POLITECNICO
DI TORINO

POLITECNICO DI TORINO

Master degree course in Computer Engineering

Master Degree Thesis

# Image Processing and Machine Learning for Engine Fault Detection


**Supervisors**
prof. Elena Baralis
dott. Andrea Pasini

**Candidate**
Alessandro Nicoletta


October 2018

**Abstract**

The aim of this thesis is to design a process for the automatic detection of engine faults by analysing sound spectrograms. The designed technique mimics the manual state-of-the-art analysis performed by domain experts. Each operation of our method is performed automatically, without requiring domain experts intervention. The manual process being replicated consists of analysing the spectrogram image obtained from the sound emitted by the engine under inspection. This visualized spectrogram allows highlighting engine faults, which occur with peculiar characteristics that are well known by the experts. In particular, we focused on the detection of a whistle called constant tone, which appears as a straight noisy line in the spectrogram. The inspection typically requires a visual analysis performed manually by a domain expert.

The approach proposed in this thesis addresses automatically this operation by means of image processing and machine learning techniques.

The results are evaluated comparing to the performances obtained with the manual process, which is considered our benchmark. Our model is able to generate high quality detections, which are also interpretable, as they are provided with the spectrogram region where the problem occurs. We hope that this work will lead a spectrogram analyst to speed up his work by automatizing steps that would be otherwise performed manually.

# Contents

# Part I

# Background

# Chapter 1

# Introduction

## 1.1 Objectives

The main objective in this thesis is to identify a specific noise signature in a spectrogram, computed by a file audio. The audio track is the sound of an engine car, recorded by a tool during an acceleration ramp. It is the input to our model. The specific noise signature we want to find is a "whistle" in a known frequency range, called *constant tone*. This process already exists but every single step, from the spectrogram analysis to the noise signature identification, is executed by a tool that requires the intervention of domain experts.

Having big dataset and machine learning algorithms, will enable a technology transfer for automatizing manual techniques. Predictive maintenance and automatic diagnostics systems will ease analysis made by domain experts (e.g. mechanical, engine manufacturers etc.).

Our work, in this thesis, aims to automatize the engine faults detection. In particular, we focused on constant tone detection, having as output a percentage which indicates its presence.

## 1.2 Existing process

The process to detect engine faults we automatized, designed by domain experts, can be summarized in four steps:

1. Spectrogram computation
2. Main order identification
3. Time to RPM conversion
4. Faults detection (e.g. constant tone)

Each step listed above has as input the output of the previous one. The audio track is the input of the first step.

### 1.2.1 Spectrogram computation

Once the audio track is recorded, it is processed by industrial tool in order to lay out a spectrogram chart. The purpose of generating a spectrogram is that it makes easier identifying and evaluating the gravity of a problem by visual inspection. Indeed, acoustic perception is more sensitive and subjective. In addition, spectrograms can reveal information which are more difficult to hear.

A spectrogram is computed by consecutive *Fourier transforms* using a sliding window that selects for each Fourier transform a subset of samples in the original signal.

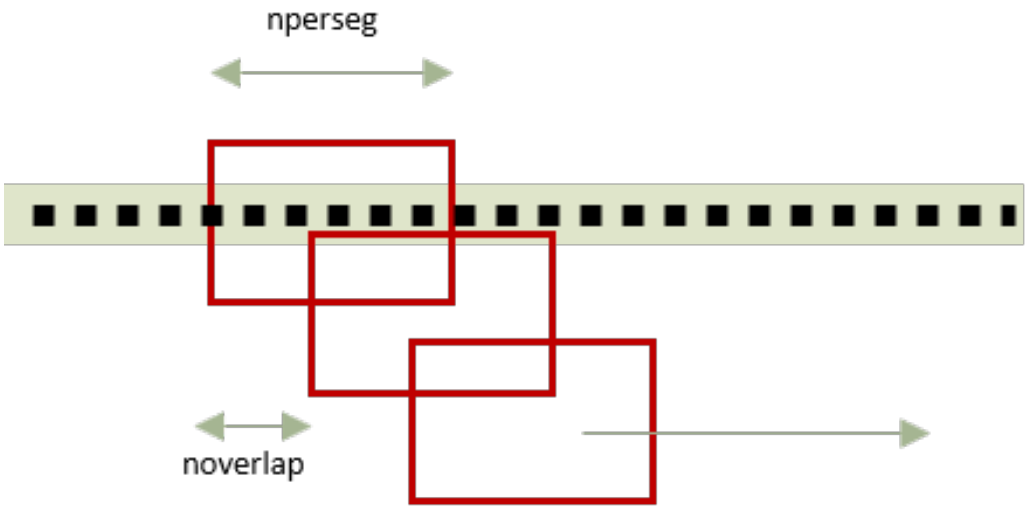



Figure 1.1: Fourier transformation process

Figure 1.1 shows the Fourier transforms process applied to an audio samples (drawn through black points). Each red rectangle represents the filter window, *nperseg* the number of samples per window and *noverlap* the number of samples overlapped between adjacent windows.

Industrial tools allow Fourier transformation parameters to be set before obtaining the spectrogram itself.

The result of the spectrogram is a two-dimensional graph having frequencies [Hz] on x-axis and times [s] on y-axis. Moreover, the third dimension represented by colour intensity, is the amplitude ($S$) of the signal.

Once the human ear can be modelled as a logarithmic listening device [1], amplitude is converted to decibel [dB], computing:

$$S_{decibel} = \log_{10} \frac{S}{20 \cdot 10^{-5}}$$

Finally, range of frequencies represented on the graph is reduced to the default range $[f_{min}, f_{max}]$ Hz where engine features and eventually problems are visible.

