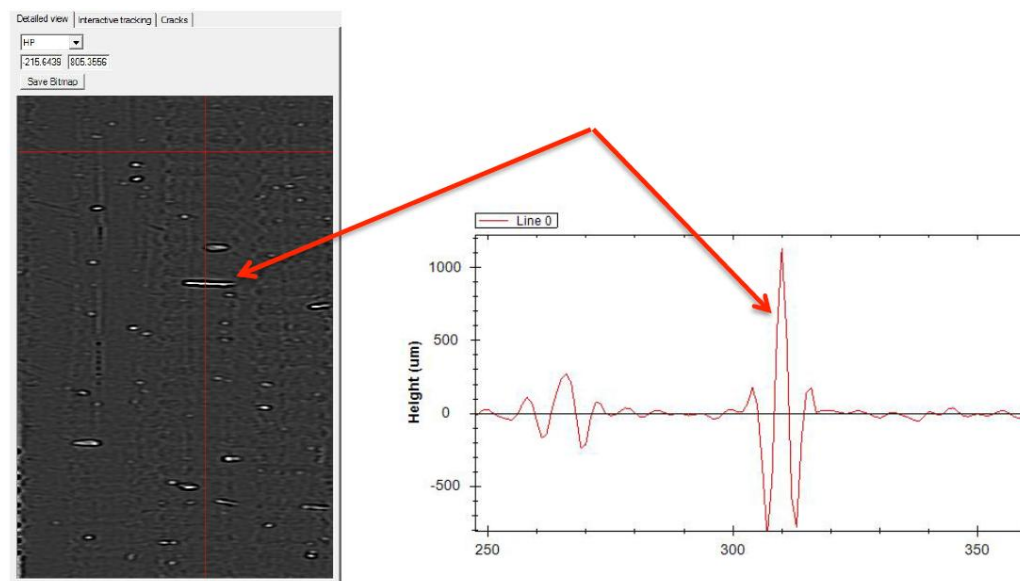


FIGURE 5.14: Output of the high-pass filter

We can see that the blobs are even more enhanced and the groove information is reduced. The next step is to compute an averaging kernel with the absolute values of the high-pass filter's output.

The goal of the averaging filter is to create a picture with only big values when the blobs are present in order to use an adaptive threshold to detect them. We take the absolute values to avoid big positive and negative values to cancel each other. This filter is implemented with an averaging kernel of size 5x5 (figure 5.15).

0.04	0.04	0.04	0.04	0.04
0.04	0.04	0.04	0.04	0.04
0.04	0.04	0.04	0.04	0.04
0.04	0.04	0.04	0.04	0.04
0.04	0.04	0.04	0.04	0.04

FIGURE 5.15: Averaging kernel

The figure 5.16 shows the input of the averaging filter, the absolute values and finally the output.

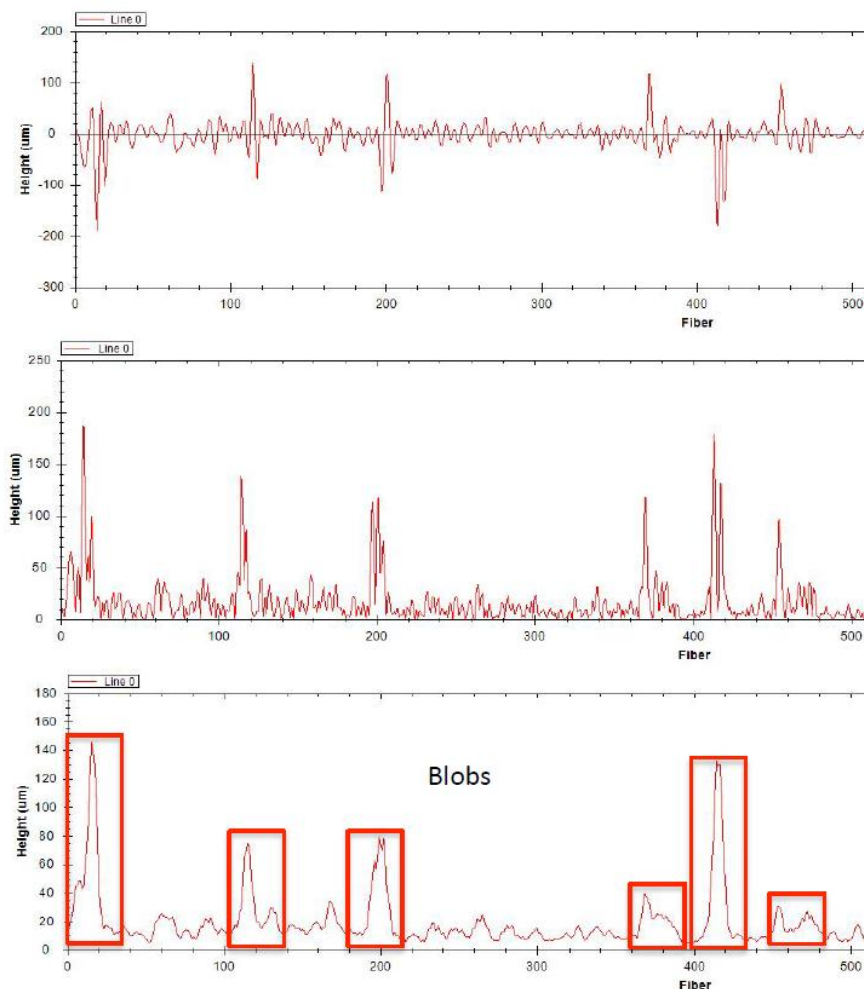


FIGURE 5.16: HP-Filter - absolute values - averaging filter output

The final operation is to extract the blobs from the picture using an adaptive threshold. This threshold is computed by getting the mean value of the specified number of points (in column and in row) around the current pixel. Then we compare its value to the mean times a specified value (equation 5.6).

$$CurrentPixel > Mean \cdot nbMeanAuth \quad (5.6)$$

As explained before, we are computing this operation vertically and horizontally around the current pixel (figure 5.17).

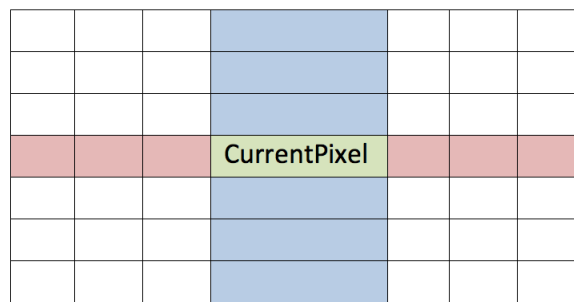


FIGURE 5.17: Pixels took into account for the adaptive threshold

The pixel is defined as a blob with the formula 5.7.

$$CurrentPixel > Mean \cdot nbMeanAuth \text{ OR } CurrentPixel > Mean \cdot nbMeanAuth \quad (5.7)$$

The number of pixels used for the mean has to be high enough otherwise the result might be wrong if a blob is big.

Adaptive threshold algorithms

For the first algorithm (AdaptThreshValBlobs), we use an adaptive threshold that is calculated by the standard deviation of the number of given point around the current pixel and then by comparing the deviation of the current pixel and the mean with it times a defined value. We decide if the current pixel is a blob with equation 5.8.

$$|CurrentPixel - Mean| > nbStdDevAuth \cdot stdDev \quad (5.8)$$

For the second algorithm (AdaptThreshDerivBlobs), we use an adaptive threshold that is calculated by the standard deviation of the derivative for the number of point given around the current pixel and then by comparing the deviation of the current derivative and the mean of derivative to a constant times the standard deviation. We decide if the current pixel and the next one are part of a blob with equation 5.10. The current derivative is compute with equation 5.9.

$$CurrentDerivative = |image[x + y \cdot width] - image[x + ((y + 1) \pmod{height}) \cdot width]| \quad (5.9)$$

$$|CurrentDerivative - MeanOfDerivative| > nbStdDevAuth \cdot stdDev \quad (5.10)$$

This algorithm detects the edge of the blobs but misses a function to fill the blobs. We developed this algorithm when were trying a lot of different functions in order to detect the blobs without parameter and with only one method. Even if this algorithm is not really useful for the blobs detection, we decided to let it in the code.

5.1.2 Blobs correction

The blob detection algorithm (figure 5.2) creates a black and white image with the value 1 if the pixel is part of a blob and 0 if it is part of the background (figure 5.18).

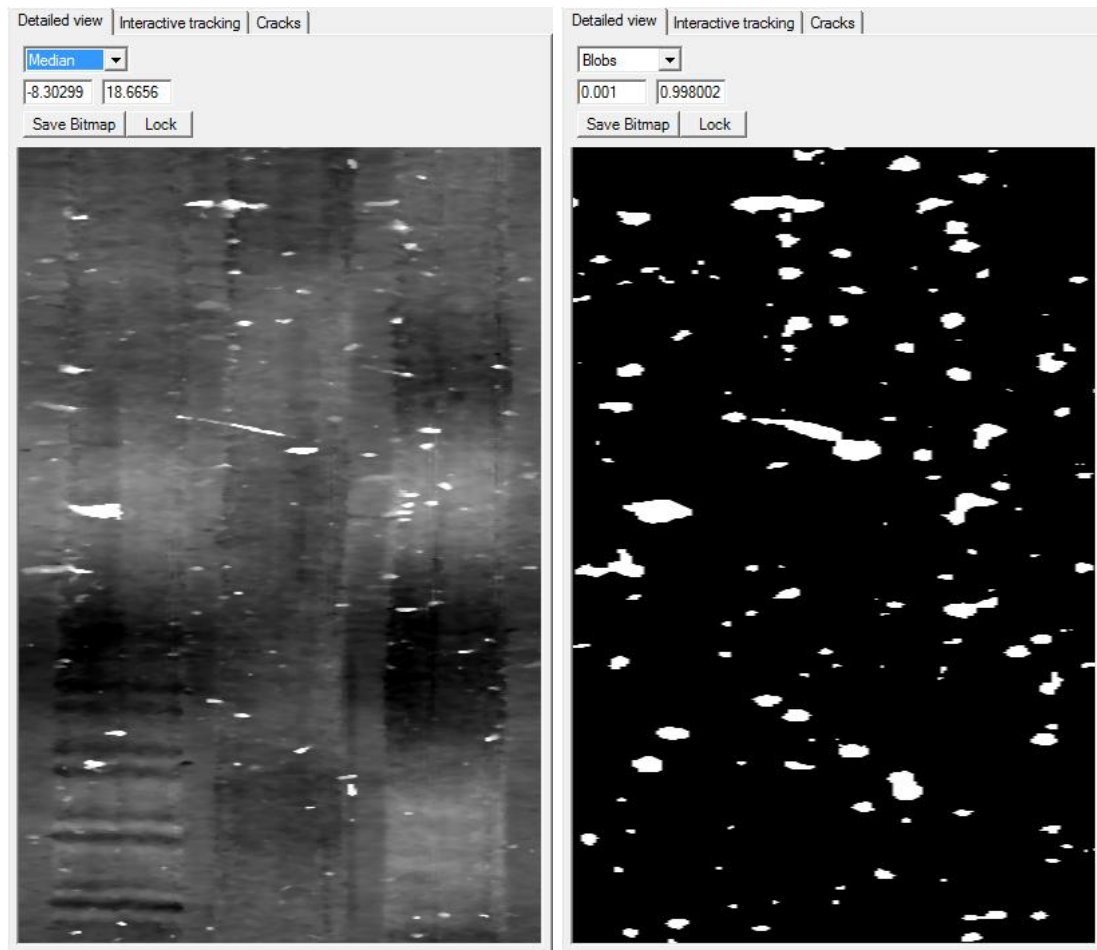


FIGURE 5.18: Black & white image created by the blob detection algorithm

As explained in the chapter 4, we use a cubic spline to correct the detected blobs. We found a cubic spline interpolation class in the Irene software (equivalent to PRISM software but for the 2D probe). The algorithm used is the one described in [21].

The figure 5.19 shows the cubic spline interpolation of the blobs. We decided to mute the sound (set values to zero) for the biggest blobs because when many points are missing, it is better to mute the sound than to create values that don't make sense.

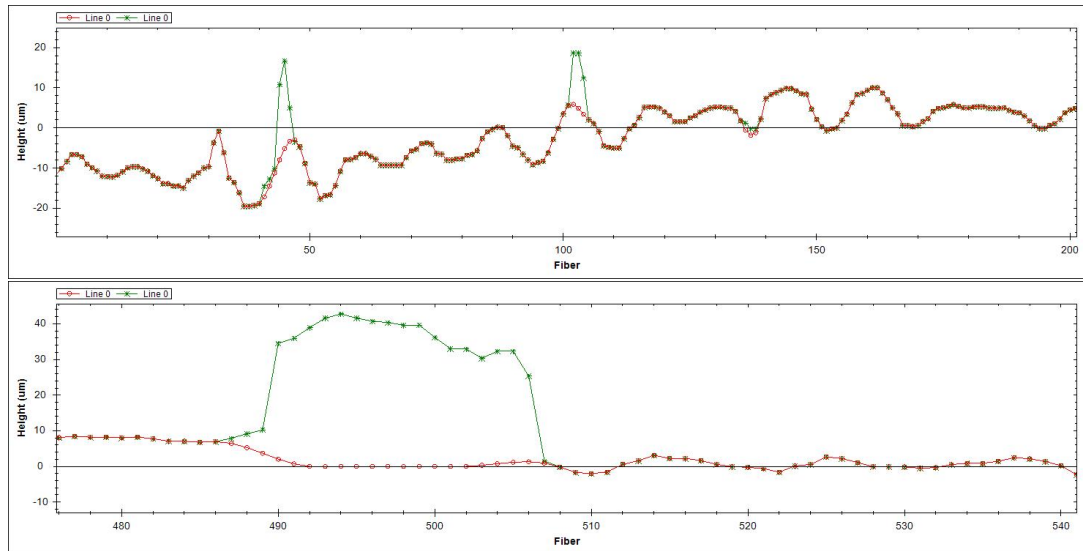


FIGURE 5.19: Results of the spline interpolation

We created an interface to use the interpolation. This interface defines the method (parameters' type, name) in order to add other type of interpolation (linear, polynomial, etc...) as further development. Each class implementing this interface will have to conform to it.

```
interface IInterpolation
{
    float[] interpolate(float[] points, Gap gap);
}
```

- This function returns an array with the interpolated values.
- The first parameter is an array of good points around the gap.
- The second parameter is an object representing a gap 5.20.

Gap
+startIndex : int
+length : int

FIGURE 5.20: Gap Class

The figure 5.21 describes the relation between the array *Points[]* and the object *gap*.

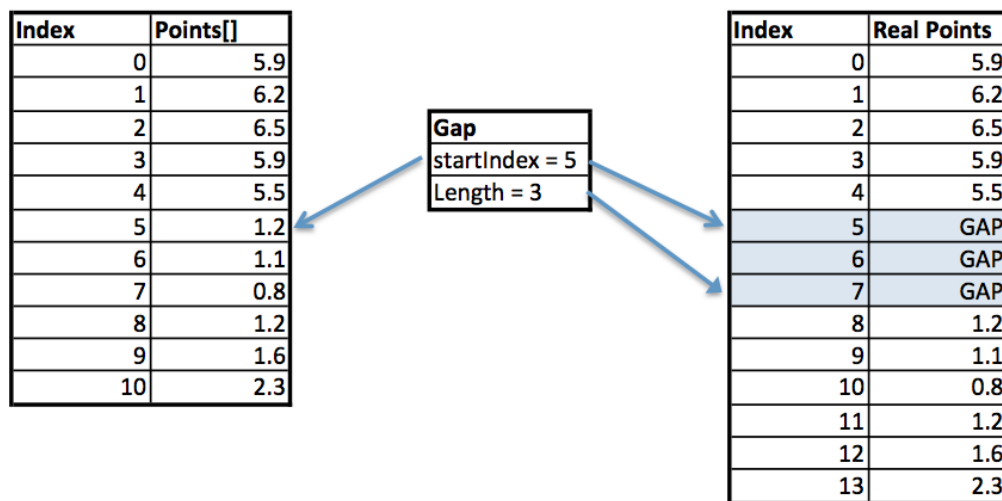


FIGURE 5.21: Relation between the points and gap

5.1.3 BlobClean v2 - Implementation

We implemented the blob detection process in the method `cleanImage` of the `Cylinder.cs` class. This method regroup all the cleaning functions that are implemented in PRISM. Each function is selected with a checkbox item in the GUI. The figure 5.22 represents the complete process of the new blob clean function.

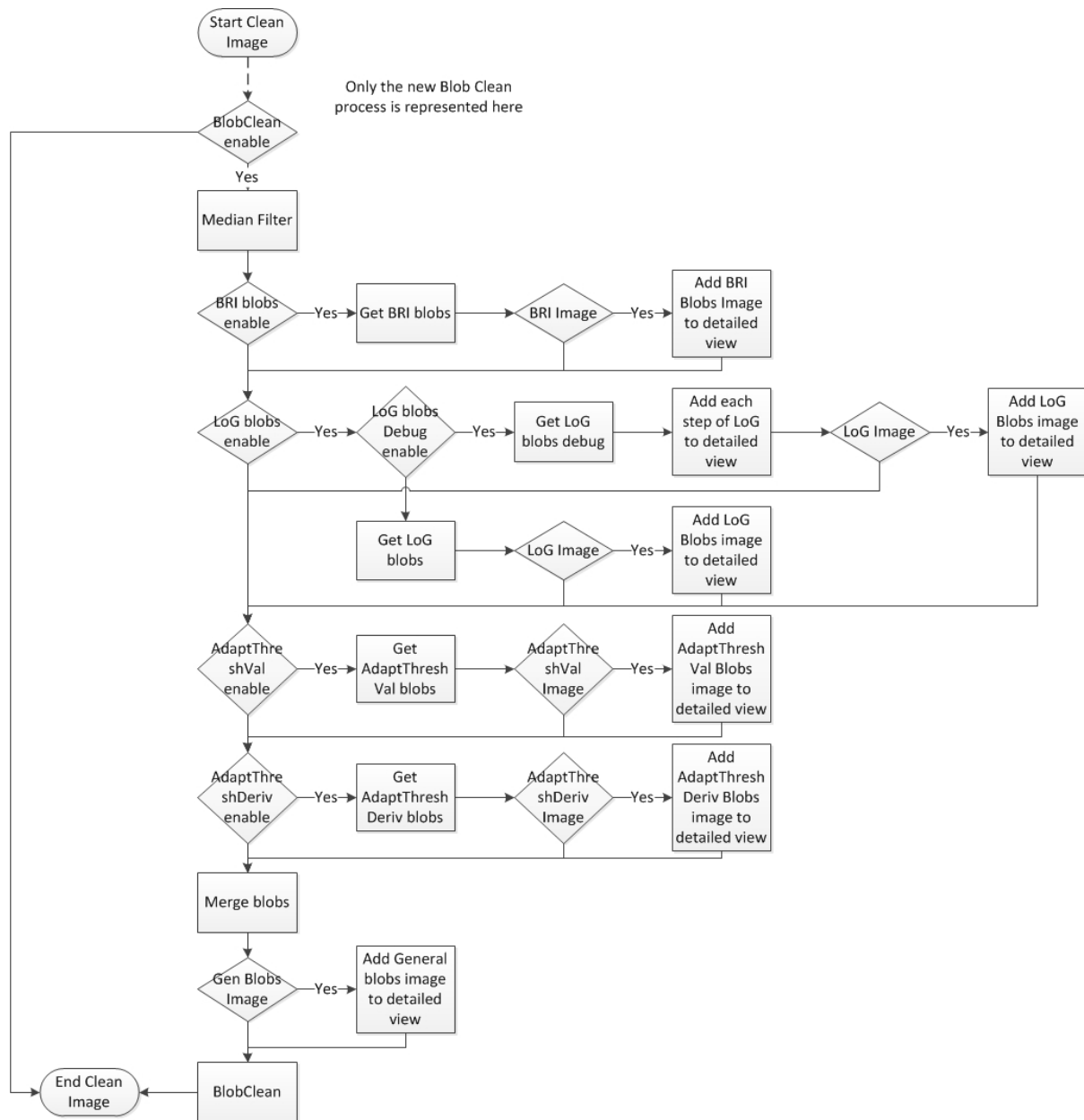


FIGURE 5.22: Complete blob clean process

BlobDetect Class

We created a static class regrouping all the algorithm and function used to detect the blobs (figure 5.23).

BlobDetect
-AdaptThreshValNbPoints : int = 201 -LoGAdaptThreshNbPoints : int = 301 -AdaptThreshDerivNbPoints : int = 201 -KernelSize : int = 0
+merge(in blobArray : float[][], in nbAlgoUsed : int, in width : int, in height : int) : float[] +getBriBlobs(in briImage : float[], in briThresh : float, in width : int, in height : int) : float[] +getLogBlobs(in image : float[], in sigmaX : float, in sigmaY : float, in nbMeanAuth : float, in width : int, in height : int) : float[] +getLogBlobsDebug(in image : float[], in sigmaX : float, in sigmaY : float, in nbMeanAuth : float, in width : int, in height : int) : float[][] +getAdaptThreshValBlobs(in image : float[], in nbStdDevAuth : float, in width : int, in height : int) : float[] +getAdaptThreshDerivBlobs(in image : float[], in nbStdDevAuth : float, in width : int, in height : int) : float[] -LoG(in x : int, in y : int, in sigma : float) : float -LoG(in x : int, in y : int, in sigmaX : float, in sigmaY : float) : float -LoG2D(in sigma : float) : float[,] -LoG2D(in sigmaX : float, in sigmaY : float) : float[,] -adaptThreshMean(in image : float[], in nbPoints : int, in nbMeanAuth : float, in width : int, in height : int) : float[] +setMoreParam(in adaptThreshValNbPoints : int, in loGAdaptThreshNbPoints : int, in adaptThreshDerivNbPoints : int, in kernelSize : int)

FIGURE 5.23: BlobDetect class

The static variables can be changed by accessing in the menu options/BlobClean2 - more parameters (see section GUI for more information).

BlobClean Class

The BlobClean class (figure 5.24) implements the interface *IInterpolation* and contains all the method to prepare the array *points* and the object *gap* for every blob to correct. This class creates a new image with the blobs replaced by the interpolation results.

«implementation class» BlobClean
-nbGoodPoints : int = 50 -t1 : int = 5 -maxSpline : int = 20
+interpolate(in points : float[], in gap : Gap) : float[] +clean(in image : float[], in thresh : float[], in width : int, in height : int) : float[] -getBlobSize(in thresh : float[], in width : int, in height : int, in x : int, in y : int) : int -getNbPrevGoodPoints(in thresh : float[], in width : int, in height : int, in x : int, in y : int, in max : int) : int -getNbNextGoodPoints(in thresh : float[], in width : int, in height : int, in x : int, in y : int, in max : int, in blobSize : int) : int

FIGURE 5.24: BlobClean class

The `clean` function is called to process the complete cleaning. It takes in parameters the image to correct, the output of the blob detection algorithm and the size of the image. This function will create the `points` array and `gap` object in order to call the `interpolate` function. This is done by using 3 private methods. One gives the blob's size in the actual column, the two others give the number of good points before and after the blob with a maximum defined by the constant `nbGoodPoints`.

To interpolate the blob's points, we use the class `Utils.CubicSpline` that was available in the Irene software. The function `FitAndEval` returns an array with the interpolated points.

```
public float[] FitAndEval(float[] x, float[] y, float[] xs, bool debug = false)
{
    ...
}
```

x represents the x values of the good points around the gap (here the index).

y represents the y values of the good points around the gap.

xs represents the x values of the points to interpolate.

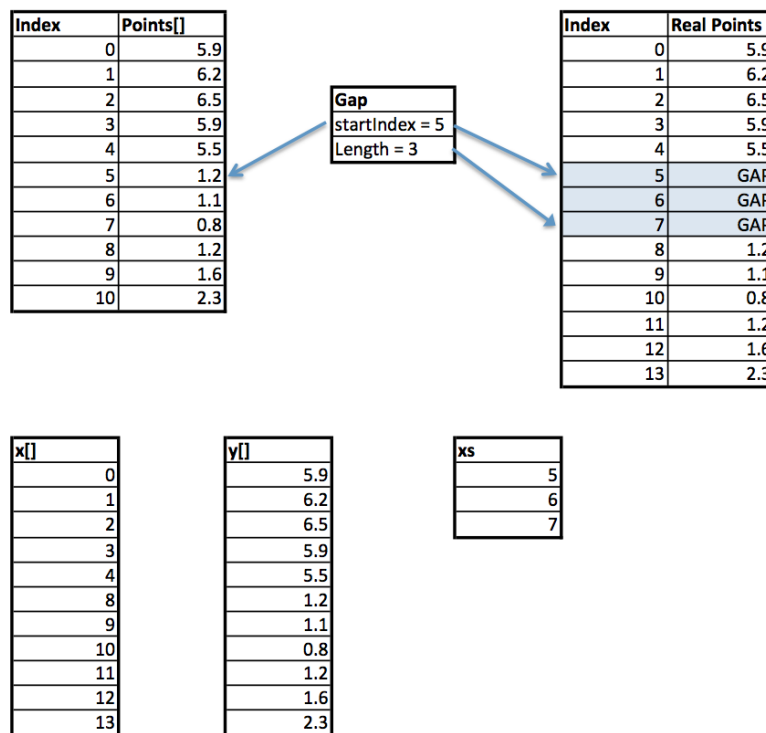


FIGURE 5.25: Relation between figure 5.21 and cubic interpolation function's parameters

The interpolation isn't use to create new real data points but to create a smooth transition between the existing data points. For the biggest blobs, we decided to make a transition to zero to avoid points that doesn't make any sense. The constants of this class define the maximum points to interpolate, the number of good points taken before and after the gap and the number of points (T1) that will create the transition to zero (see figure 5.26).



FIGURE 5.26: Interpolation - T1 parameter

This will result in a muted part in the audio file because the derivative of these points will be zero.

5.1.4 GUI

All the basic parameters and function are implemented in the top section of PRISM (red square in figure 4.2). The figure 5.27 represents the tabs available for the new blob clean function.

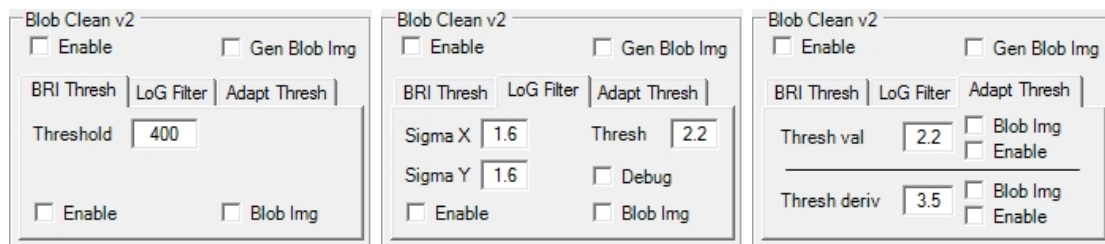


FIGURE 5.27: BlobClean v2 GUI

The function has to be enabled with the top left check box. The top right check box gives the option to add the general blob image to the detailed view.

Each blob detection algorithm can be enabled. This has been done for parameter tuning. When adjusting the parameters of one algorithm, the user doesn't have to compute all of them. For each blob detection algorithm, it is possible to create and add the blob image in the detailed view.

The others parameters correspond to those explained before in this chapter.

For the blob detection, some parameters are hidden in the menu *option/BlobClean2 - more parameters*. Those parameters should not be changed for most of the recordings. The figure 5.28 shows the dialog box that appears when the user accesses to this menu.

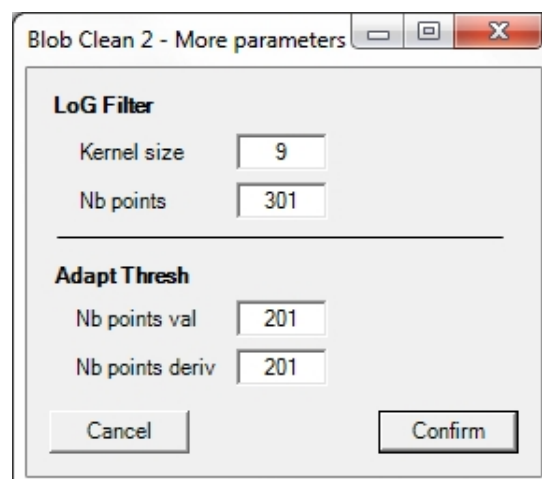


FIGURE 5.28: BlobClean v2 - More parameters

5.2 Replacement of muted part with "silence"

As explained before in this chapter, we decided to mute the sound when the blob is too big. After some tests and analysis, it appears that those muted parts were not pleasant for the auditor. This phenomenon is comparable to telephone's communications. In those communications, there is always a background noise even if nobody is speaking to let the users know that the connection is still set.

The goal of this function is to detect the background noise and to replace the muted part with it so the user can not hear the gap.

5.2.1 Algorithm

The figure 5.29 represents the general idea of the algorithm. It uses the fact that most of the samples can be used as background noise. Even when some music is playing or somebody is talking, the samples values go by the lowest one. We use this to detect the longest part with only low samples values. This part is assumed to be background noise. However this algorithm wasn't good enough because even if we take 70% of the samples, the noise can vary around the threshold value resulting in a short silent part that were repeated to fill the muted part. This created strange noise in the audio file.

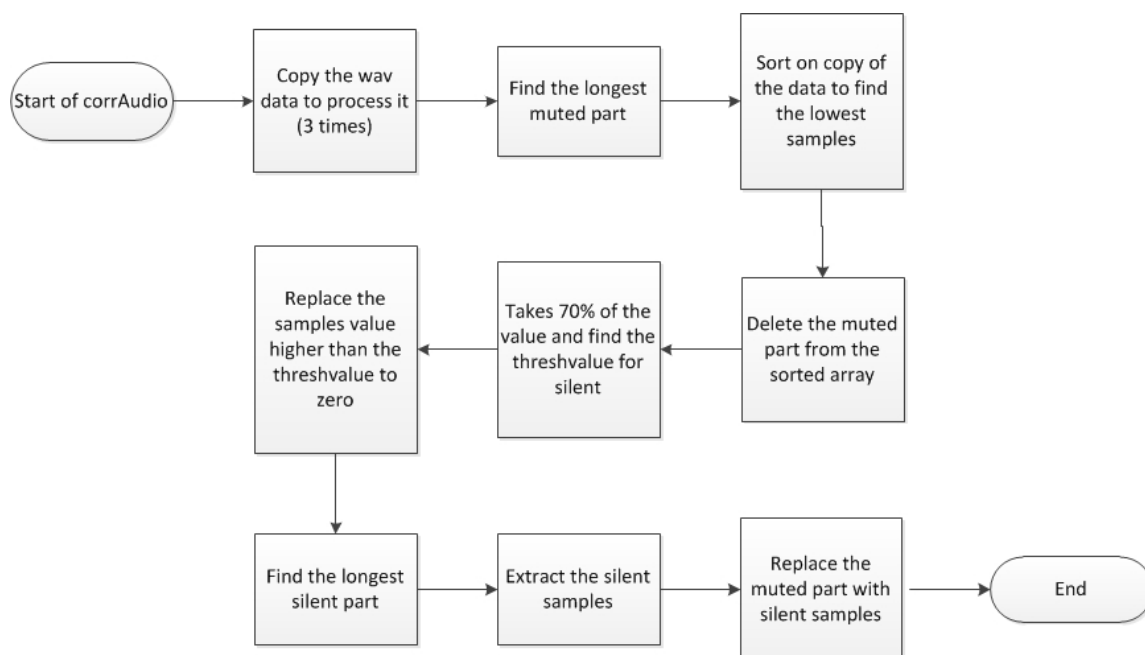


FIGURE 5.29: Replacement of muted part with "silent" algorithm

To correct this, we decided to set a maximum gap value between two silent part when we detect the longest silent part. If the gap is smaller, the samples that were higher than the threshold value are integrated to the silent part. The maximum gap size starts at 15 samples and is incremented by 5. This is computed until the longest silent part is equal or longer than the longest muted part (with a limit of 5 iterations, which means that the maximal gap is 35 samples). If the longest silent part found is smaller than the muted one, we repeat it to fill the gap. The figure 5.30 represents the algorithm to find the longest silent part.

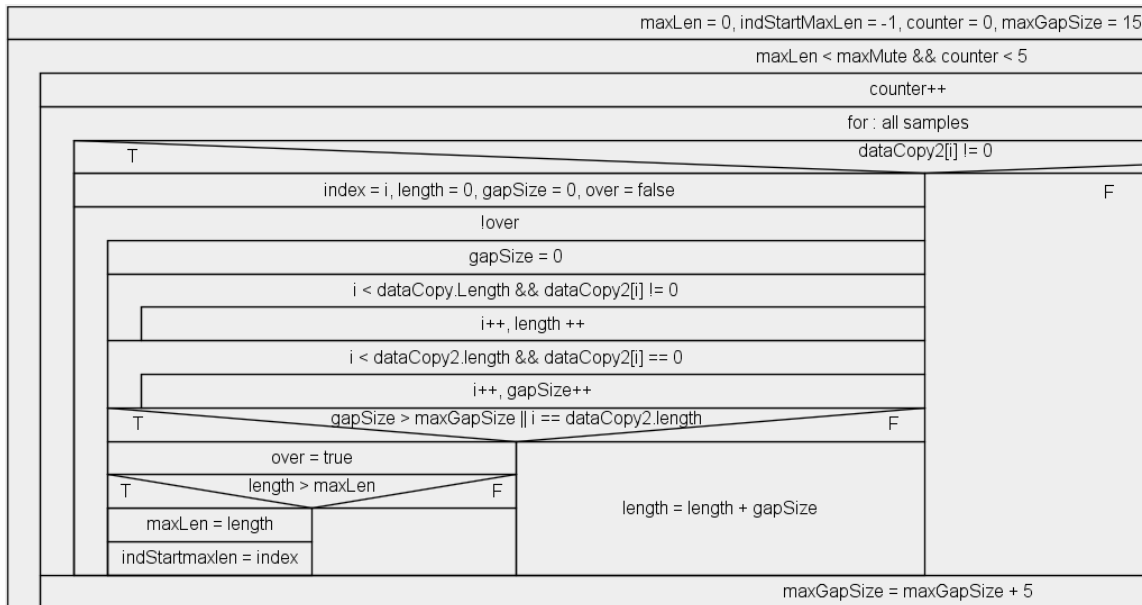


FIGURE 5.30: Find the longest silent part algorithm

We added a function called `corrAudio` in the class `Utils.Wav` taking in parameter the data that will be written into the wav file and returning a new array with the corrected data.

```

public int[] corrAudio(int[] data)
{
    ...
}

```

This algorithm can be activated in the GUI with the check box Audio Correction (figure 5.31).



FIGURE 5.31: Audio correction - GUI

Chapter 6

Tests and validation

This chapter describes all the tests' procedures and an analysis of the results. We tested all the implemented algorithm and compare the results in order to find the best set of parameters. As second test, the audio quality will be compared with the previous BlobClean function. We proceeded most of the test on every records. However we presents only the interesting results in this chapter. For every record, we just tested a part of them because of the time it takes to create the tracking and to process the blob cleaning.

6.1 Tests description

LoG with 2 different sigmas

The goal of this test is to compare the results of the LoG algorithm when using 2 different sigmas and define if this might be a subject of further development in order to automatize these parameters. First we will use the record 287881 and if the results are good enough, we will proceed the test on all the other records.

The test will consist of opening 3 different rings of the record. One in the border, one in the middle and one in the center. For each rings, 7 different combinations of sigmas will be tested and compared.

- $\sigma_x = \sigma_y = 1.0$
- $\sigma_x = 1.6, \sigma_y = 1.0$
- $\sigma_x = 2.8, \sigma_y = 1.6$
- $\sigma_x = \sigma_y = 1.6$
- $\sigma_x = 1.6, \sigma_y = 2.8$
- $\sigma_x = 1.0, \sigma_y = 1.6$
- $\sigma_x = \sigma_y = 2.8$

BlobDetect algorithms

The goal of this test is to verify that the implemented blob detection algorithms work on all the available records.

6.1.1 Replacement of muted part with "silence"

The goal of this test is to verify that the implemented algorithm works on all the available records. Only the final audio file with and without this option activated will be analyzed.