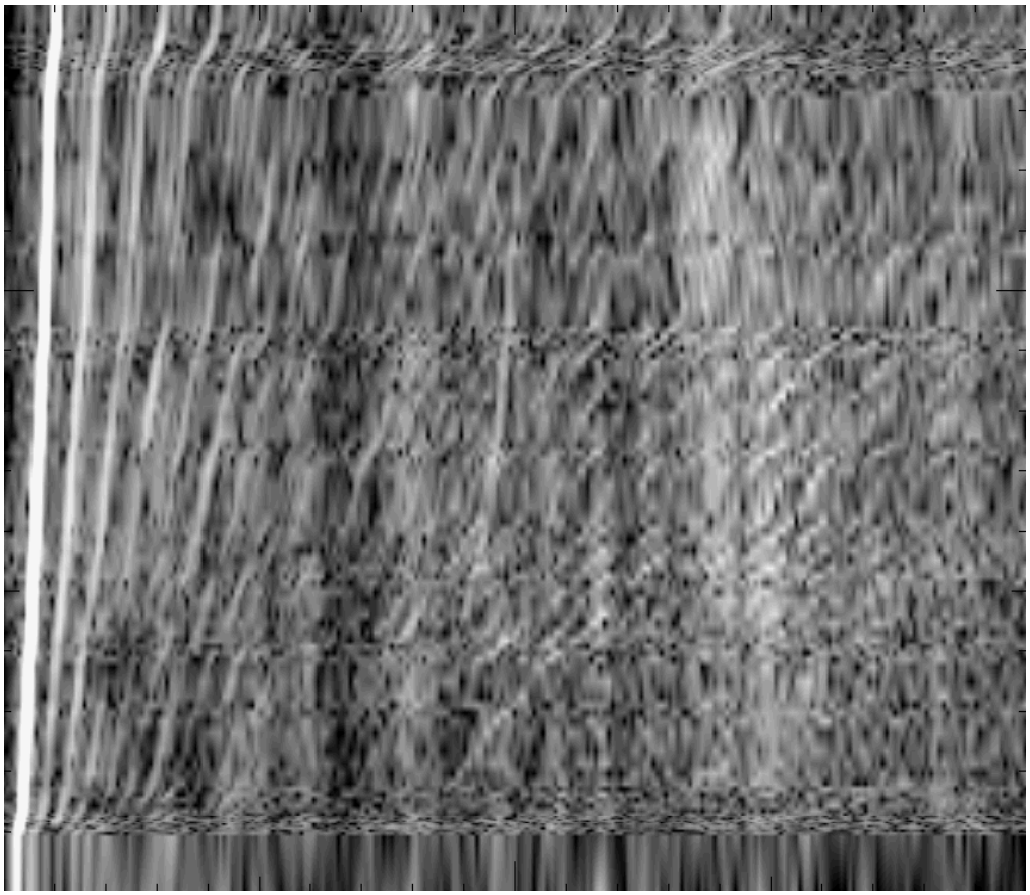


Figure 1.2: Example of spectrogram produced by typical industrial tool

Figure 1.2 shows an example of spectrogram produced by an industrial tool, zoomed on the area between $[f_{min}, f_{max}]$ Hz, represented on x-axis. The bright lines in Figure 1.2, represent the engine *orders*. An engine order is a vibration and/or acoustic response due to rotating components in the engine (e.g. crankshaft, toothed wheels, gears etc.). Each order is generated by the rotation of a particular component. Once the acceleration of these components changes, their response change too. The brightest line in Figure 1.2, is the *main order*. It is brighter than others because it is produced by the combustion of the crankshaft that will produce a more intense response. Thanks to the formulas created by domain experts, the combustion order can be just calculated by dividing number of cylinders by 2.

### 1.2.2 Main order identification

The next step is to shift domain from time to RPM. However, to obtain this conversion, the main order (what orders are described at Section 1.2.1) must be identified and its trend must be traced through points. They have to be manually drawn inside the main order itself.

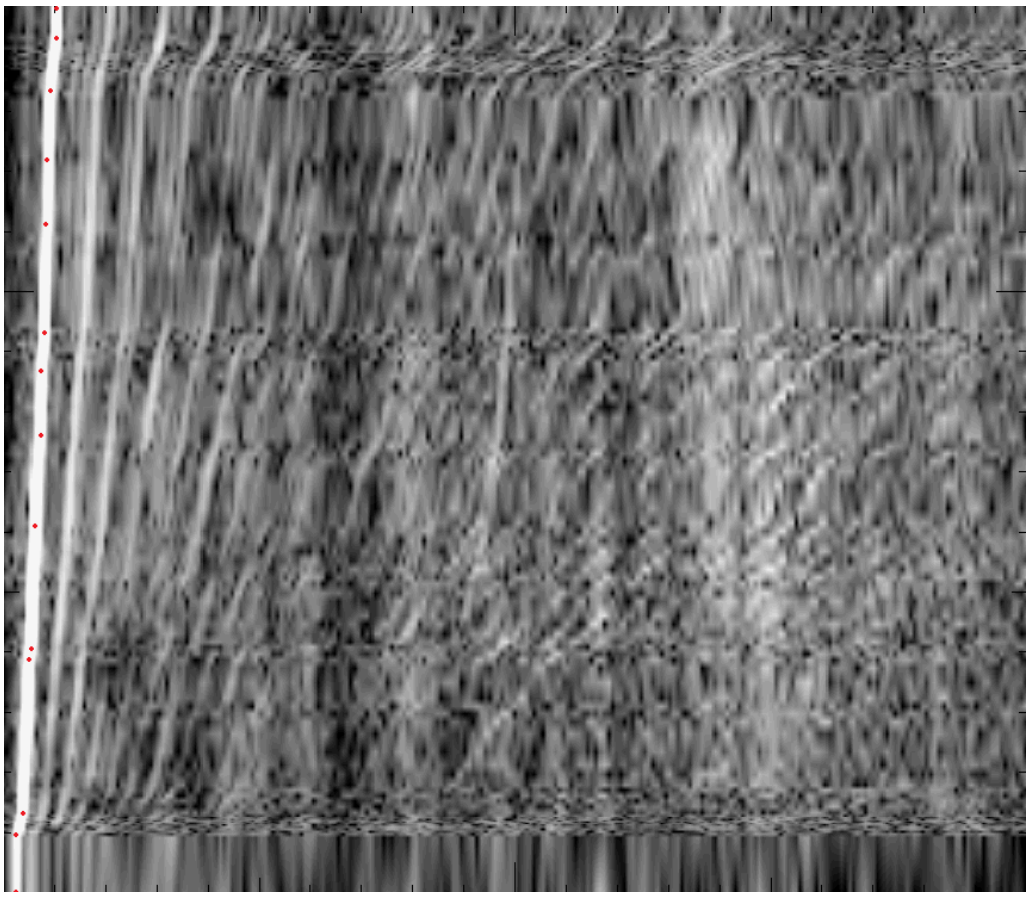


Figure 1.3: Example of manually detected main order

Figure 1.3 shows how to trace points inside the main order, following its shape. For what concerns the identification task, it is performed just knowing the number of cylinders and, as a consequence, the main order number, computing the formula explained in Section 1.2.1. Moreover, being the most intense one in magnitude,

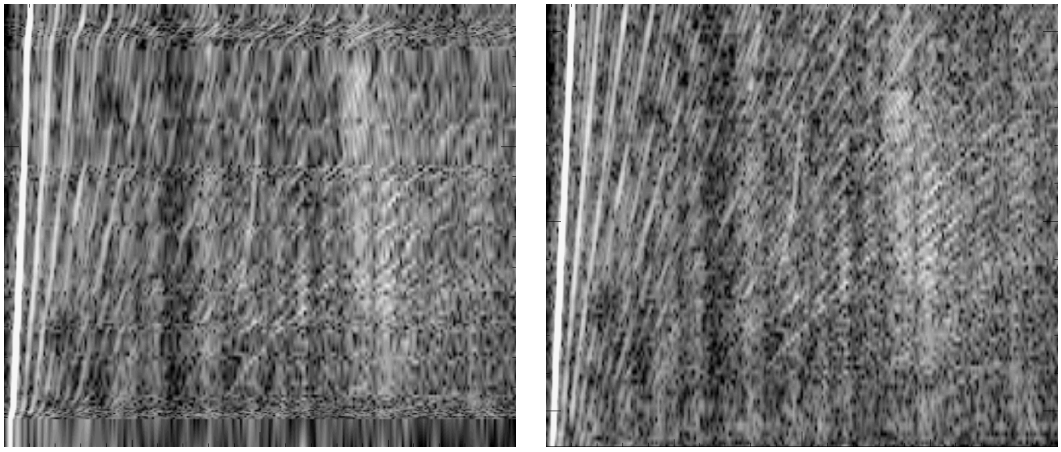
main order is clearly distinguishable than other orders.

### 1.2.3 Time to RPM conversion

Finally, industrial tool uses points tracked on main order, as described in Section 1.2.2, for performing the conversion from time to RPM domain. RPM stands for revolution per minute and it represents the number of cycles computed by rotating components of the engine per minute. The target of this conversion is to straighten the main order and, as a consequence, all the orders. Indeed, an order linearly increases in RPM for increasing frequency values, as demonstrated by the given formula

$$RPM = f_n \cdot \frac{60}{n}$$

used for computing RPM value, having the frequency $f_n$, where $n$ is the main order number.