6.1.2 Audio extraction

The goal of this test is to compare the quality of the final audio file between the previous algorithm and the new ones. This is usually done by testing the algorithm with our own record. However this is not possible and will result in a purely subjective comparison. The criteria will be the noise level and its characteristics and the number of clicks presents on the records. This test will be run for every available record.

6.1.3 Tracking

During the tests' procedures, it appears that the interactive tracking was less efficient than the manual tracking. It creates oscillations in the tracking resulting in clicks and artifacts in the final audio file (figure 6.1).

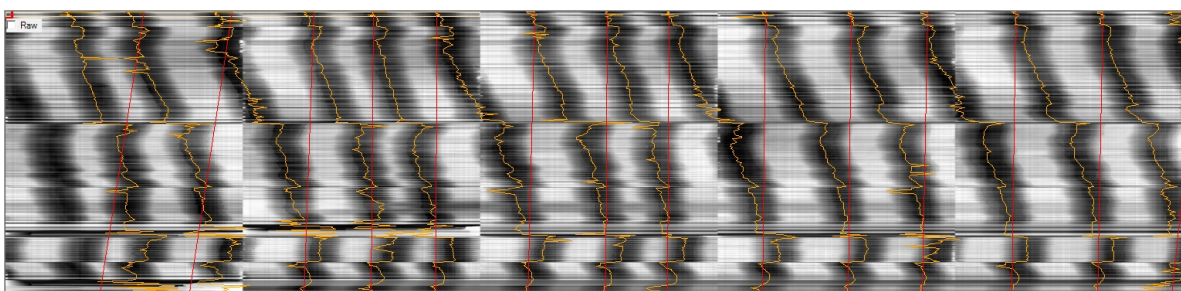


FIGURE 6.1: Interactive tracking

This is the reason we only used the manual tracking tool for the tests.

6.2 Results

LoG with 2 different sigmas

This test shows that it is not a good idea to use two different σ for the Laplacian of Gaussian algorithm. The blobs are deformed after the convolution with a kernel created with two different σ . The figures 6.2 and 6.3 show the blobs before and after the convolution, we can see how the blobs are deformed in the axis where the σ is bigger. We decided to let the this option in the code and GUI in case of further development.

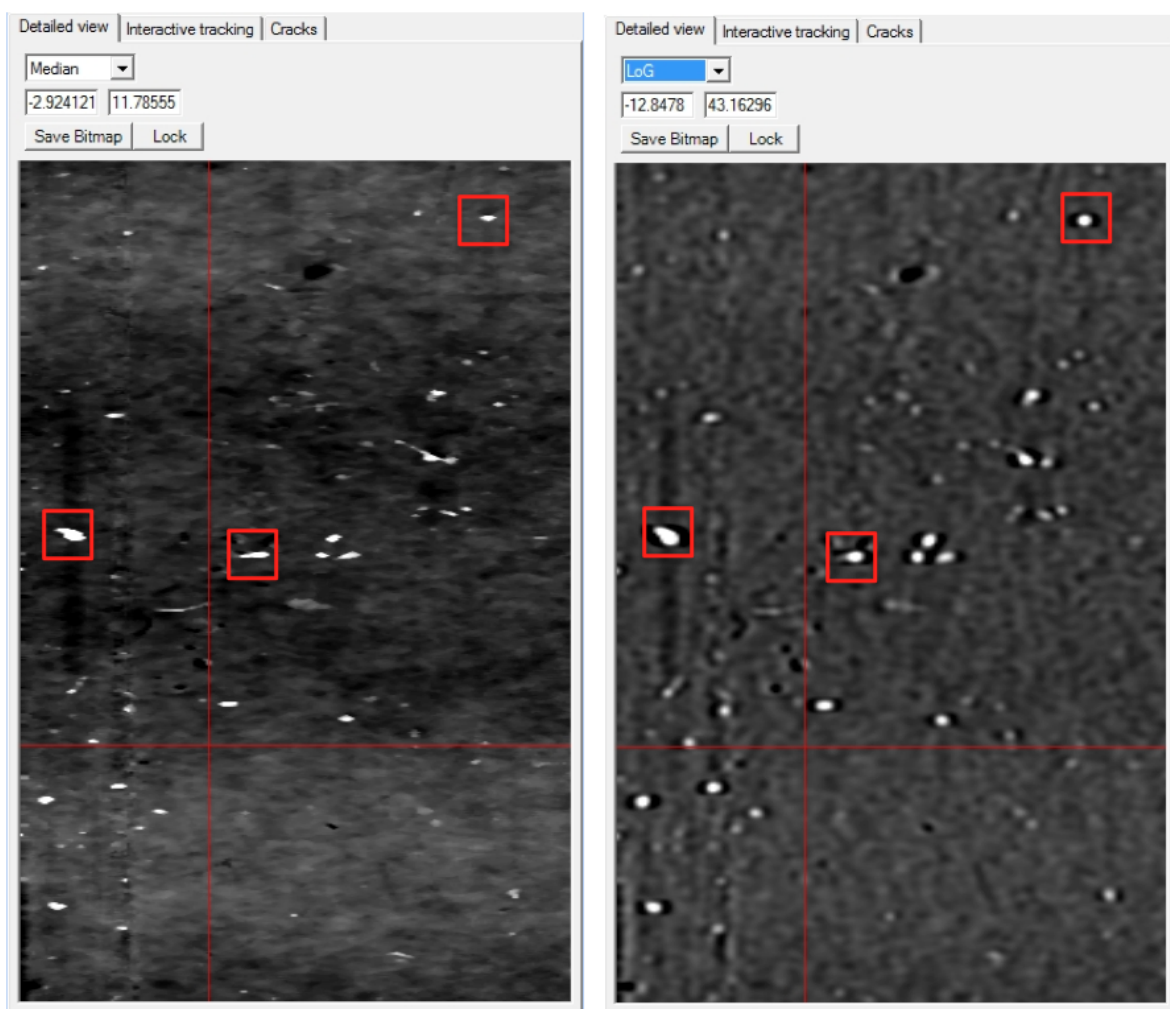


FIGURE 6.2: Results of LoG filter with $\sigma_x = 1.6$ and $\sigma_y = 2.8$

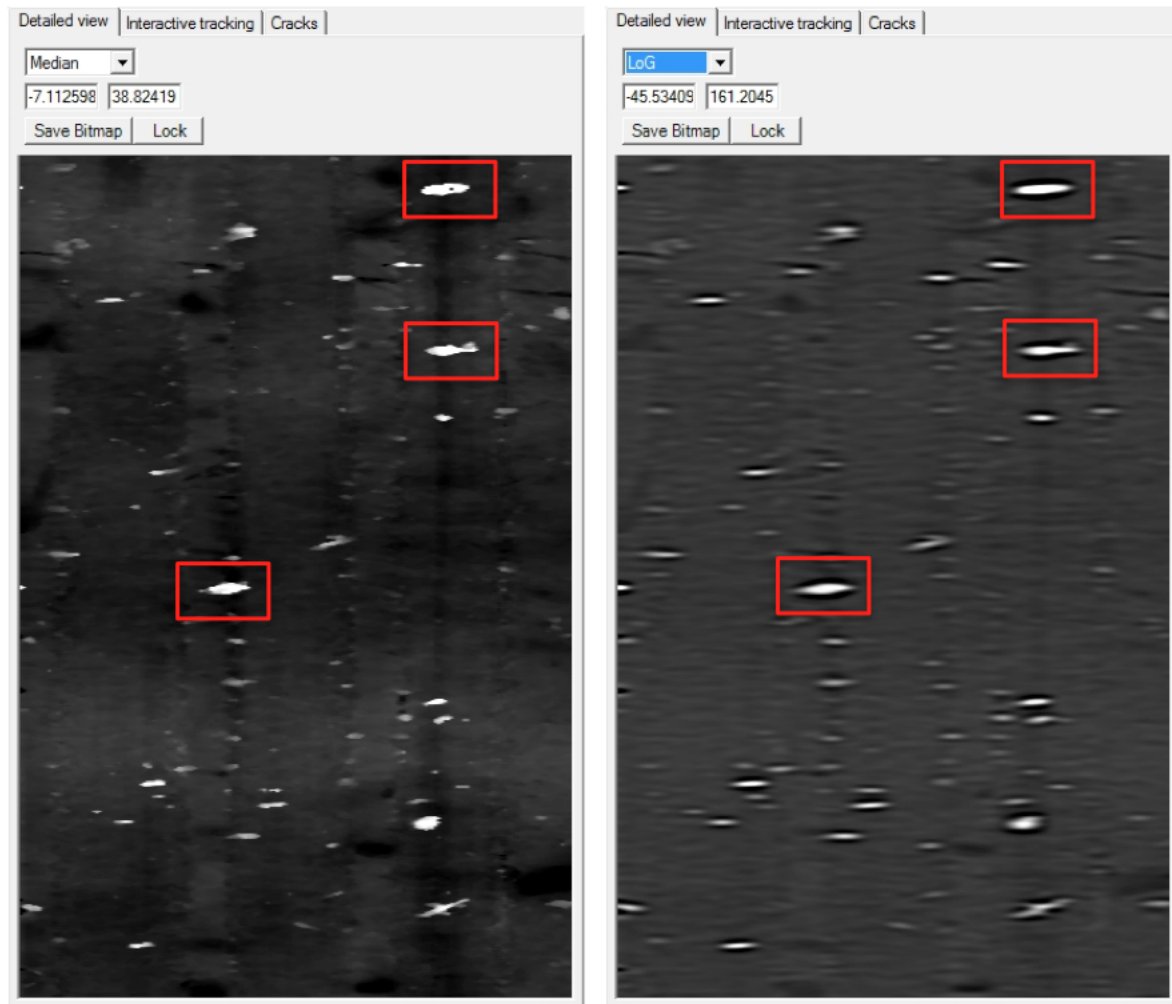


FIGURE 6.3: Results of LoG filter with $\sigma_x = 2.8$ and $\sigma_y = 1.6$

This deformation is normal, it appears because of the kernel (see figure 5.6). Even if the blobs aren't represented in the picture as they really appear on the record, the Laplacian of Gaussian is an image-processing function that doesn't take into account this fact. It detects the blobs independently of their shape.

This test also demonstrates how the σ parameter influences the results. It correspond to the level of the Gaussian blur (low-pass filter) that is applied before the Laplacian operator. The figures 6.4, 6.5 and 6.6 represent the results of three different σ (1.0, 1.6, 2.8). We found that the best value is 1.6. With 1.0, the low-pass is too "soft". It doesn't cut the low frequencies of the picture enough which will create issues with the next step of the algorithm (The main goal is to enhance the blobs and reduce the background to be able to extract them afterward). With 2.8, the low-pass is too "strong" which makes the small blobs disappear (figure 6.6).

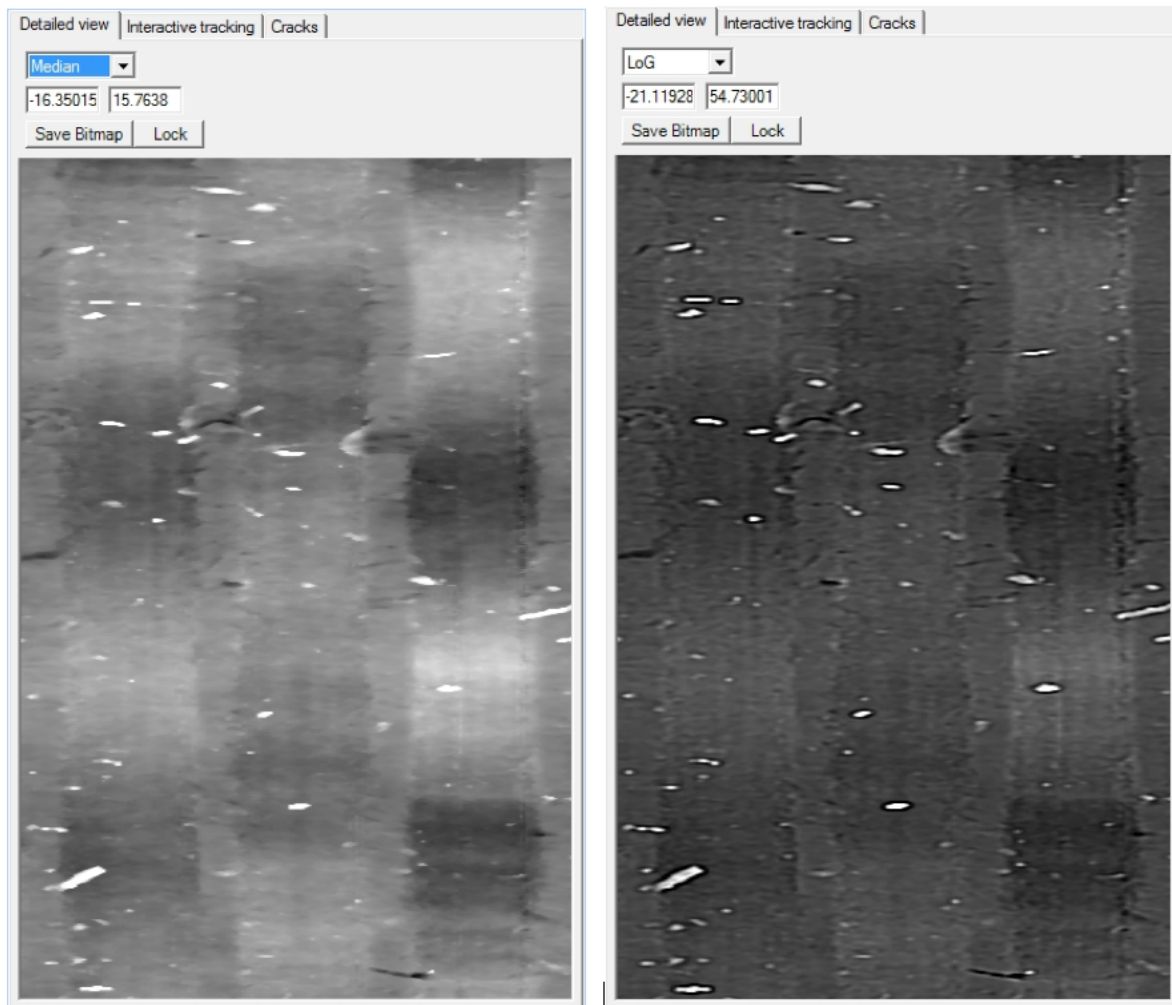


FIGURE 6.4: Results of LoG filter with $\sigma_x = 1.0$ and $\sigma_y = 1.0$

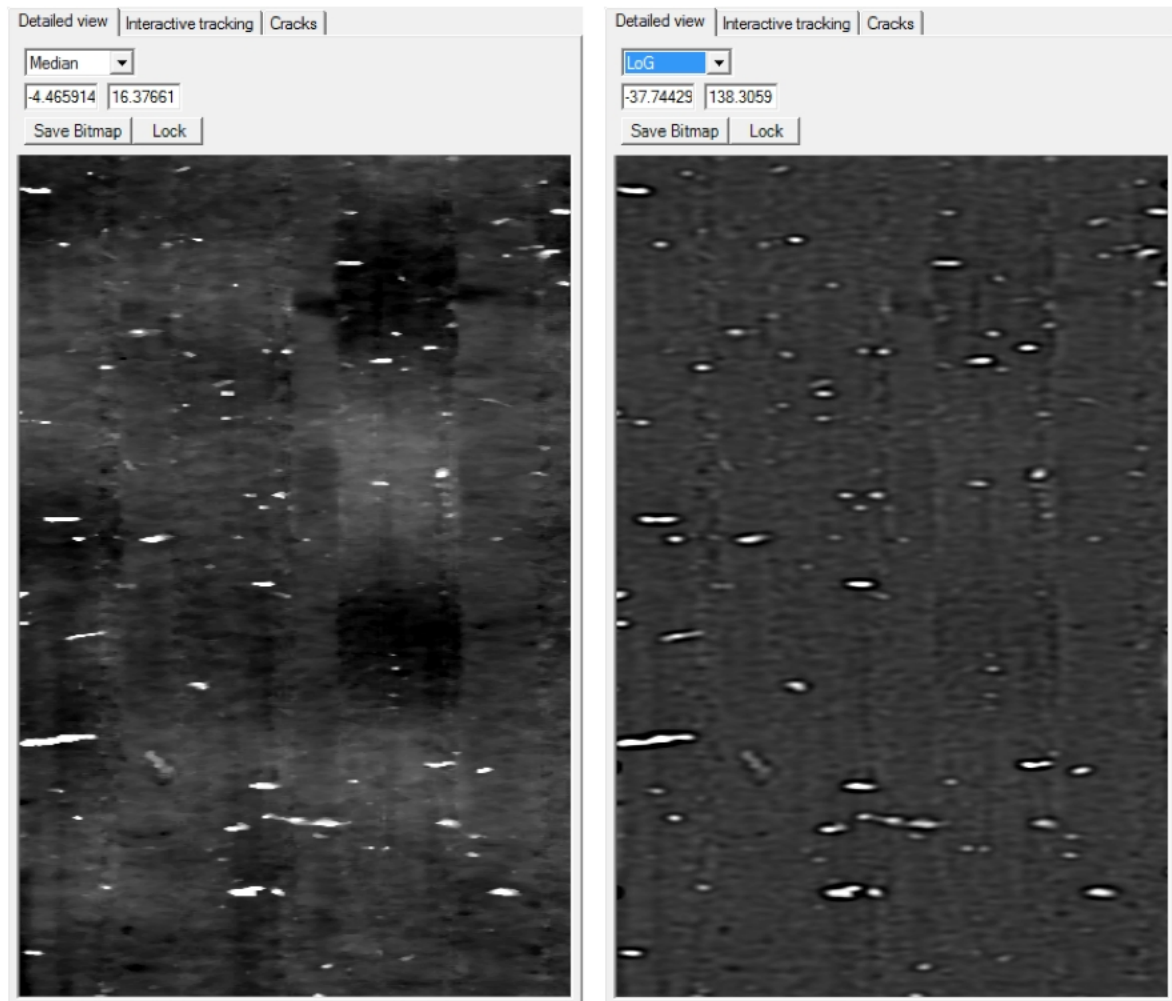


FIGURE 6.5: Results of LoG filter with $\sigma_x = 1.6$ and $\sigma_y = 1.6$

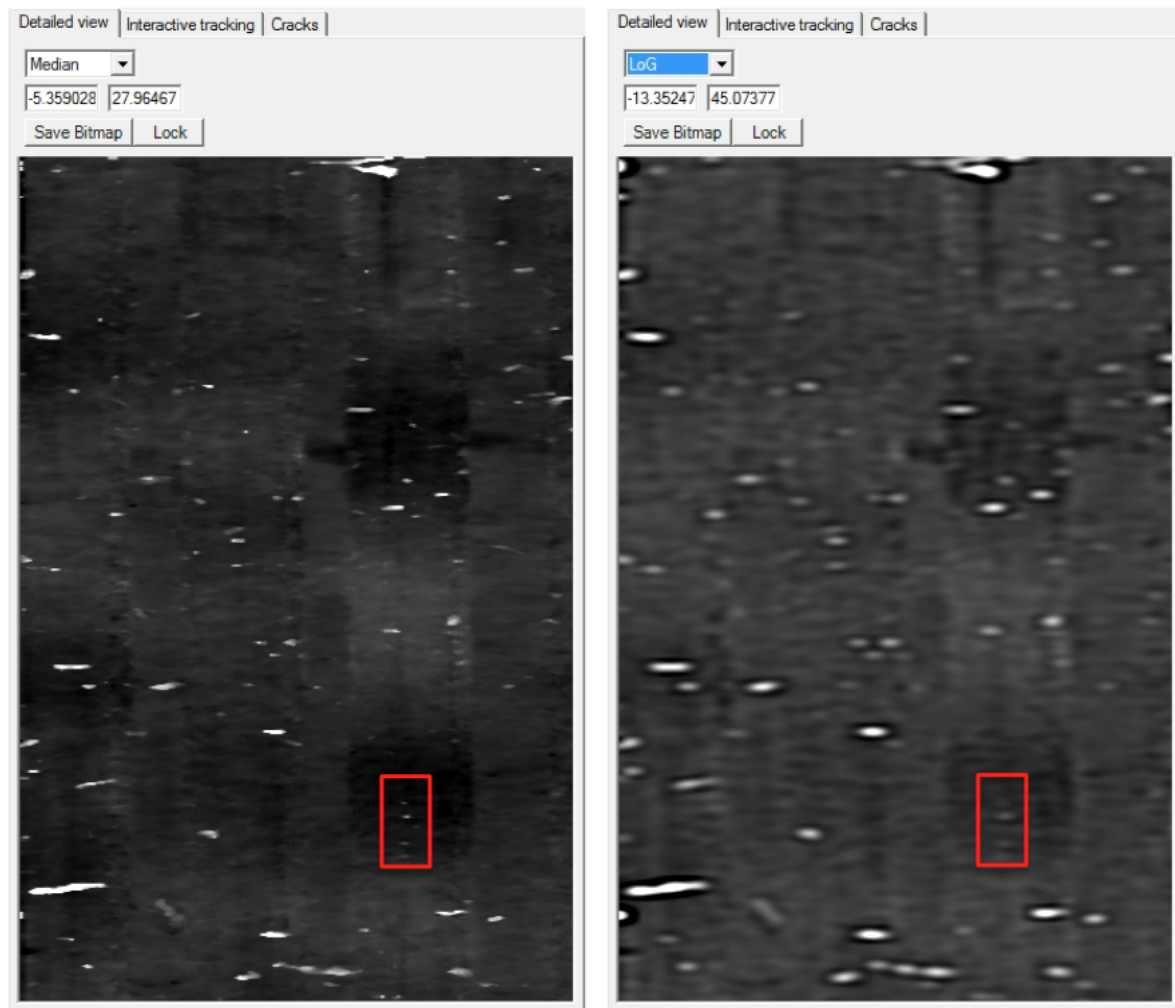


FIGURE 6.6: Results of LoG filter with $\sigma_x = 2.8$ and $\sigma_y = 2.8$

The red square shows small blobs that almost disappear due to the Gaussian low-pass filter. We decided to take a value of 1.6 for σ_x and σ_y for all the others tests.

6.2.1 Blobs detect algorithms

The implemented algorithms works for all the records. This section presents the different results and how the parameters affect them.

BRI algorithm

This algorithm detects the probe lost of focus (figure 6.7) and cracks (figure 6.8). The only parameter is the threshold value. This value is almost the same for every record and is around 400. If the value is to high, the algorithm will be to sensitive and detect to much blobs (figure 6.9).

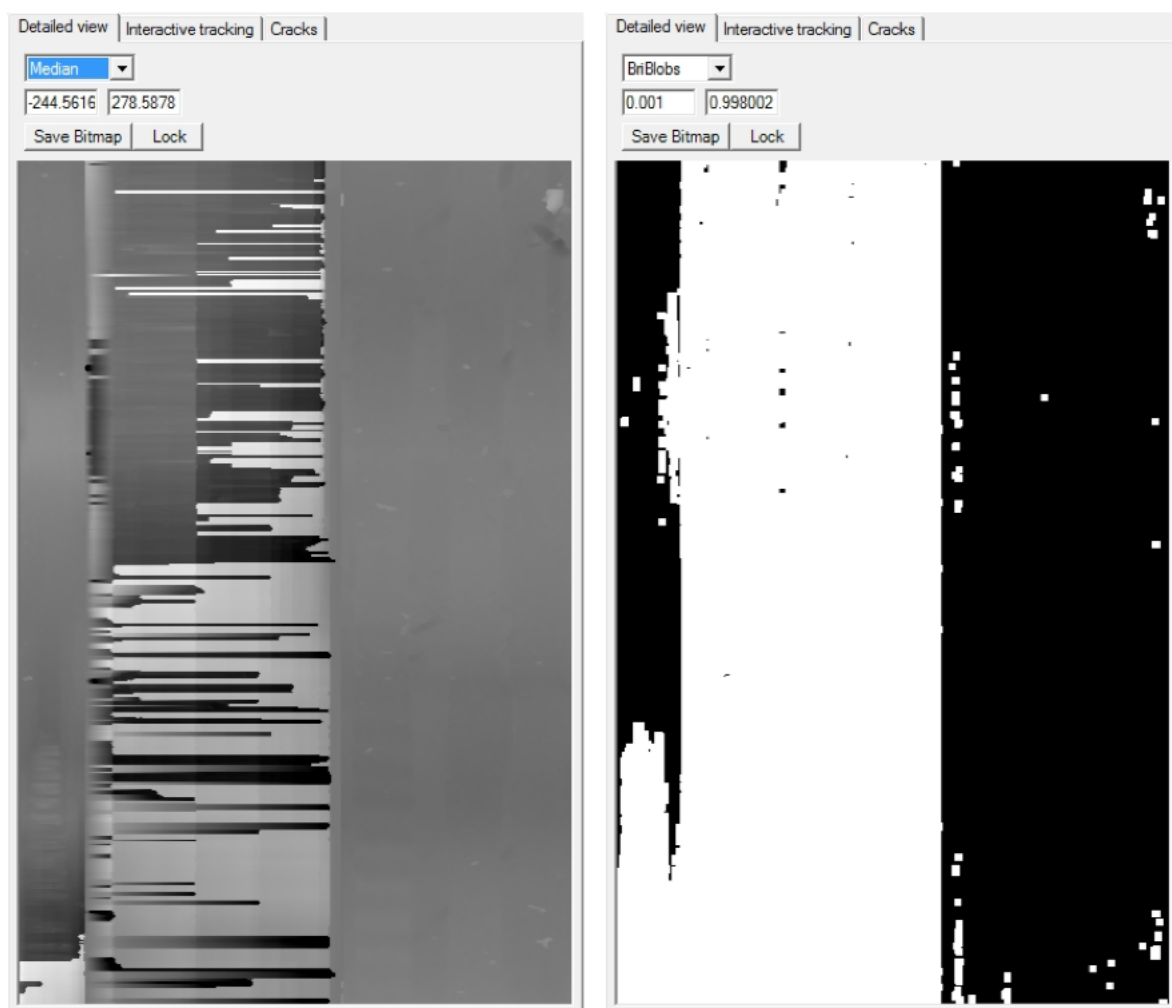


FIGURE 6.7: BRI Algorithm - lost of focus detection

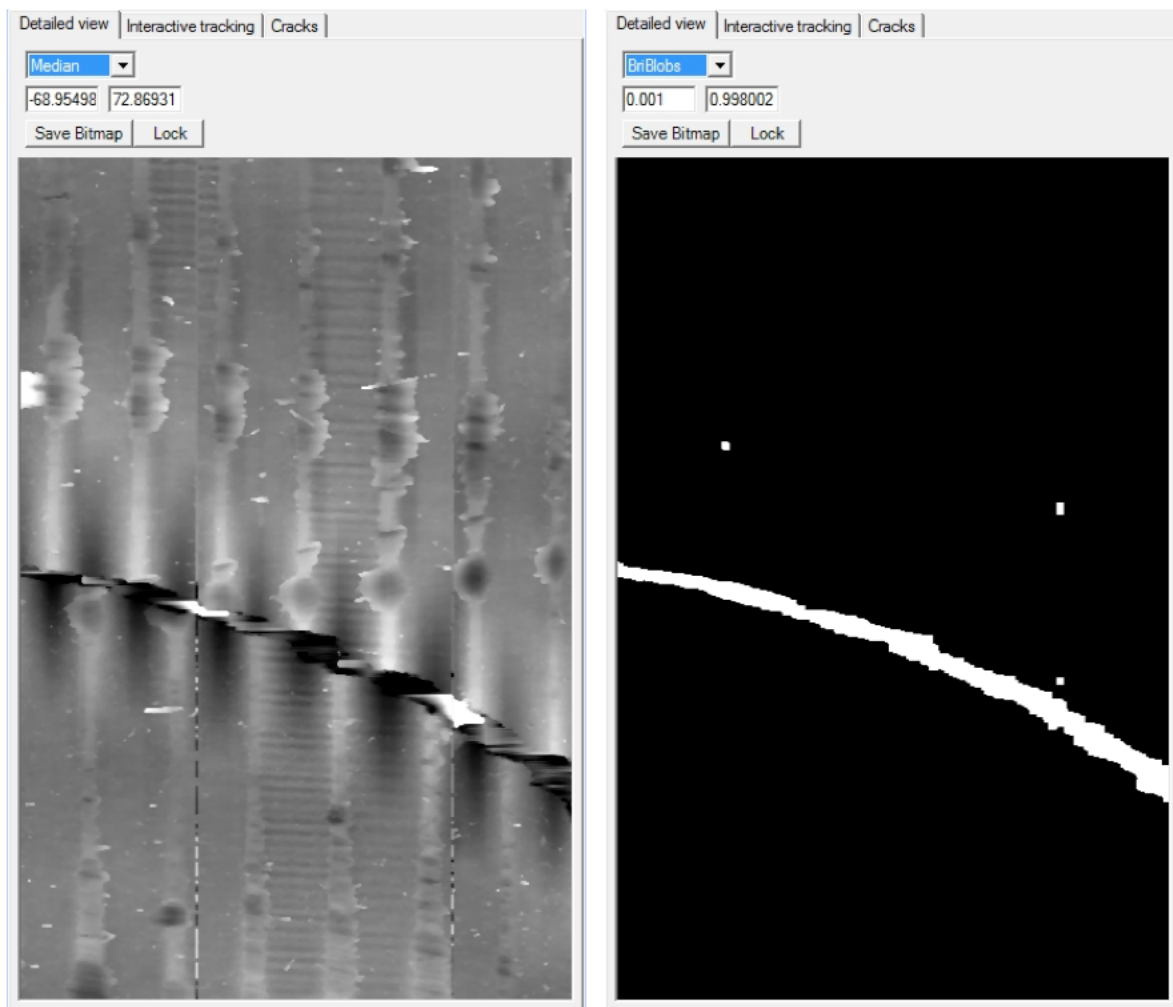


FIGURE 6.8: BRI Algorithm - cracks detection

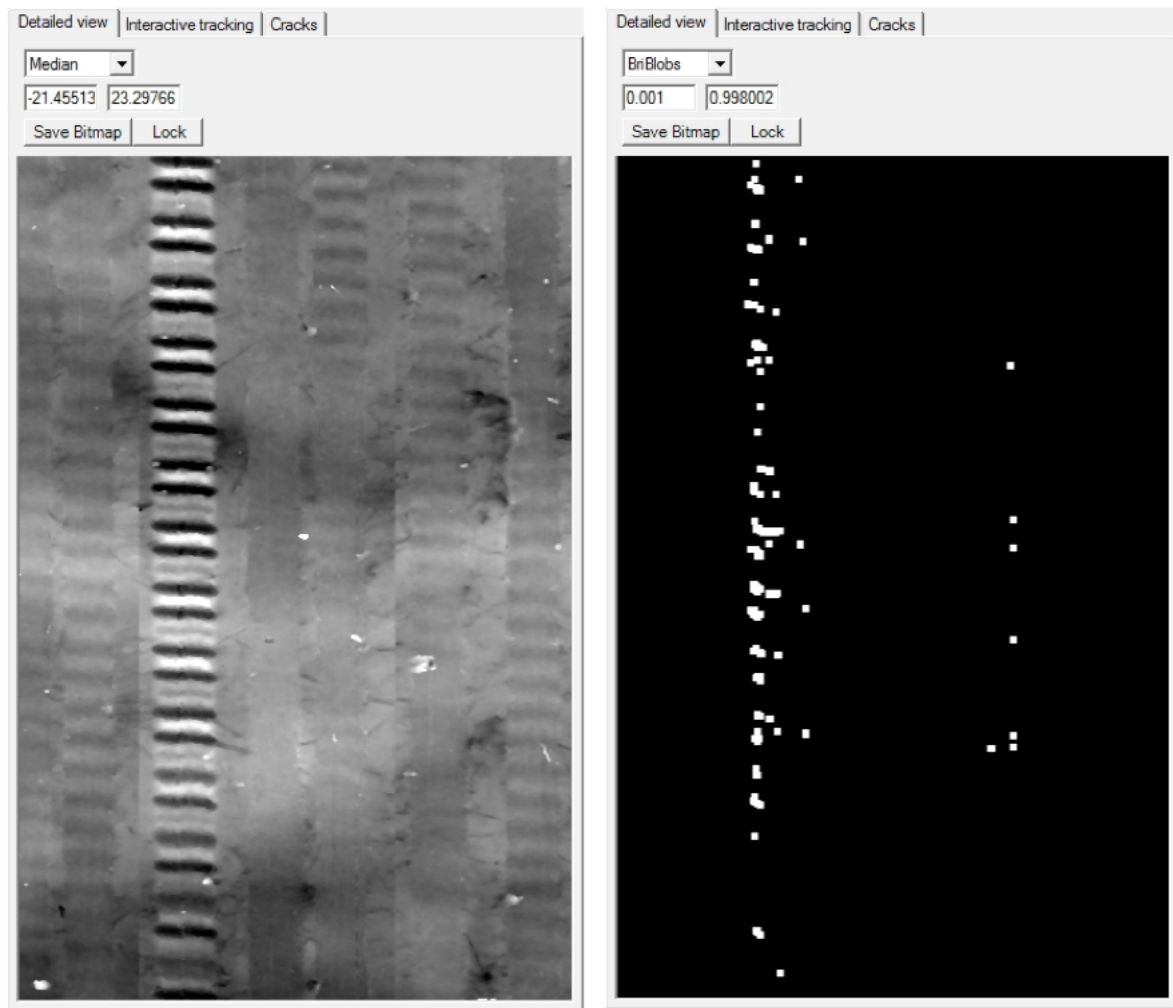


FIGURE 6.9: BRI Algorithm - too sensitive threshold

In the figure 6.7 we can see that it miss a blob filling function. This could be a possibility of further development.

6.2.2 LoG algorithm

As explained before, the σ_x and σ_y parameters influence the Gaussian low-pass filter. For these tests, we chose a value of 1.6 for both of them. The other parameter is the threshold value used in the adaptive threshold algorithm (see chapter 5 for more information) and represent the sensitivity of this algorithm. The user has to be careful that the grooves aren't detected as blobs. The figure 6.10 shows the LoGBlobs with a threshold value of 2.4. We can see that the algorithm detects a groove as a blob.

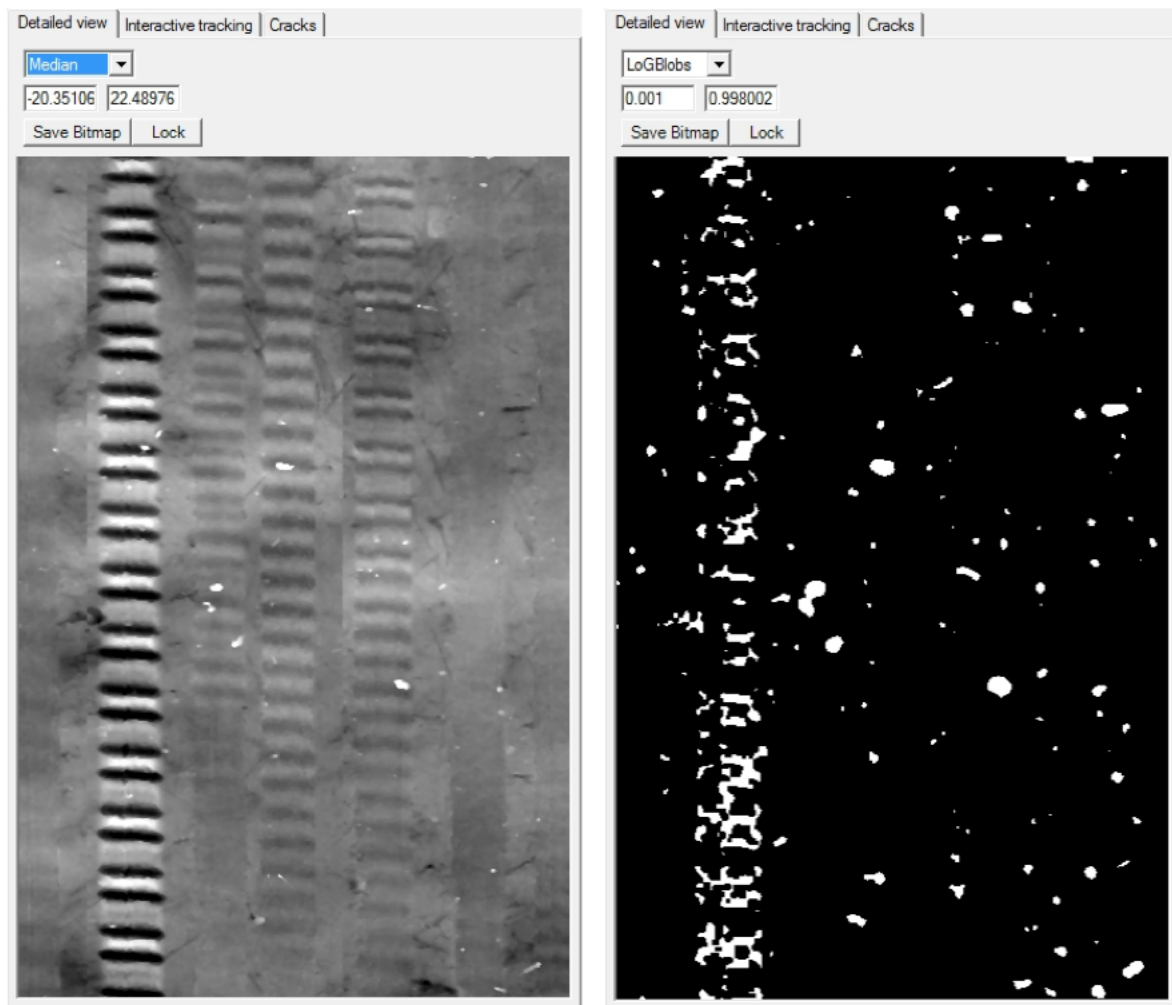


FIGURE 6.10: LoG Algorithm - Too sensitive threshold

This algorithm is used to detect all kind of blobs, cracks and probe's lost of focus. However the results weren't good enough to let only this algorithm handle the complete blob detection process.