(a) *Time domain* (b) *RPM domain*

Figure 1.4: Examples of time to RPM domain conversion

Figure 4.31 depicts a time to RPM conversion and how main order and orders in general are straightened out in RPM domain than the time domain. The conversion is necessary because the acceleration performed by a human in order to collect the audio track, cannot be perfectly linear in the time domain, as Figure 4.31

demonstrates. In addition, having linear orders, let domain experts to apply known formulas, applicable only on RPM domain, to evaluate severity of engine faults, if any.

### 1.2.4 Engine faults detection

The last step is to understand if there are any engine faults and in case analyse them. In particular, constant tone is an unusual noise that is driven by the charging system. This noise is almost constant in frequency but the frequency is not always the same because it is in function of the temperature of the oil at which the measurement is performed. The phenomena appears in known frequency ranges. Then, operator, knowing constant tone characteristics, is able to detects it and analyse it by marking its shape with the tool.

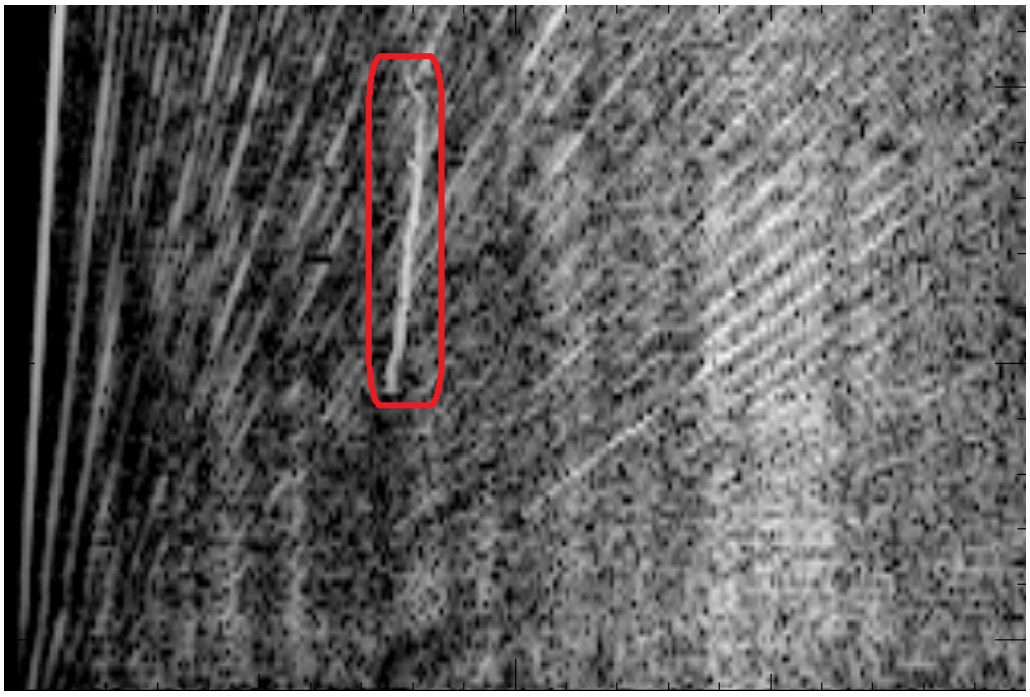


Figure 1.5: Example of constant tone

Figure 1.5 depicts how the constant tone, inside the red rectangle, appears in a

spectrogram graph.

# Part II

# Model implementation

# Chapter 2

# Data Collection

In the actual state-of-the-art we cannot find a database that satisfies objectives described in Section 1.1, so data had to be collected from scratch. After collecting the audio tracks, each of them must be manually annotated with the engine informations needed to train and test the process designed for this thesis.

## 2.1 Chosen tool for recording audio track

The choice of the tool used to collect data has been directed to smartphone microphones for two main reason. First, they fully satisfy the characteristics needed to catch all the informations to be analysed, after converting it into related spectrogram. Secondly, they are available almost to everyone by now and, in addition, the spread of smartphones all over the world is increasing every year.

## 2.2 Data collection process

The process made to collect input data requires two operators. Having a car with known engine characteristics, such as the model and especially number of cylinders, one operator, at the wheel, puts neutral gear and, from idle, speeds up to a default

$RPM_{max}$ value for a determinate interval of seconds. Meanwhile, the other operator is recording, with a smartphone, the sound produced by the engine during the whole acceleration, with the hood of the car opened so that the sound results clearer. When the operator at the wheel terminates the acceleration, the other one has to stop the recording.

In our research, in order to collect enough data for the model, we applied the process just described to 10 different cars and for each one we have repeated the process 5 times for a total of 50 files. Each repetition has been executed with different durations for studying best duration time and seeing if other aspects, such as engine heat, could influence measurements. Smartphones used for our experiments are one Android and one iOS device, the two best-selling brands in the word, choosing best recording quality for both devices.

## 2.3 Audio tracks characteristics

The audio collected characteristics can be summarized in the Table 2.1:

| | **Android** | **iOS** |
|---|---|---|
| **Codifier** | WAV | WAV |
| **Channel** | Mono | Mono |
| **Sampling [Hz]** | 44100 | 48000 |
| **Max time [s]** | 37 | 37 |
| **Min Time [s]** | 8 | 8 |
| **Avg Time [s]** | 24 | 24 |

Table 2.1: Audio collected characteristics

These chosen values guarantee a good quality and let engine faults be detected. We recorded some audio for less and more time than indicated range in standard procedure, in order to understand how the results change

## 2.4 Spectrogram generation

The next step of analysis replicates the existing process described at Section 1.2.1.

First, to avoid microphone dependencies, the audio samples are normalized

$$samples = [x_0, x_1, ..., x_{N-1}]$$

$$N = total\ number\ of\ samples$$

using the *z-score* function

$$z_i = \frac{x_i - \overline{x}}{S}$$

applied to each sample

$$x_i,\ i = 0\ ...\ N-1$$

having the mean function

$$\overline{x} = \frac{1}{N}\sum_{i=0}^{N-1} x_i$$

and the standard deviation

$$S = \frac{\sum_{i=0}^{N-1}(x_i - \overline{x})}{N-1}$$

.

Except this last operation, the existing process, described in Section 1.2, is replicated as it is. In summary:

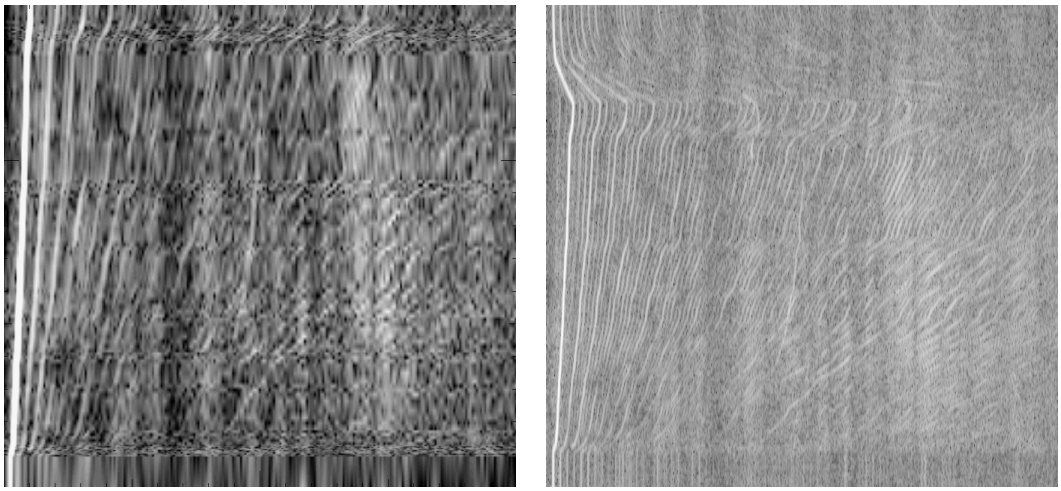
- consecutive Fourier transforms applied to recorded audio (choosing the same parameters used for the existing process)
- the conversion of resulting spectrogram to decibel (dB), to enhance analysed features
- frequencies window reduction to $[f_{min}, f_{max}]$ Hz, to zoom on the area where wanted features appear

The Fourier transforms are performed by applying the Discrete Fourier Transform (DFT)

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{kn}, \qquad k = 0, 1, ..., N-1$$

to each subset of samples $x[n]$ of length $N$, where $W_N = e^{-j(2\pi/N)}$. For optimize the DFT computation we applied the Fast Fourier Transform algorithm (FFT) [2].

The spectrogram computation generates a *matrix* and two *vectors*. The matrix, representing the spectrogram image, contains a gray intensity value (i.e. from 0 to 255) for each pixel in position $(x, y)$. Moreover, the first vector is the *frequencies vector* contains for each x-coordinate of the image, the corresponding frequency value, expressed in hertz (Hz). The other one is the *time vector*, contains for each y-coordinate of the image the corresponding time value, expressed in seconds (s).



(a) *Spectrogram by existing tool unfiltered* (b) *Spectrogram by our process unfiltered*

Figure 2.1: Spectrograms comparison

Figure 2.1 depicts the comparison between the spectrogram obtained by our process than by an industrial tool. The two spectrograms show the orders and engine faults in the same way.

# Chapter 3

# Data Labelling

## 3.1 LabelMe annotation tool

For evaluating the accuracy of our method in comparison with the existing one and keeping track of all the known features of spectrogram described in Section 2.4, we needed annotated images providing a ground truth. These have to be generated with an annotation tool which allows user to add additional informations to images. *LabelMe* [3] covers the purposes, is in fact a simple tool to build image databases, based on annotations. It is possible to draw polygons on each image, characterizing them with a name, attributes and different colours.

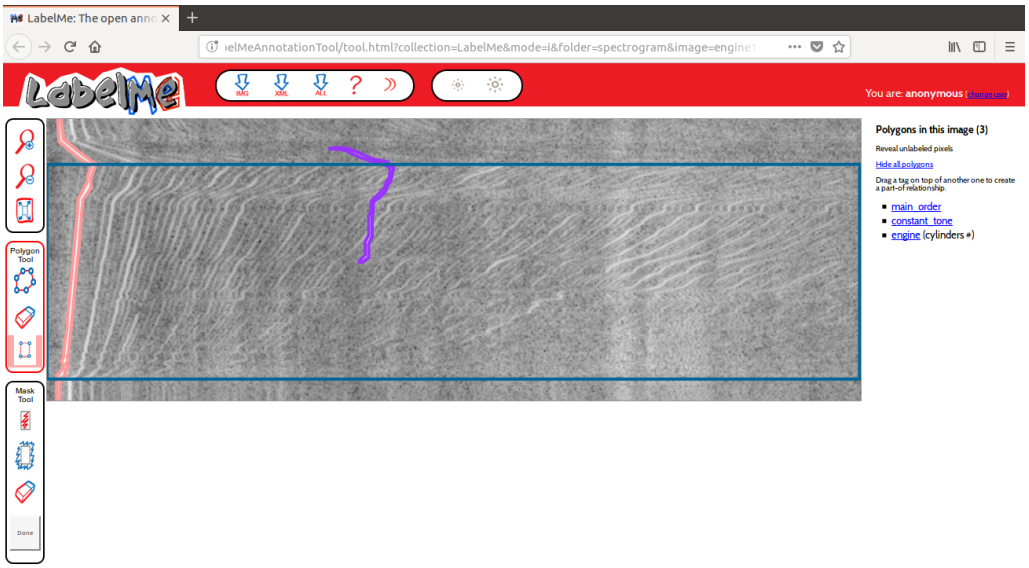



Figure 3.1: LabelMe tool

Figure 3.1 shows a screenshot of the tool and the main three sections used for labelling each image.

The first section is the toolbar on the left, especially useful to draw polygons on the image.


Figure 3.2: LabelMe tool - drawing toolbar

Starting from the top down of Figure 3.2, the first two buttons are used for zooming respectively in out the image, to obtain more accuracy in the drawing phase, while the third one is used for restoring the original size of the image. In order to draw polygons, there are two options. The first one is the fourth button that allows drawing polygons (pink and purple one in Figure 3.1) point by point, each one connected with the previous one with a segment until the first point is selected and polygon is closed. The second option is the fifth button used to draw rectangular polygons (blue one in Figure 3.1) selecting just the top left and bottom right points that define a rectangle in the image. The last button is useful to undo drawing operations, step by step (e.g. deleting one by one the points defined for a polygon, starting from the last to the first one).

The second main section in Figure 3.1 is the central part where there is the image that needs to be labelled. After selecting one of the two options described before for drawing polygons, the user can interact with the central image to build them.

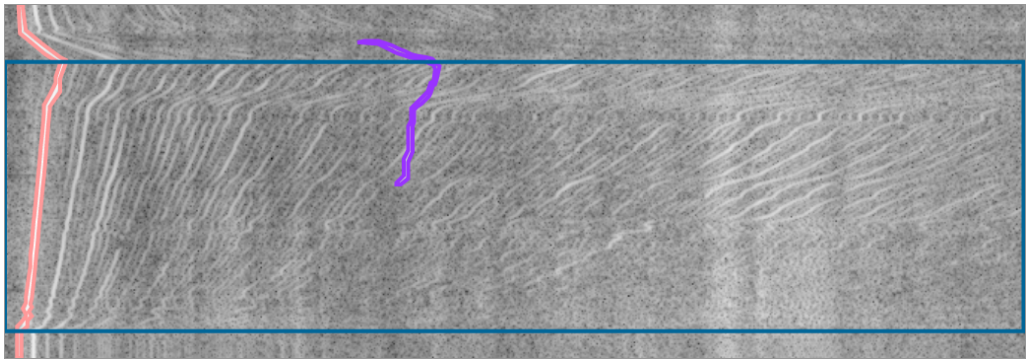


Figure 3.3: LabelMe tool - labelled image

Figure 3.3 shows an example of image loaded and labelled by the tool. When a polygon is drawn, a window appears in the image for inserting *object name* and eventually *attributes* of polygon itself.