The figure 6.11 shows the detected blobs on a part of the record 287881 with a threshold value of 2.8.

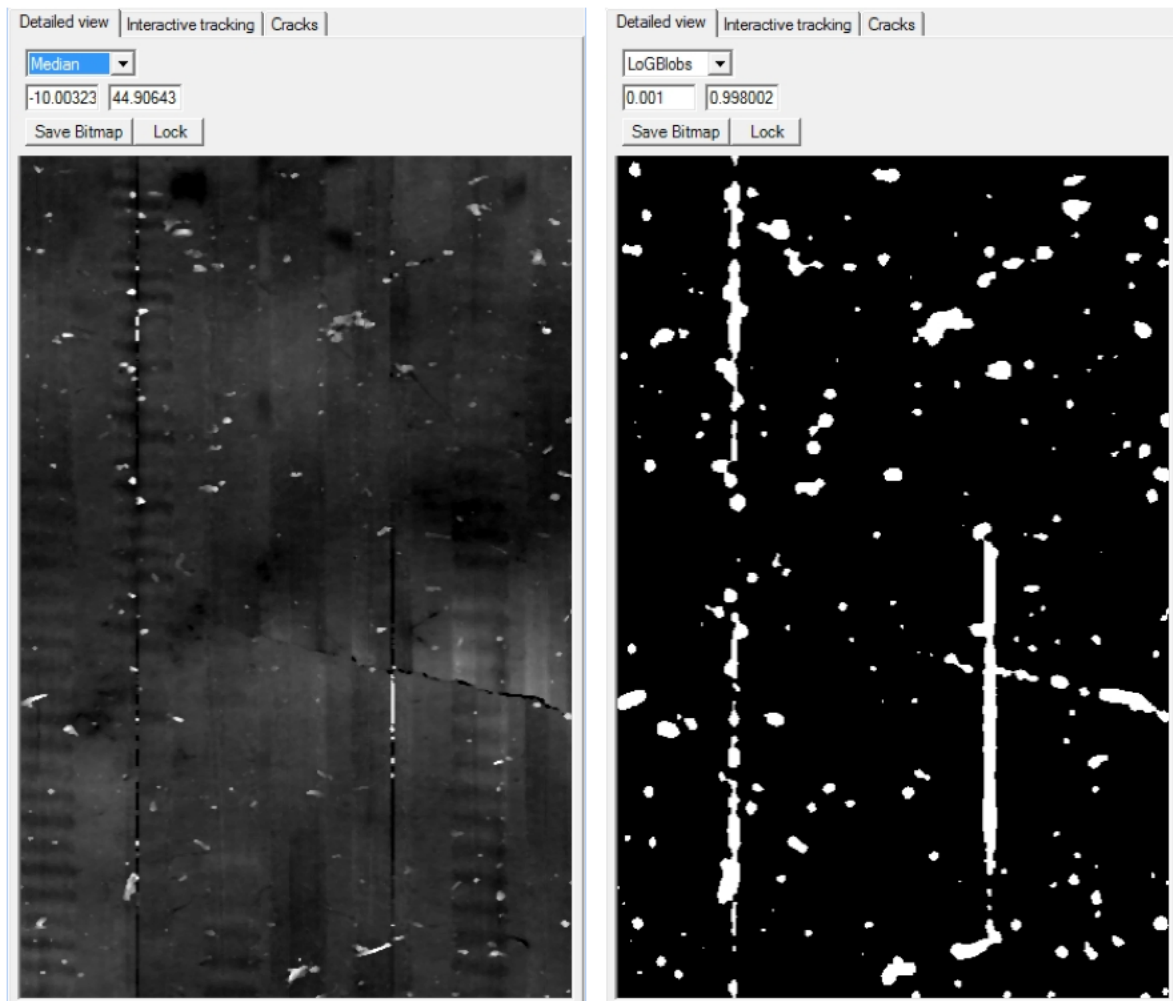


FIGURE 6.11: LoG Algorithm - Threshold = 2.8

We can see that the detected blobs are a little bit bigger than in the record's picture. This is due to the algorithm (averaging filter). We decided that it was better to detect blobs bigger than they are instead of smaller.

6.2.3 Adaptive threshold algorithms

AdaptThreshVal

This algorithm is very powerful to detect the blobs (figure 6.12). However it fails when it comes to detect cracks (figure 6.13) and probe's lost of focus (figure 6.14). The use of an adaptive threshold makes it very strong on every kind of record. Moreover with the new version, the computation time is reduced by a factor 6 (figure 6.15). The only parameter of this function is the threshold. It modifies the sensitivity of the algorithm.

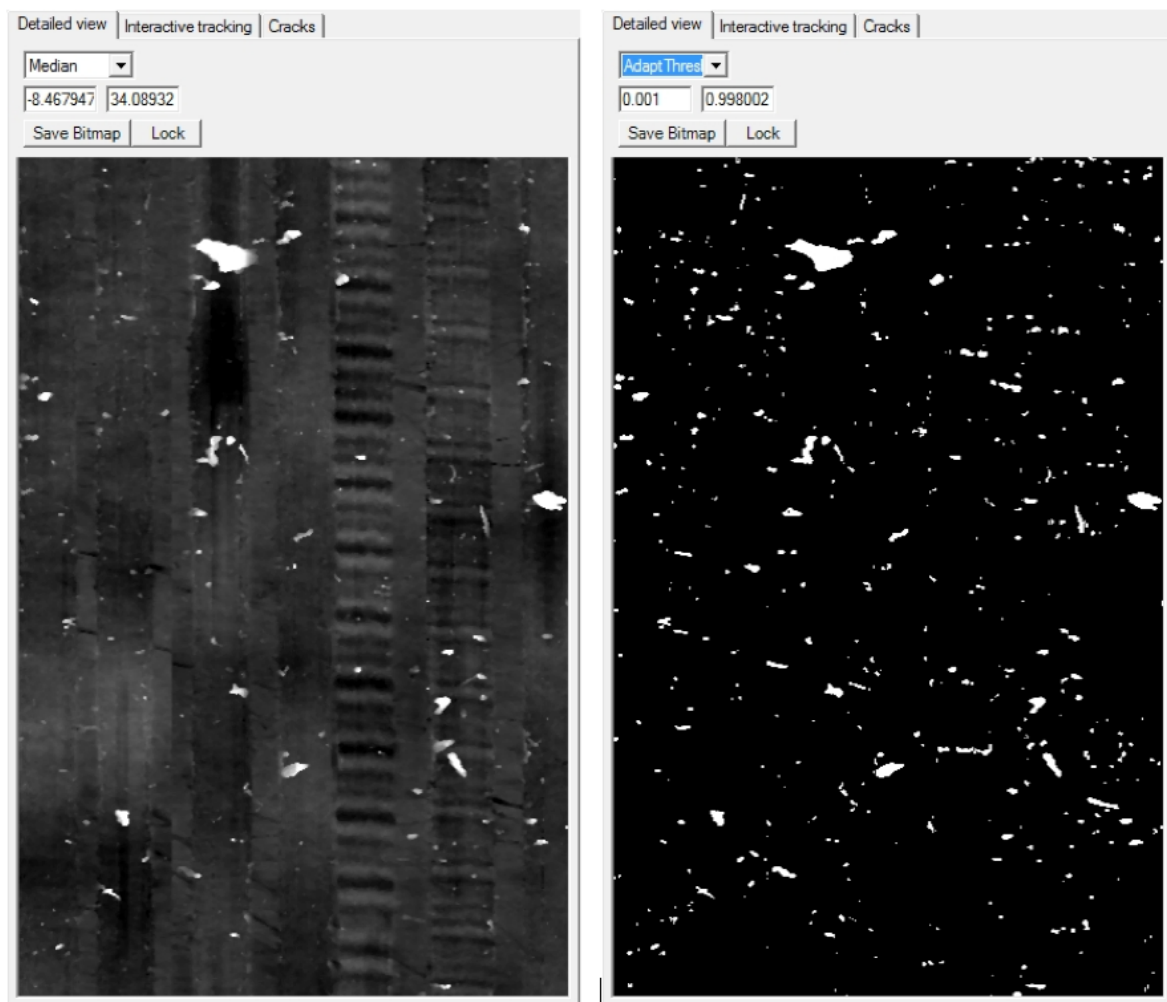


FIGURE 6.12: Adaptive threshold Algorithm (value)

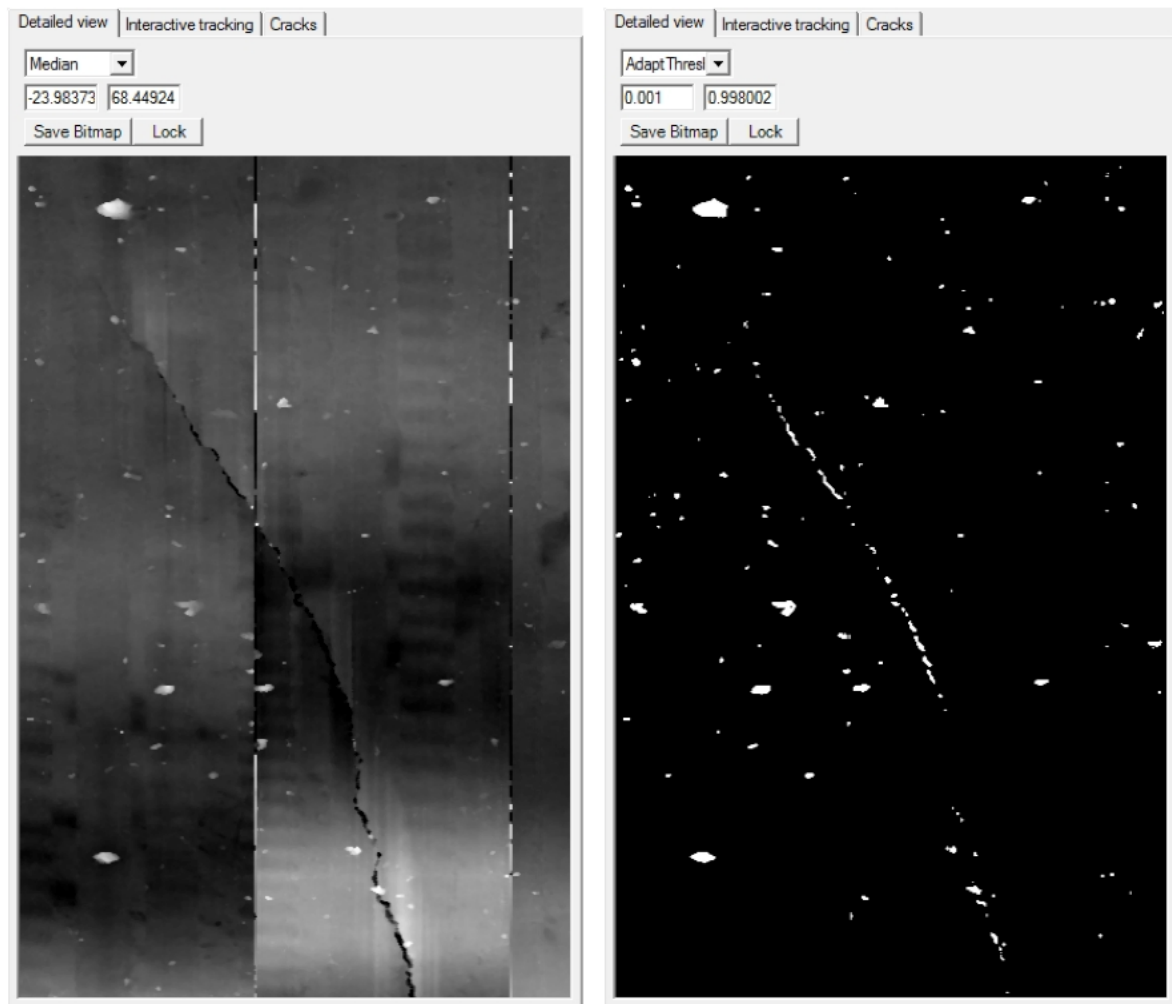


FIGURE 6.13: Adaptive threshold Algorithm (value) - Cracks

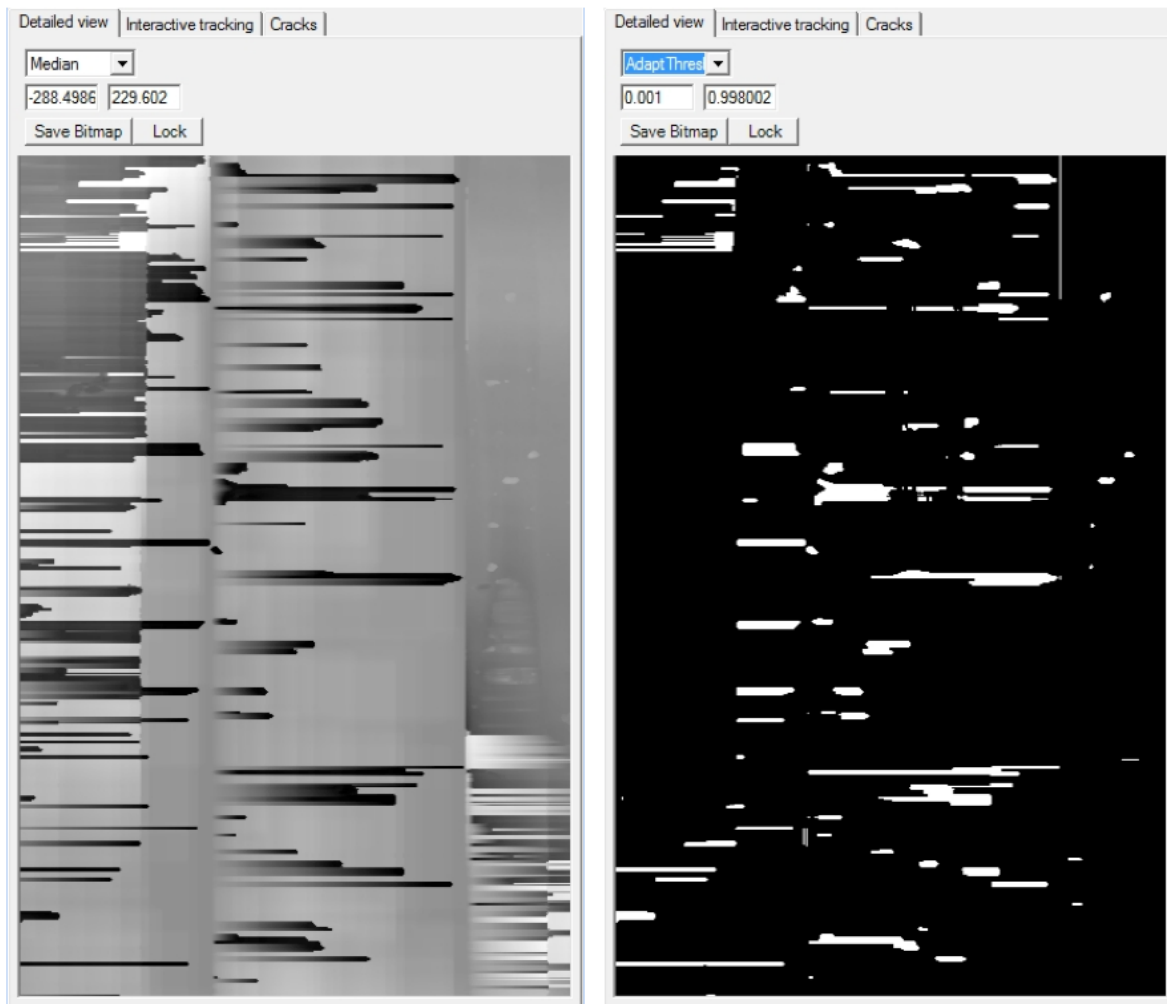


FIGURE 6.14: Adaptive threshold Algorithm (value) - Lost of focus

The figure 6.15 shows a comparison between the computation time of this algorithm. On the left, the equation 4.5 was used and on the right, the equation 4.6 was used. The top images are the computation for 3 rings and the bottom one are for 6 rings. We can see that the new algorithm is 6 times faster.