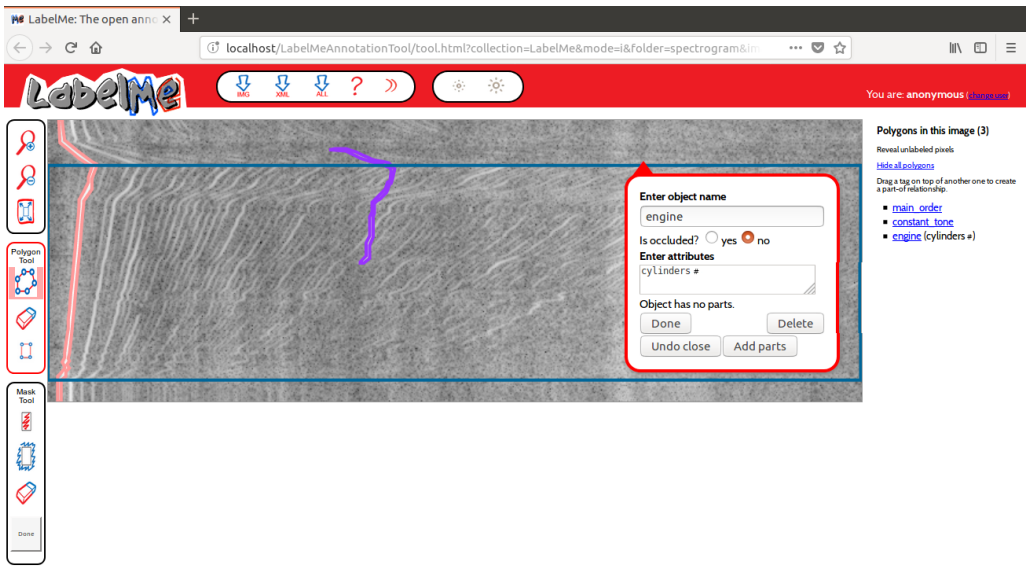



Figure 3.4: LabelMe tool - object name, attributes

Figure 3.4 shows the described window of the polygon having *engine* as object name and *cylinders #* as attribute. After inserting at least the object name and clicking on "done", the polygon is saved in a XML file and we can see its name appears in the last main section on the right in Figure 3.1.

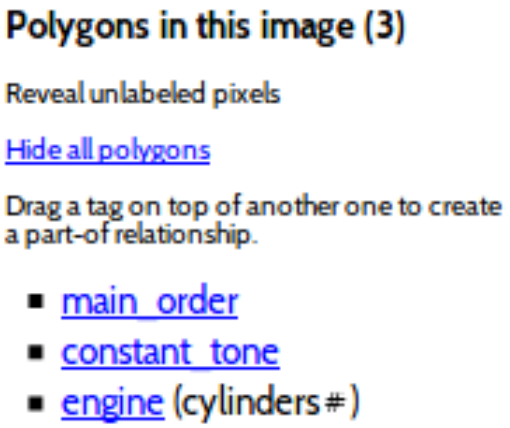



Figure 3.5: LabelMe tool - polygons summary

Here in fact, as Figure 3.5 depicts, there are all the polygons drawn down with their names and eventually attributes. The tool allows the user to click on them and, as result, it displays the window previously shown in Figure 3.4 of the corresponding polygon, giving also the possibility to modify its name and attributes already chosen.

## 3.2 Annotated features

Each spectrogram is provided with the following annotations:

- engine type
- main order
- constant tone

Each of them is useful for training and test our process.

### 3.2.1 Engine

This annotation describes general features of the engine analysed with the spectrogram. It is represented with a rectangle (blue one in Figure 3.3) that covers the significant part of the image, excluding the upper region where the engine RPMs start decreasing and the lower region where the engine RPMs are constant. This is done to evaluate the accuracy of the automatic detection of this main region of the image.

This annotation includes also the number of *cylinders* of the engine in order to obtain the *main order number*. This value is computed by dividing the number of cylinders by 2. The number obtained is used in the time-to-RPM process, described in Section 1.2.3. Applying the equation:

$$RPM = \frac{60}{n} \cdot f_n$$

where $n$ is the main order number and $f_n$ is the frequency of order n, it is possible to compute the RPM value of an order at a specific frequency.

### 3.2.2 Main order

The main order contour (pink polygon in figure 3.3) is annotated to evaluate the accuracy of its identification. Indeed, its correct identification is necessary for performing the time-to-RPM process.

### 3.2.3 Constant tone

This annotation is required for building a classifier that identifies constant tones when there are present and test the accuracy of identification. An example of annotation is the pink polygon in Figure 3.3, representing the constant tone.

## 3.3 XML format

Annotations created by drawing polygons are stored to file system in XML format and, therefore, they can be automatically read and elaborated by our process.

Listing 3.1: LabelMe annotation

```
<annotation>
        <filename>engine1.png</filename>
        <folder>spectrogram</folder>
        <source>...</source>
        <object>...</object>
        <object>...</object>
        <object>...</object>
        <imagesize>...</imagesize>
</annotation>
```

Listing 3.1 shows how XML is formatted. Each polygon is associated to an *object tag*, whose most significant fields are: *object name*, such as main order, constant tone and engine, that let our process automatically understand the kind of polygon read and how process it; *attributes*, such as number of cylinders, which add extra information to object name and, in the end, *polygon* which represents the annotation points, expressed as $x$ and $y$ coordinate tags.

Listing 3.2: object tag

```
1  <object>
2          <name>engine</name>
3          <deleted>0</deleted>
4          <verified>0</verified>
5          <occluded>no</occluded>
6          <attributes>cylinders #</attributes>
7          <parts>...</parts>
8          <date>15-Sep-2018 20:44:11</date>
9          <id>2</id>
10         <type>bounding_box</type>
11         <polygon>
12                 <username>anonymous</username>
13                 <pt>
14                         <x>0</x>
15                         <y>38</y>
16                 </pt>
17                 <pt>
18                         <x>682</x>
19                         <y>38</y>
20                 </pt>
21                 <pt>
22                         <x>682</x>
23                         <y>218</y>
24                 </pt>
25                 <pt>
26                         <x>0</x>
27                         <y>218</y>
28                 </pt>
29         </polygon>
30 </object>
```

Listing 3.2 shows an example of object tag created by drawing a polygon. The main fields are: line 2, which describes the polygon name ("engine" in this example), line 6 which shows the cylinders attribute and line 11 that represents the polygon tag. Inside it, line 13, 17, 21 and 25 display the points that establish the polygon. Each of them is composed by x and y coordinates (e.g. lines 14 and 15) of the image where that point can be found.

# Chapter 4

# Base Model

## 4.1 Main order identification

Starting from the spectrogram representation described in Section 2.4, our process mimics the process performed manually by domain experts. The first step of this process is the *main order identification*. This operation, as described in Section 1.2.1, is based on the fact that the main order is the *most intense* order in the spectrogram and it is in a specific *window*, depending on known engine characteristics. Indeed, these two conditions lead our procedure to correctly recognize it. Moreover, the image must be first preprocessed to ease the identification phase, highlighting, as much as possible, the main order and removing noise.

### 4.1.1 Image filtering

The *image filtering* is the process typically performed on images to reduce noise and highlighting useful informations. Indeed, image filtering is the first step of our preprocessing phase. Especially, we analysed in our experiments *bilateral filter* and *mean shift filter*.

Bilateral filter [4], is a technique implemented for smoothing images while preserving edges (differently to a simpler Gaussian Filter). This aspect is important in our case, for distinguishing the main order from the others. This algorithm changes each pixel value by a weighted average of its neighbors. It depends only on two parameters:

- $d$ that indicates the diameter within considered a pixel a neighbour
- $\sigma$ representing the intensity of the features to preserve

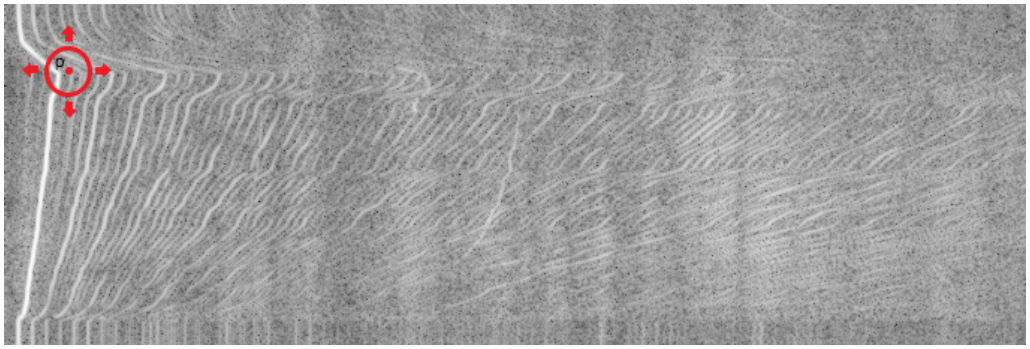
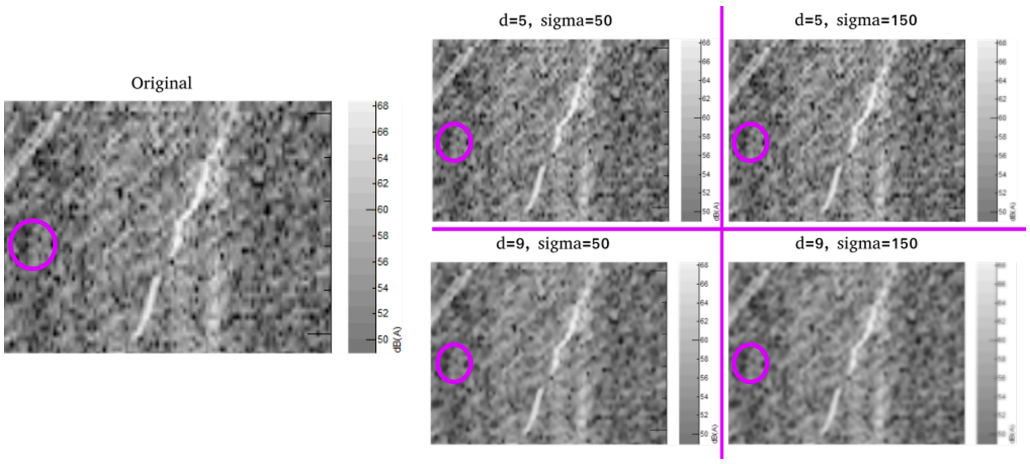


Figure 4.1: Example of bilateral filter window.

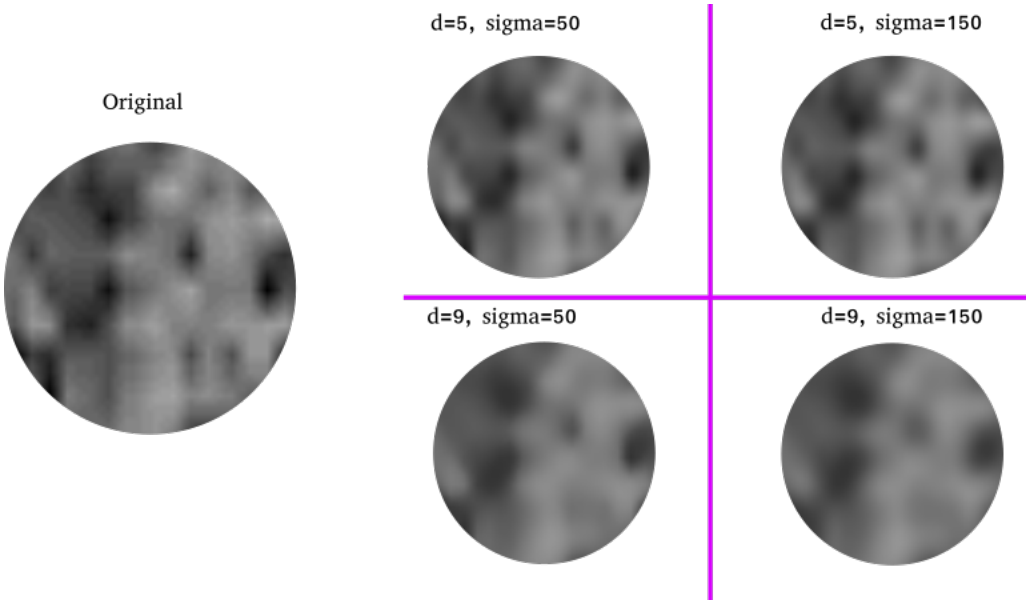
Figure 4.1 shows the fixed window of bilateral filter, represented with a red circle, having pixel $p$ as center and all the pixels inside the window with diameter $d$ as neighbours. The key idea of this algorithm is that a pixel will affect another pixel, if it is in a nearby area and it has a similar intensity value. The similarity threshold between two pixels is defined by $\sigma$ parameter. Higher values of $d$ smooth larger features while higher values of $\sigma$ make widens and flattened images (intensity approximately constant over all the image).

In our analysis, we investigated different values of $d$ and $\sigma$ to find the best combination for our images. The set of considered values are:

- $d = [50, 150]$;
- $\sigma = [5, 9]$.




(a)




(b)

Figure 4.2: Example of spectrogram filtered by bilateral filter with different value of $d$ and $\sigma$.

Figure 4.2 depicts how image changes combining different values of $d$ and $\sigma$. Zooming on images processed by bilateral filter, we better noticed that the best trade-off

for the analysed values is $d = 9$ and $\sigma = 50$ because noise is reduced and, at the same time, edges are still preserved. Combination $d = 9$ and $\sigma = 150$ preserves less the edges, as we can notice in Figure 4.2 where the borders of the numbers representing decibels are too smoothed. Instead, with $d = 5$ and any combination of $\sigma$, noise is not properly removed, as we can notice in Figure 4.2. To further confirm this, we investigated the colour histogram of images filtered with the parameters listed before.

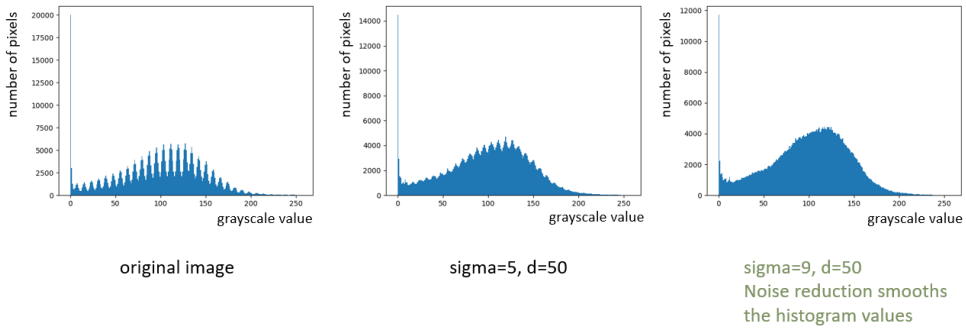
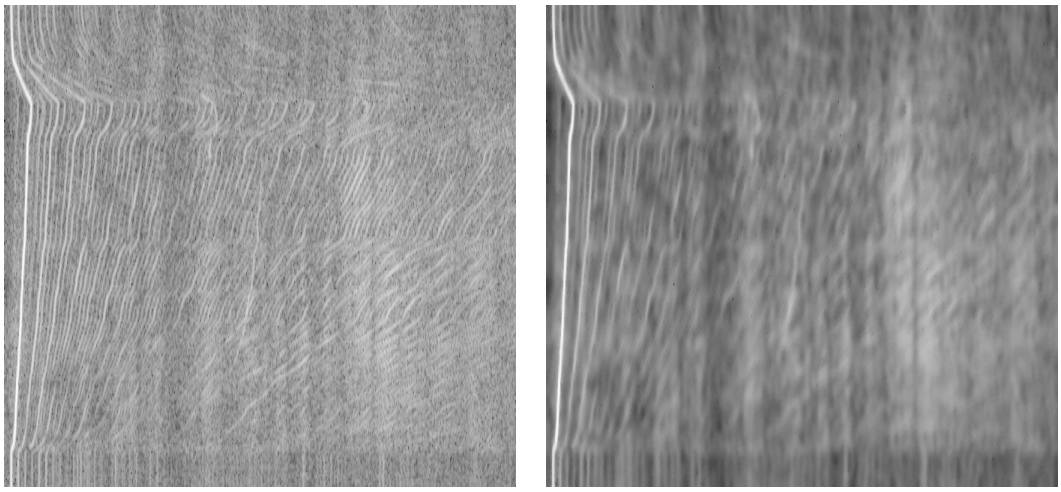



Figure 4.3: Histogram of images filtered by different $d$ and $\sigma$ values.