Select File: 2.57 sec Load Image: 3.71 sec Clean Image: 67.81 sec Create Image: 0.58 sec Smooth Bin Image: 0 sec Create BMP interactive: 0.42 sec	Select File: 1.95 sec Load Image: 3.74 sec Clean Image: 10.89 sec Create Image: 0.59 sec Smooth Bin Image: 0 sec Create BMP interactive: 0.42 sec
Select File: 2.39 sec Load Image: 7.44 sec Clean Image: 138.12 sec Create Image: 1.11 sec Smooth Bin Image: 0 sec Create BMP interactive: 0.87 sec	Select File: 1.64 sec Load Image: 7.43 sec Clean Image: 22.24 sec Create Image: 1.09 sec Smooth Bin Image: 0 sec Create BMP interactive: 0.87 sec

FIGURE 6.15: Adaptive threshold Algorithm (value) - Computation time for 3 and 6 rings

AdaptThreshDeriv

This algorithm detects the edge of the blobs (figure 6.16). It can be used in combination with the AdaptThreshVal algorithm to circle the blobs in order to be sure the detected blobs are big enough. Without any other algorithm, this function is useless because it doesn't fill the blobs.

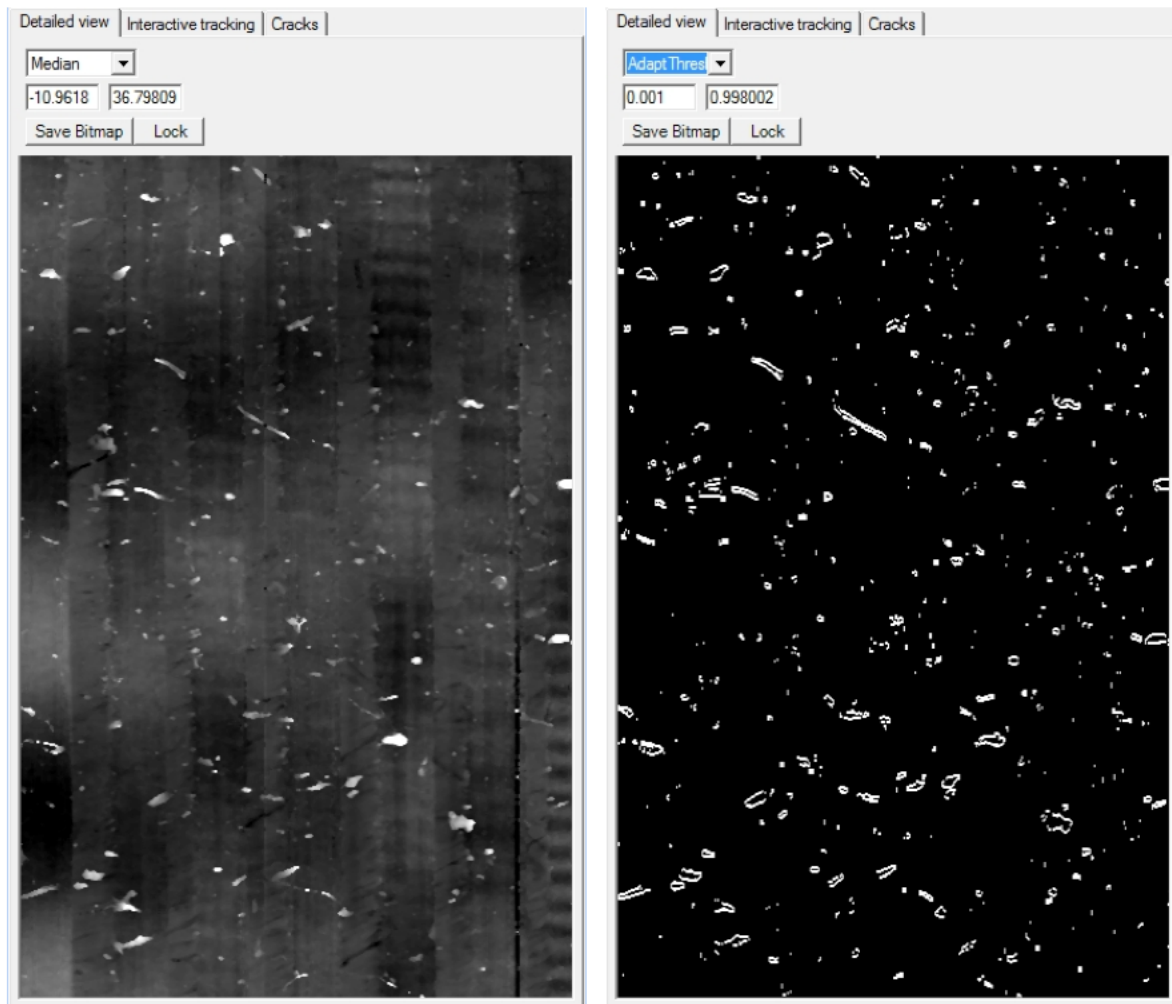


FIGURE 6.16: Adaptive threshold Algorithm (derivative)

The figure 6.17 shows the detected blobs with the two adaptive threshold algorithms. We can see the result is almost perfect. However some blobs aren't detected completely and these functions do not detect the cracks and probe's lost of focus.

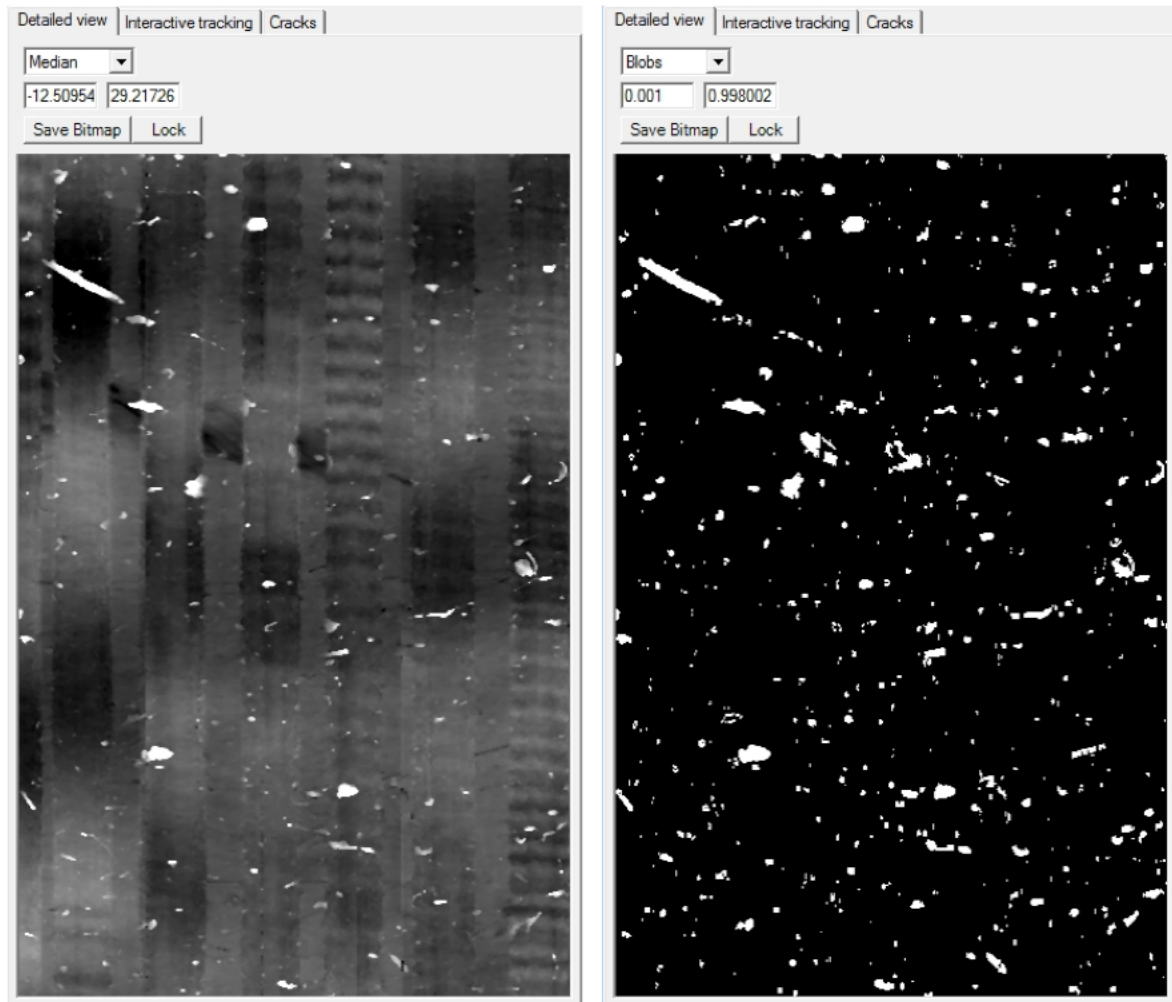


FIGURE 6.17: Adaptive threshold Algorithms

6.2.4 Complete blobs cleaning algorithm

As we expected, the implemented cleaning algorithm is totally independent of the record type. We decided to present some results on different records with it. The figure 6.18 shows a perfect cleaning of the image. The big gray area are the biggest blobs that we set to zero.

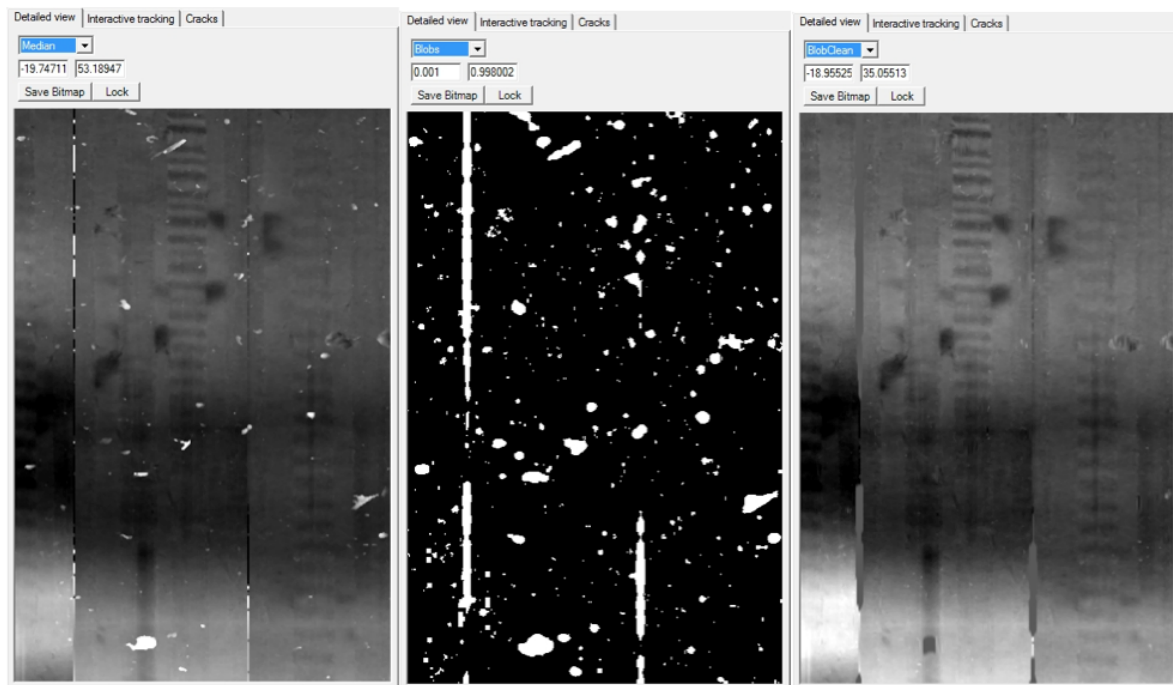


FIGURE 6.18: Blob Clean v2 result

Moreover, the cubic spline interpolation won't create clicks in the final audio file like the linear interpolation.

However, the new blobs cleaning function isn't perfect all the time. The figure 6.19 shows a part of the record 287881 with uncleaned blobs. This is due to the fact that some blobs were not detected in full, creating strange points during the interpolation. This results in uncleaned blobs even if most of it was detected.

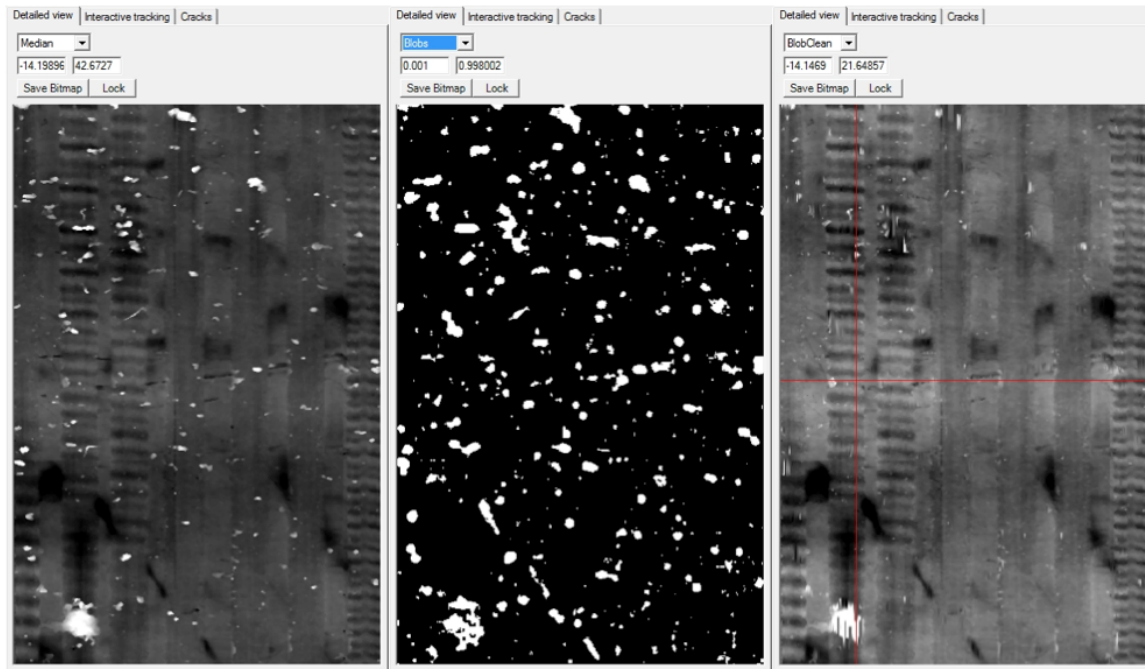


FIGURE 6.19: Blob Clean v2 result 2

The figure 6.20 shows the points value under the vertical red line in the right picture of figure 6.19. The green line represent the points' values before the cleaning and the red one after the cleaning.

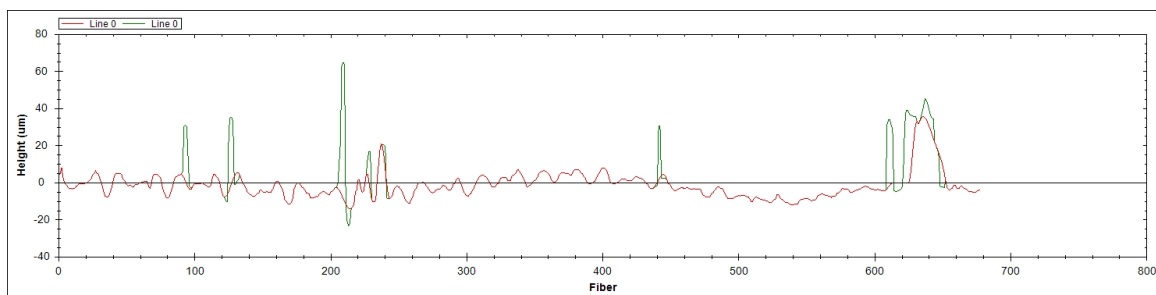


FIGURE 6.20: Blob Clean v2 result 2 - graphic

We can see that only one undetected points can mess up the blob cleaning process. As explained in chapter 4 a blob is a region that differ in properties compared to areas surrounding it. If the blob doesn't have constant properties, it might not be completely detected.

This fact also takes into account the blobs represented in figure 6.21. Those blobs are scratches that destroyed the record's wax surface.

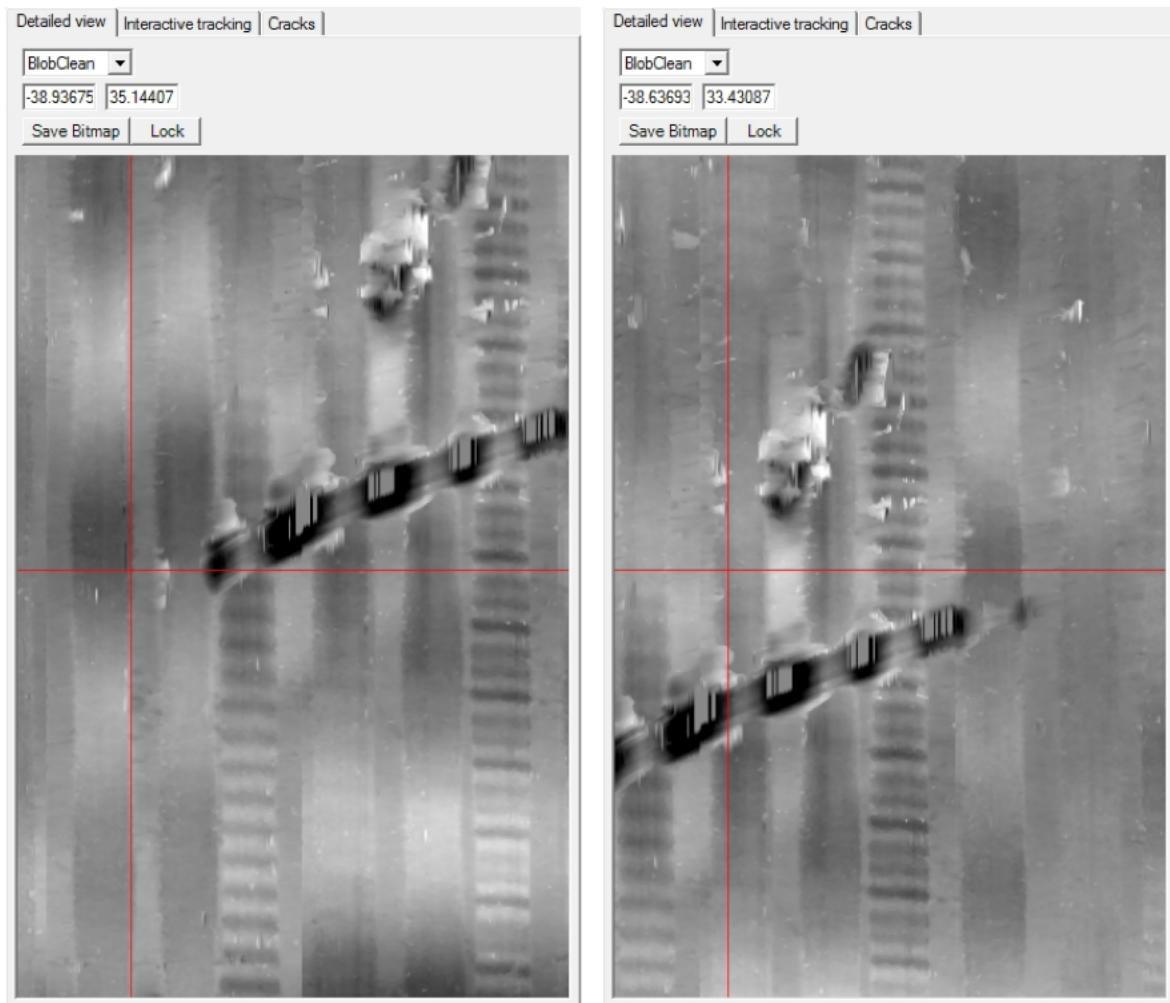


FIGURE 6.21: Blob Clean v2 result 3 - undetected blobs

We can't do anything about those scratches, however for the small undetected blobs, the outlier function in the PRISM software will handle it. It takes the groove's points and compares the mean value with all the groove's pixels. If a pixel differs from more than the specified value, the pixel is considered as an outlier and is deleted. The mean is computed again and so on till no pixel differs or until the minimum number of pixel left is reached (parameter that can be set in the GUI).