Histograms in Figure 4.3 show the distribution of the pixels over each gray scale value. As expected, the right most image in Figure 4.3 presents a better smoothing of the histogram peaks. Higher peaks (left most image in Figure 4.3) mean less uniformity of gray scale in the image. Hence, from this analysis, we decided to apply the bilateral filter with $d = 9$ and $\sigma = 50$ to the spectrograms generated as explained in Section 2.4.



(a) *Original spectrogram* (b) *Filtered spectrogram*

Figure 4.4: Example of spectrogram filtered by bilateral filter with $d = 9$ and $\sigma = 50$.

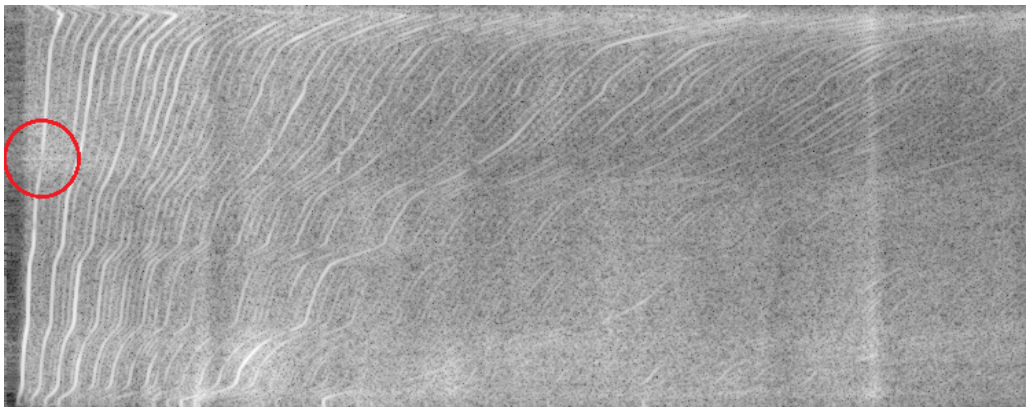
Figure 4.4 depicts how bilateral filter preserves edges and highlights the main order by removing noise.

Besides the bilateral filter, we also analysed the effects of *mean shift filter* [5]. Both methods process both the spatial and colour intensity range domains. However, while the bilateral filter exploits a static window in the two domains, the mean shift window is dynamic as it moves in the direction of the maximum increase in the density gradient. Hence, the mean shift filter is more suitable to the local structure of the data. [6]

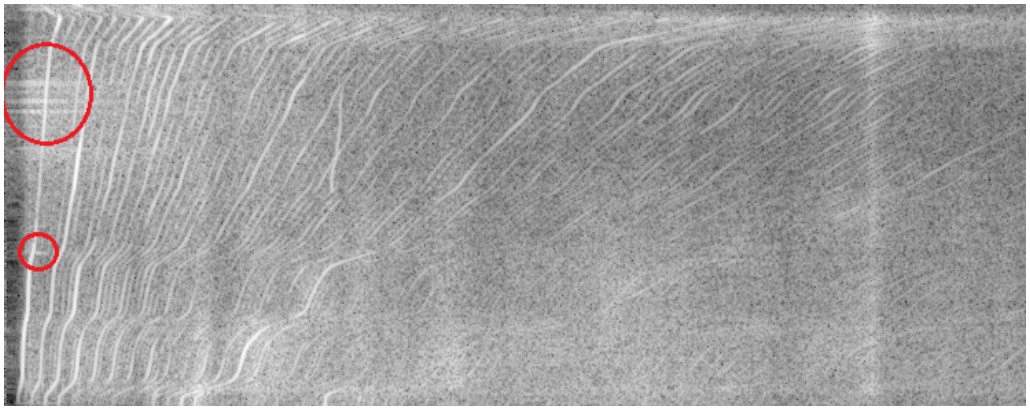
The parameters needed for mean shift filter are similar to the ones of bilateral filter and, to choose the best combination, we adopted the same method for bilateral filter. We analysed all the configurations from the Cartesian product between the sets of values chosen for $d$ and $\sigma$. The sets are:

- $d = [1, ..., 50]$;
- $\sigma = [10, ..., 50]$.

In order to choose the best combination, we inspected the most noisy images in our dataset.



(a) *Engine 2*



(b) *Engine 3*

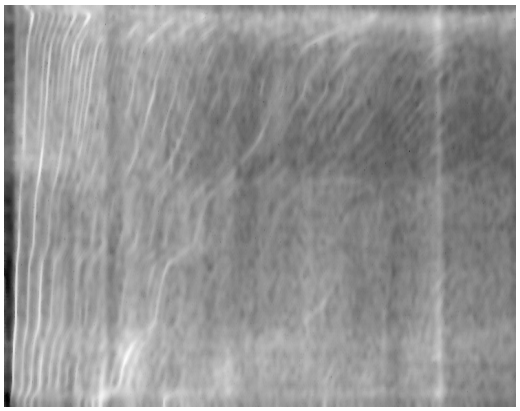
(c) *Engine 4*

Figure 4.5: Example of spectrograms affected by noise.

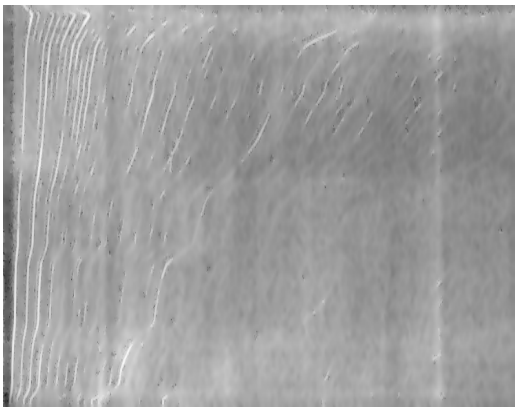
Figure 4.5 shows some example of noisy images (without applying mean shift filter).

Noisy parts, marked with red circles, can affect negatively the main order identification, especially when they are touching its shape. We observed that the combination that better reduces noise, while preserving the quality of the main order, is $d = 1$ and $\sigma = 30$.