Computation time

We decided to test the computation time of every algorithm. The time displayed represents the real processing time of the function. We changed the code in order to bypass the median filter and the cleaning process. It was taken from the console of the GUI (Clean Image).

Algorithm	3 rings	6 rings	9 rings
BRI	11.58 sec	23.46 sec	35.88 sec
LoG	42.46 sec	93.98 sec	136.58 sec
AdaptThreshVal	2.25 sec	4.82 sec	7.22 sec
AdaptThreshDeriv	2.93 sec	6.15 sec	9.47 sec

TABLE 6.1: Computation time

We can see that the computation time is in $o(n)$ with n representing the number of rings to process. The LoG takes more time to compute because of the convolution operator. An improvement would be to implement these function with concurrent programming in order to use all processor's cores.

When the user selects more than one algorithm, the total process time is the sum of each algorithm's computation time.

Note that in the final code, the median filter and cleaning process are included in the cleaning time displayed in the GUI's console.

6.2.5 Replacement of muted part with "silence"

For this test, only the audio file was analyzed in Sound Forge software.

This test was performed on the record 287881 only, because it is the only one that had lost of focus data creating big blobs and muted part in the audio. In the next figures, we shows the audio waveform with and without this function. When we ear the audio file with the muted part, it s very difficult to understand what's on the record because the sound is always cut. This is corrected by this algorithm. It improves the final audio quality of the file because the noise level is constant. During this test we discovered that this function could create clicks at the border of the muted part. We didn't manage to correct this because of the time constraint.

Record 287881

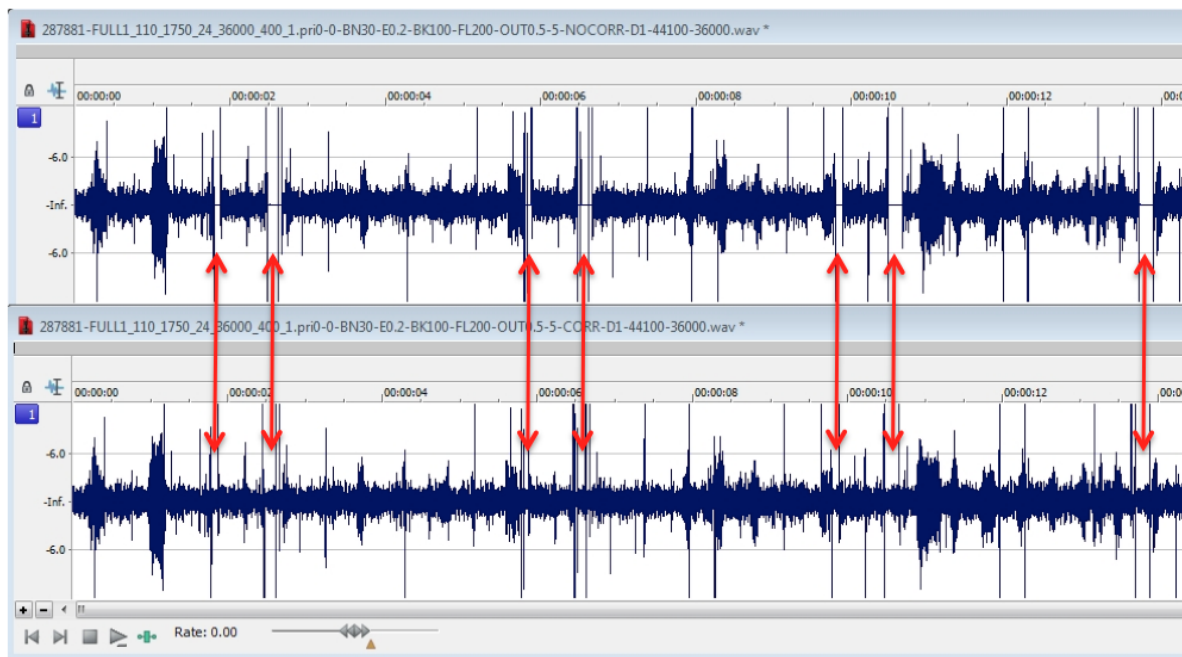


FIGURE 6.22: Record 287881 - Replacement of muted part with "silence"

6.2.6 Audio extraction

For all the figure of audio waveform, the top one is without any blob clean function, the middle one is with the previous blob clean function and the bottom one is with the new blob clean algorithm. All the audio files were extracted with the same tracking to not falsify the results. We compared the audio waveform and the frequencies spectrum of the available records. Those records are all vertically cut. We applied a low pass filter in order to delete the frequencies higher than $\frac{F_{sample}}{2}$.



FIGURE 6.23: Records 287700, 287701, 287860.2, 287881

Record 287700

This record contains a lot of cracks (figure 6.24, the red squares represent them). These cracks make the audio quality very bad. Moreover, the noise level on the record is high.

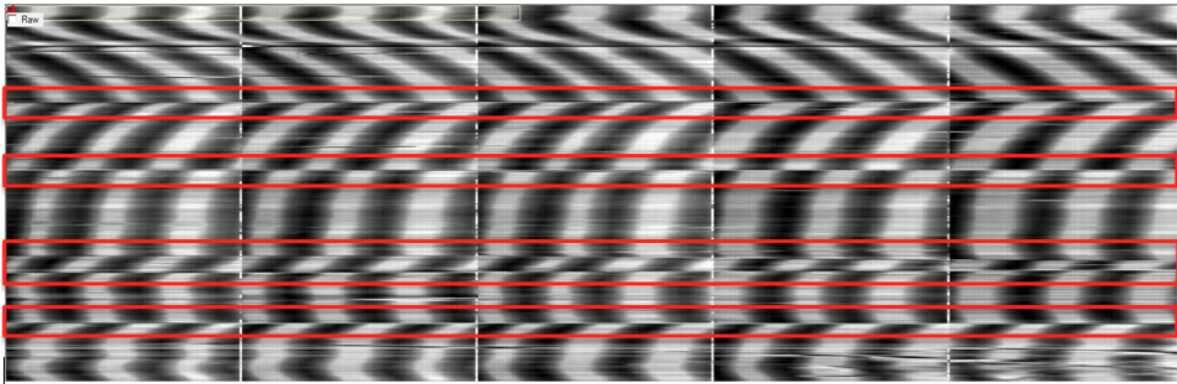


FIGURE 6.24: Record 287700 - cracks

On the audio waveforms (figure 6.25) we can see that the extracted file has a bad quality.

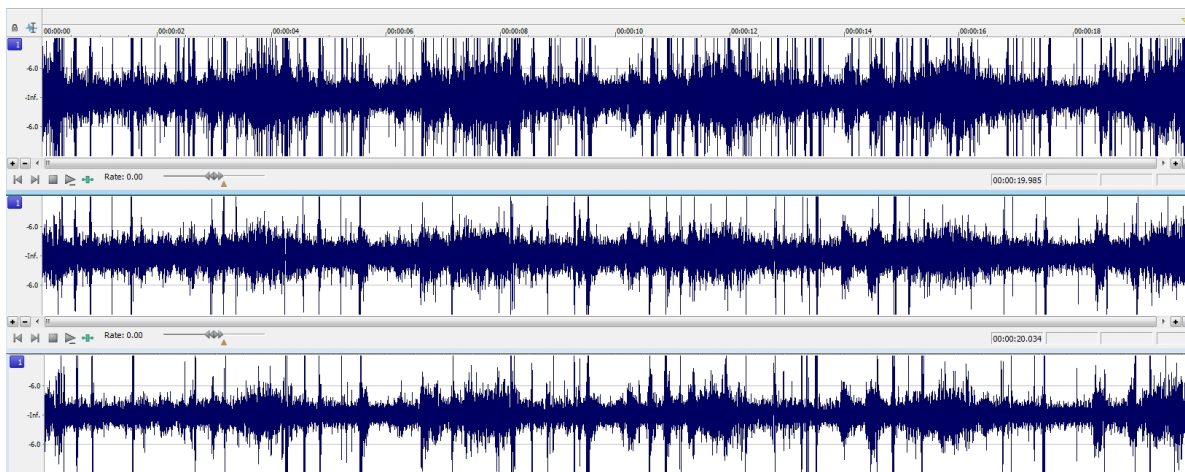


FIGURE 6.25: Record 287700 - waveforms

In the frequency spectrum (figure 6.26) we can see that the new algorithm is quieter than the previous one. There are less clicks due to the blobs and interpolation.

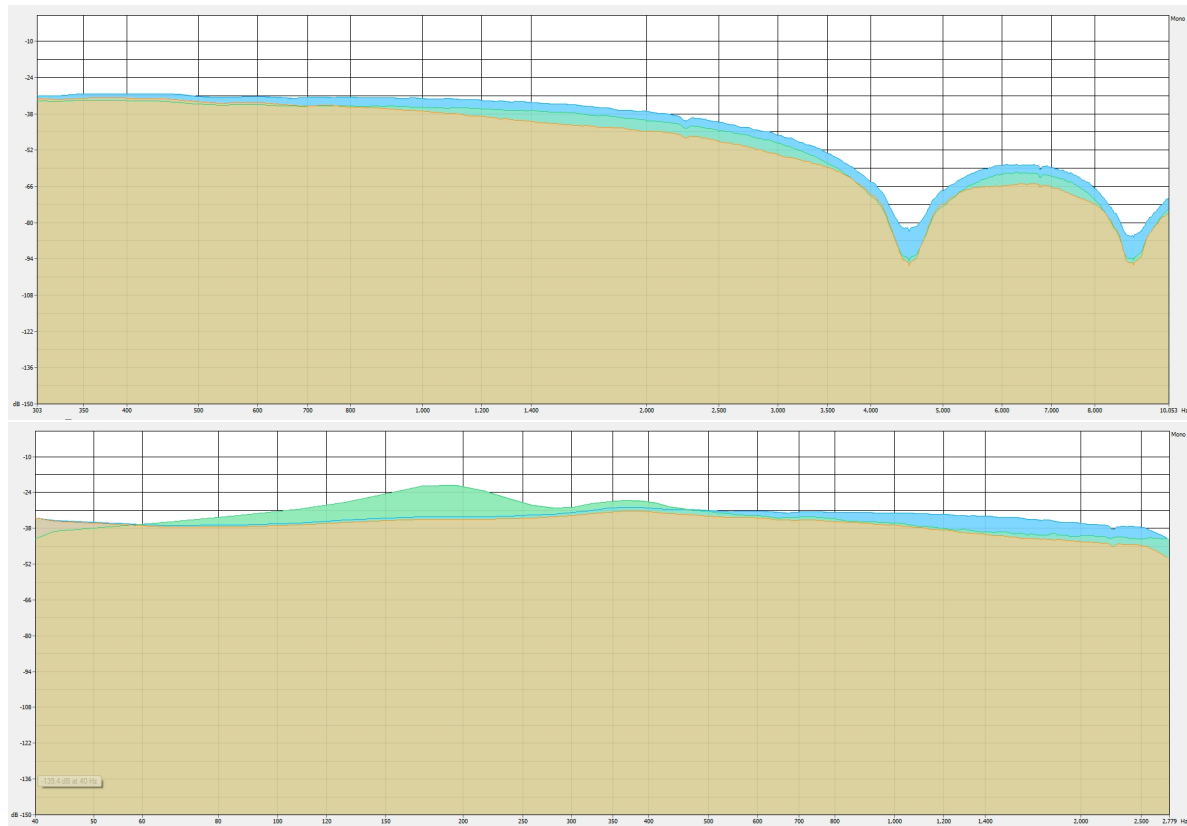


FIGURE 6.26: Record 287700 - frequency spectrum - Blue = no blobs clean / Green = old blobs clean / Yellow = new blobs clean

We can see the effect of the sample frequency of the disc (low pass filter around 4.5kHz).

The noise level and cracks make this record almost inaudible. New tools may have to be developed to restore these kind of recordings.

Record 287701

The figure 6.27 shows the waveform of the three different audio file. The bad quality of the audio file made this record not understandable.

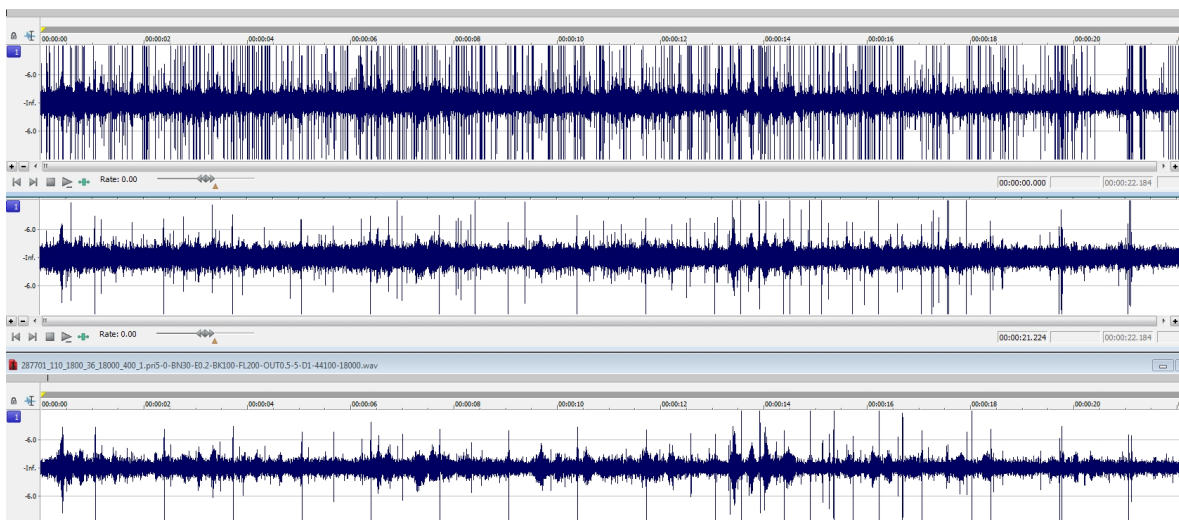


FIGURE 6.27: Record 287701 - waveforms

We can see that the first waveform contains a lot of clicks due to the cracks, dust and dirt particles present on the record. The new algorithm reduces the number of clicks present in the audio file. However some of them are still present (see the Remaining clicks section of this chapter). The figure 6.28 shows the frequencies' spectrum of these audio files.

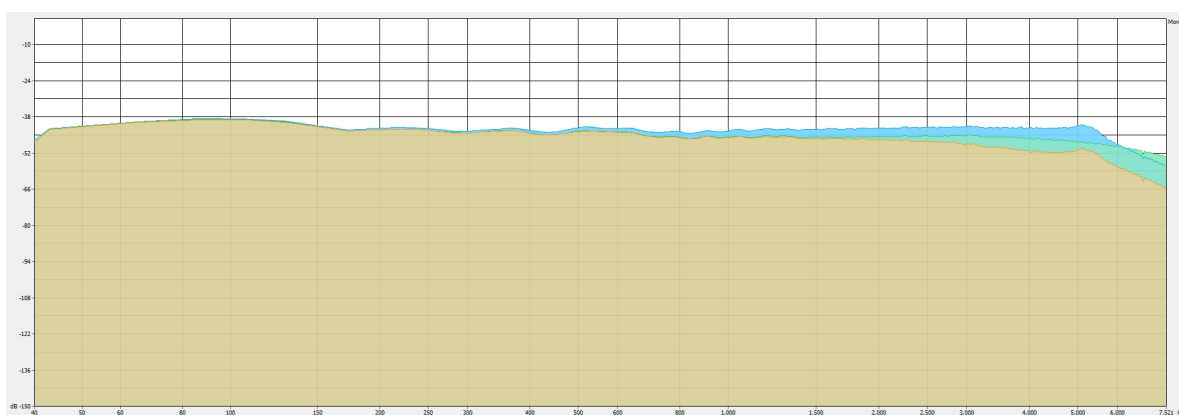


FIGURE 6.28: Record 287701 - frequencies' spectrum - Blue = no blobs clean / Green = old blobs clean / Yellow = new blobs clean

We can see the new algorithm is quieter than the previous blob clean function (around 6db).

Record 287881

Alexander Graham Bell is counting on this record. He was testing the recording of numbers. It is the only record that is audible and understandable. The figure 6.29 shows the waveform of the three different audio files.

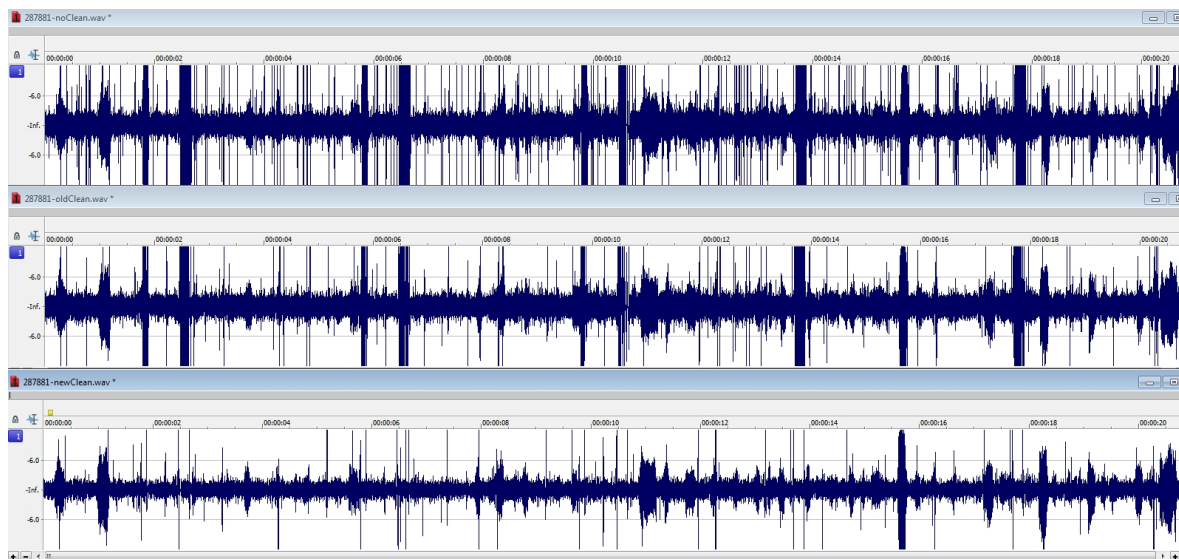


FIGURE 6.29: Record 287881 - waveforms

Like for the previous records, the new algorithm reduces the number of clicks present. The figure 6.30 shows the frequency spectrum of these audio files. The new algorithm is 7 db quieter than the previous one on this record.

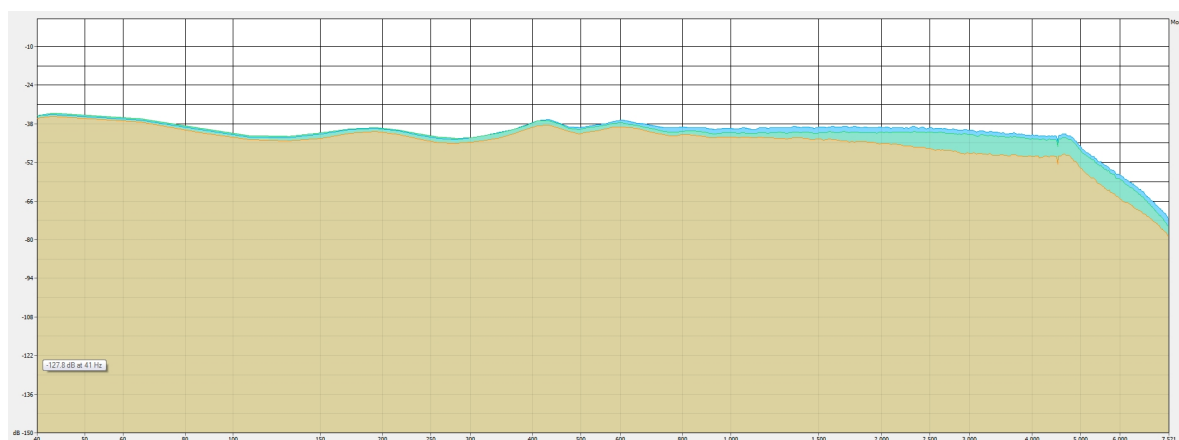


FIGURE 6.30: Record 287881 - frequency spectrum - Blue = no blobs clean / Green = old blobs clean / Yellow = new blobs clean

Remaining clicks

After all the test we proceeded, we found that some clicks appear due to the tracking. Every time the groove gets from the bottom to the top of the picture, it creates a click (figure 6.31). The others clicks are due to some undetected blobs. A possible improvement to the PRISM software could be to develop a new tracking tool adapted to this kind of records.

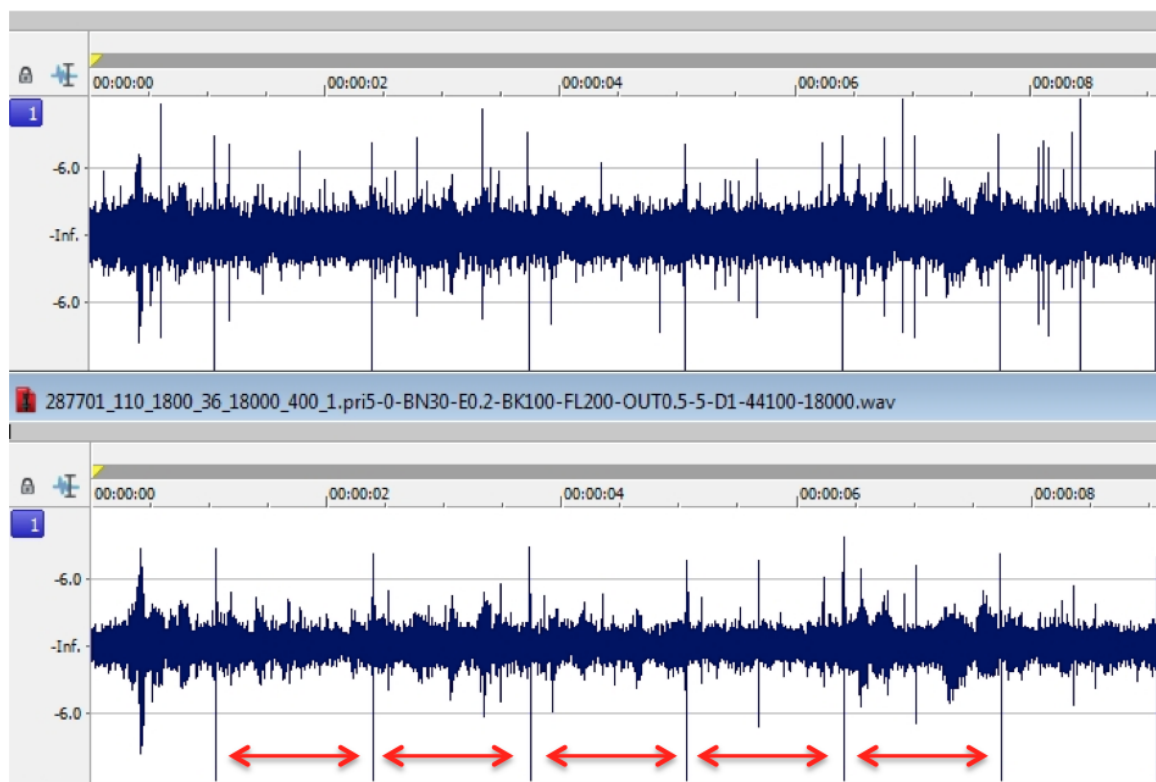


FIGURE 6.31: Record 287881 - zoom, remaining clicks

Noise

Those records are the noisiest that Carl Haber's team dealt with. The recording devices and records' deterioration are responsible of this. However we didn't apply any post processing algorithm such as noise reduction, clicks removal, etc... This part is done by professional audio reconstitution engineers and all the archives want a clean version (no post-treatment) of the record. The only record that is audible and understandable is the 287881. On this record, Alexander Graham Bell tested the recording of numbers.

6.2.7 Conclusion

These tests demonstrate that the new blobs clean function works better on all the available recordings. The new interpolation function helps to avoid a lot of clicks in the final audio file. However, there is still a need for improvement. Some blobs are still undetected by the function.

The Laplacian of Gaussian with two different σ didn't work as we expected. We decided to let the function in the source code and in the GUI if it can help for a further development.

The function to replace the muted parts with background noise worked fine expect for the artifacts at the border of these parts. We didn't manage to correct this. The solution would be to create audio samples in the border of the muted part with a spline interpolation (same class used for the blobs cleaning) to create a smooth transition between the real audio samples and the inserted background noise samples.

Chapter 7

Further developments

At the end of this project, some parts needed more consideration in order to improve the quality of results. This chapter presents those points in order to give ideas for further developments.

7.1 Blob detection

The blob detection is a very hard subject due to the large variety of blobs that can appear on the recording. Even if the new algorithm detects more blobs and is stronger than the previous one due to the adaptive thresholds and implemented algorithms, some blobs are still undetected or not completely detected. This results in clicks and noise in the audio file. A improvement would be to continue the search of new algorithms to detect blobs in order to detect all the blobs. It might be interesting to implement a blob detection function that performs multiple passes over the record. For every pass the cleaning result is used to detect the remaining blobs.

As presented in the chapter [6](#), the computation time can be long if the user processes a lot of rings at the same time. This is due to the convolution operator that is used in almost all blob detection algorithms. An improvement would be to modify the convolution function by using concurrent programming in order to reduce the computation time.

7.2 Tracking

As explained in the chapter [6](#), some remaining clicks come from the tracking. It would be useful to develop a new tracking tool adapted to this kind of record. This would lead to a better audio quality and give the possibility to analyze the new results to find other improvements.

7.3 Replacement of muted part with "silence"

During the tests' procedures we discovered that this function can create artifacts resulting in audible clicks in the border of the muted parts. This is due to the jump between the last sample value of the sound and the first sample value of the background noise. This can be corrected by using a spline interpolation between these values to create a smooth transition. Unfortunately we didn't manage to correct this before the end of the project.

Another improvement would be to try to interpolate the biggest blobs by taking into account the sound before and after the blobs instead of replacing it with the background noise. It would be possible by interpolating samples to reproduce the frequency spectrum before the blobs and by creating a smooth transition to the frequency spectrum of the audio after the blobs.

7.4 Cracks

As explained in the chapter [6](#), the cracks create clicks and make the tracking difficult. A function that try to correct the shift of the image before and after the crack could reduce the number of clicks.

Chapter 8

Conclusion

The goal of this project was to implement new tools and features to PRISM in order to deal with the records of the Volta Laboratory. The main objective was to develop a new blobs detection and cleaning function with stronger algorithms taking no parameter as input. After a lot of tests and analysis it appears that it wasn't possible to design such an algorithm. We implemented different blobs detection algorithms that are stronger than the previous one due to the adaptive thresholds. The new interpolation function (cubic spline interpolation) suits perfectly to this project and the results are, as expected, better than with a linear interpolation.

The results demonstrates that the audio quality is improved with the tools we developed and that the number of clicks present in the audio file is reduce. However as explained in chapter 7, there are some points that need more analysis in order to obtain the best quality from the recordings of Alexander Graham Bell.

During this project we had the opportunities to apply many different subjects learned during our Bachelor formation in the College of Engineering and Architecture of Fribourg. The materials form the lessons of signal processing, image processing, programming, statistic and applied physics were very useful to complete this project and my previous work on Visual Audio helped me to quickly understand the problematic.

I learned a lot about data analysis, image processing, audio processing, signal processing and project management during this project.

Finally this experience in the Lawrence Berkeley National Laboratory was incredible and working on this historical subject was very rewarding. Beside the work, I had the opportunity to discover a new country and culture. I met a lot of people and had the chance to improve my English skill. I highly recommend this experience to everyone.

Appendix A

Parameters of the new BlobClean function

The goal of this appendix is to resume all the BlobClean v2 parameters. For more information about these, please refer to the chapter 4.

The figure A.1 shows the different parameters that are available with the new BlobClean function.

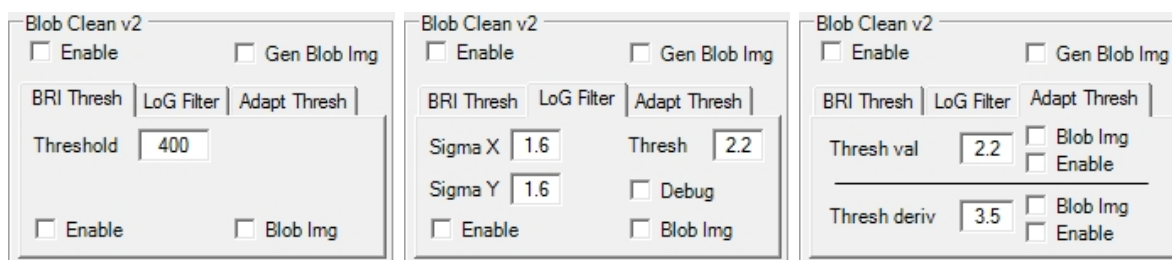


FIGURE A.1: BlobClean v2 GUI

The top right check box (Enable) enables the BlobClean v2 function. However each algorithm has to be enable in the correct tab.

The top left check box (Gen Blob Img) give the possibility to display in the detailed view the general blobs images. In each tab, a similar check box (Blob Img) is available and displays the blob image of the corresponding algorithm.

After the test of the LoG with two different sigma it appears that this function should be used with $\sigma_x = \sigma_y$ to obtain the best results

Algorithm	Parameter	Description
BRI Thresh	Threshold	Threshold value applied to the BRI file to detect lost of focus and cracks.
LoG Filter	SigmaX	Sigma for the x axis. Used to create the LoG kernel
LoG Filter	SigmaY	Sigma for the y axis. Used to create the LoG kernel
LoG Filter	Thresh	Sensibility of the algorithm, big value = less sensitive, small value = more sensitive
AdaptThresh	Thresh val	Sensibility of the algorithm, big value = less sensitive, small value = more sensitive, represents the number of standard deviation allowed
AdaptThresh	Thresh deriv	Sensibility of the algorithm, big value = less sensitive, small value = more sensitive, represents the number of standard deviation allowed

TABLE A.1: Blob Clean v2 function's parameters

Some parameters are hidden in the menu *option/BlobClean2 - more parameters* (figure A.2).

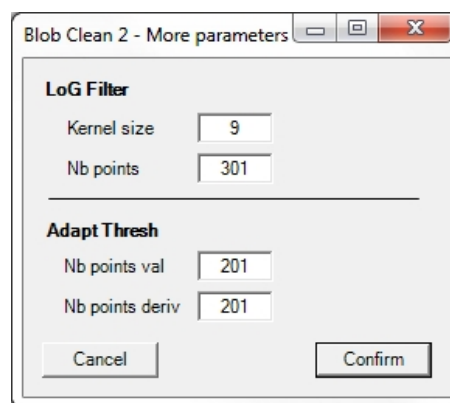


FIGURE A.2: BlobClean v2 - More parameters

Algorithm	Parameter	Description
LoG Filter	Kernel size	Size of the LoG kernel. Shouldn't be used because the kernel size is computed automatically in the code
LoG Filter	Nb points	Number of points taken when computing the adaptive threshold (use a big value)
AdaptThresh	Nb points val	Number of points taken when computing the adaptive threshold on the value (use a big value)
AdaptThresh	Nb points deriv	Number of points taken when computing the adaptive threshold on the derivative (use a big value)

TABLE A.2: Blob Clean v2 function's parameters 2

Bibliography

- [1] Video Interchange. Vintage audio history, July 2013. URL http://www.videointerchange.com/audio_history.htm.
- [2] Wikipedia The Free Encyclopedia. Phonograph, June 2013. URL <http://en.wikipedia.org/wiki/Phonograph>.
- [3] Leslie J. Newville. *Development of the Phonograph at Alexander Graham Bell's Volta Laboratory*. Project Gutenberg eBook, September 2009.
- [4] Patrick Feaster. *A Discography of Volta Laboratory Recordings at the National Museum of American History*. PDF document, March 2012.
- [5] That Eric Alper. Photo: Close-up of a record stylus on the grooves of a vinyl record, July 2013. URL <http://www.thatericalper.com/2013/01/06/photo-close-up-of-a-record-stylus-on-the-grooves-of-a-vinyl-record>.
- [6] Wikipedia The Free Encyclopedia. Noise (audio), July 2013. URL [http://en.wikipedia.org/wiki/Noise_\(audio\)](http://en.wikipedia.org/wiki/Noise_(audio)).
- [7] Wikipedia The Free Encyclopedia. Blob detection, July 2013. URL http://en.wikipedia.org/wiki/Blob_detection.
- [8] Tobias Muller. New probe for depth estimation of records: Probe. Bachelor thesis, EIA-FR, LBNL, 2010.
- [9] Adrien Nicolet. New probe for depth estimation of records: Software. Bachelor thesis, EIA-FR, LBNL, 2010.
- [10] Apple. Performing convolution operations, July 2013. URL <http://developer.apple.com/library/ios/#documentation/Performance/Conceptual/vImage/ConvolutionOperations/ConvolutionOperations.html>.
- [11] Wikipedia The Free Encyclopedia. Kernel (image processing), June 2013. URL [http://en.wikipedia.org/wiki/Kernel_\(image_processing\)](http://en.wikipedia.org/wiki/Kernel_(image_processing)).

- [12] Wikipedia The Free Encyclopedia. Standard deviation, July 2013. URL http://en.wikipedia.org/wiki/Standard_deviation.
- [13] Marquette University (Author unknown). Log filter, July 2013. URL <http://academic.mu.edu/phys/matthysd/web226/Lab02.htm>.
- [14] Wikipedia The Free Encyclopedia. Gaussian filter, July 2013. URL http://en.wikipedia.org/wiki/Gaussian_filter.
- [15] MIPAV wiki. Edge detection: Zero x laplacian, July 2013. URL http://mipav.cit.nih.gov/pubwiki/index.php/Edge_Detection:_Zero_X_Laplacian.
- [16] Alain Boucher. Vision par ordinateur - laplacien de gaussienne, July 2013. Lesson's slides.
- [17] University of Edinburgh. Adaptive thresholding, July 2013. URL <http://homepages.inf.ed.ac.uk/rbf/HIPR2/adpthrsh.htm>.
- [18] Wikipedia The Free Encyclopedia. Interpolation, June 2013. URL <http://en.wikipedia.org/wiki/Interpolation>.
- [19] Wikipedia The Free Encyclopedia. Runge's phenomenon, June 2013. URL http://en.wikipedia.org/wiki/Runge%27s_phenomenon.
- [20] Wikipedia The Free Encyclopedia. Piecewise, July 2013. URL <http://en.wikipedia.org/wiki/Piecewise>.
- [21] Wikipedia The Free Encyclopedia. Spline interpolation, July 2013. URL http://en.wikipedia.org/wiki/Spline_interpolation.
- [22] Topher Lee. Creating a wav (riff) file, July 2013. URL <http://www.topherlee.com/software/pcm-tut-wavformat.html>.