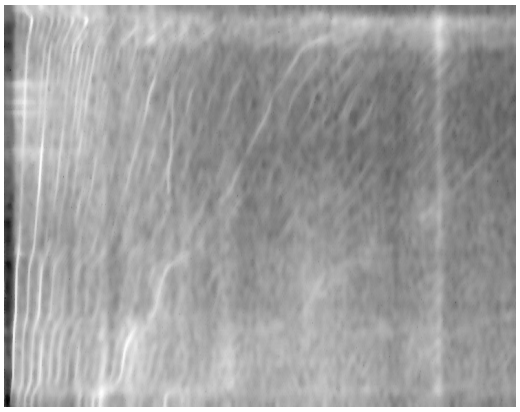
In general, our experiments confirmed a better attitude of mean shift filter than bilateral filter to noise reduction thanks to its non static window.



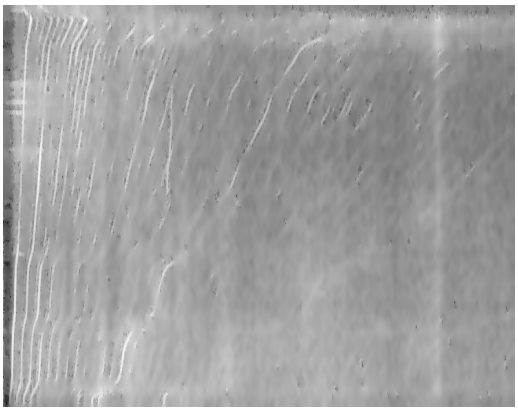
(a) *Engine 2 bilateral filter,* d = 9 *and* $\sigma$ = 50.



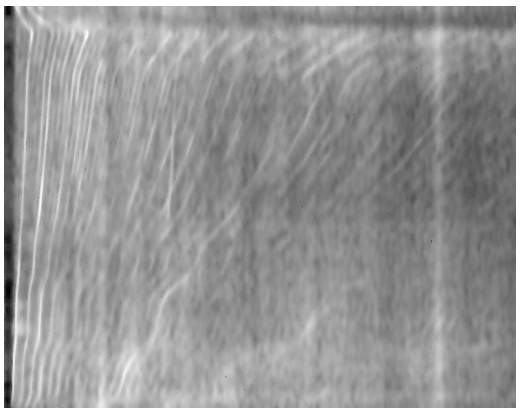
(b) *Engine 2 mean shift,* d = 1 *and* $\sigma = 30$.



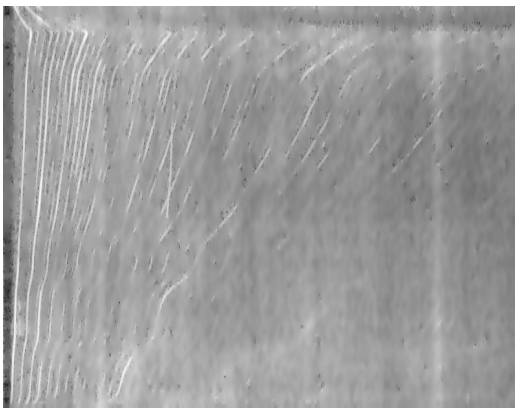
(c) *Engine 3 bilateral filter,* d = 9 *and* $\sigma$ = 50.



(d) *Engine 3 mean shift,* d = 1 *and* $\sigma = 30$.



(e) *Engine 4 bilateral filter,* d = 9 *and* $\sigma$ = 50



(f) *Engine 4 mean shift,* d = 1 *and* $\sigma = 30$.

Figure 4.6: Comparison between *bilateral filter* and *mean shift.*

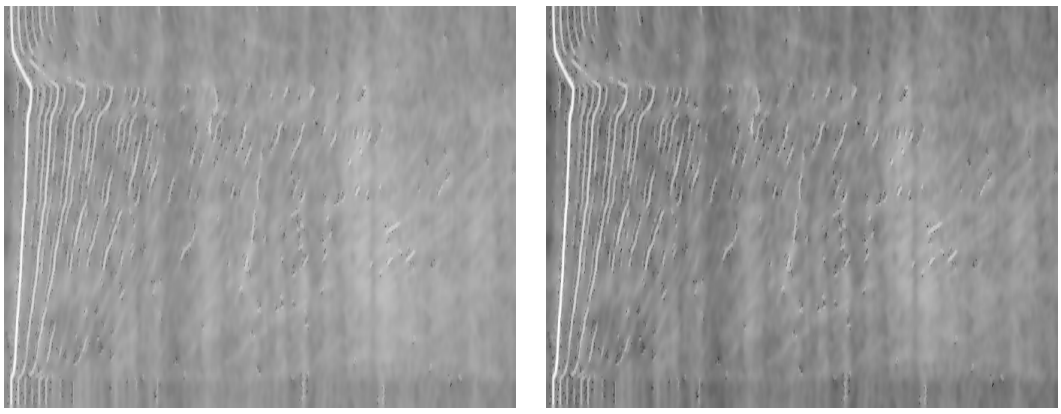
Indeed, Figure 4.6 depicts how the main order is finer approximated by mean shift. Noisy regions, indicated in Figure 4.5, are more separated from the main order by mean shift rather than bilateral filter. Hence, mean shift is confirmed as our final choice for the image filtering process.

### 4.1.2 Normalization

The second phase of main order identification pipeline, is the *normalization* of the image produced by the image filtering step, described in Section 4.1.1. The main goal of this step is to obtain a common magnitude scale for all images and then focus the analysis on relative magnitudes, instead of absolute scale values. This operation helps to make our analysis independent of recording device. We adapted the min-max normalization, as it allows our process to have a fixed range of the resulting pixel values. It is defined as:

$$image_{norm} = \frac{image_{filtered} - min(image_{filtered})}{max(image_{filtered}) - min(image_{filtered})} \cdot M$$

where $image_{norm}$ is the normalized image, with values ranging from 0 to $M$ (i.e. 255 for images with byte depth).



(a) *Spectrogram filtered by mean shift* d = 1 *and* $\sigma = 30$.

(b) *Spectrogram normalized.*

Figure 4.7: Example of normalization process

Figure 4.7 demonstrates how normalizing the gray scale improves contrast, due to a complete distribution of values along gray scale.

### 4.1.3 Discretization

Normalization phase just described in Section 4.1.2, assumes an important role for the *discretization* step. Actually, the goal of this phase is to identify regions characterized by a specified magnitude, especially the region where the main order magnitude can be found. This is performed by exploiting the characteristic of main order to be the most intense one in magnitude. Hence, normalization fits our aim of isolating better the magnitude of main order from the others.

The discretization process reduces the cardinality of the possible pixel values. This is helpful to reduce the high gray scale, that would be useless in the identification phase.

To discretize the gray scale, we inspected two possible techniques: *k equal-sized partitions* [7] or *clustering* [7]. The first approach is simpler but it is data independent, unlike the second approach that is data dependent and more effective. For this reason, our choice is to use clustering methods. Clustering algorithms aim to group related objects, according to a particular distance metric. The clustering method used in our work is *K-means* [7]. After discretization phase, the obtained gray scale ranges are based on the distribution of the pixel intensities in the analysed image. Considering that the main order to find is characterized by the highest intensity, it can be found in the last cluster, which represents the most intense magnitude. The gray scale of each pixel grouped in the last cluster is converted to the max intensity of gray scale, in our case value 255, representing white colour. The other values with lower intensities are replace with black pixels. This is done to have a binary image, condition needed for the contour detection phase described later in Section 4.1.4. The pseudocode describing K-means is shown in Algorithm 1:

**Algorithm 1** K-means

**Require:** $K$ value {number of clusters}
  *choose random K centroids*
  **repeat**
    assign all gray values to the nearest K-th centroid
    recompute centroid as the center of gravity of each cluster
  **until** centroids moves less than epsilon **or** max number of iterations are reached

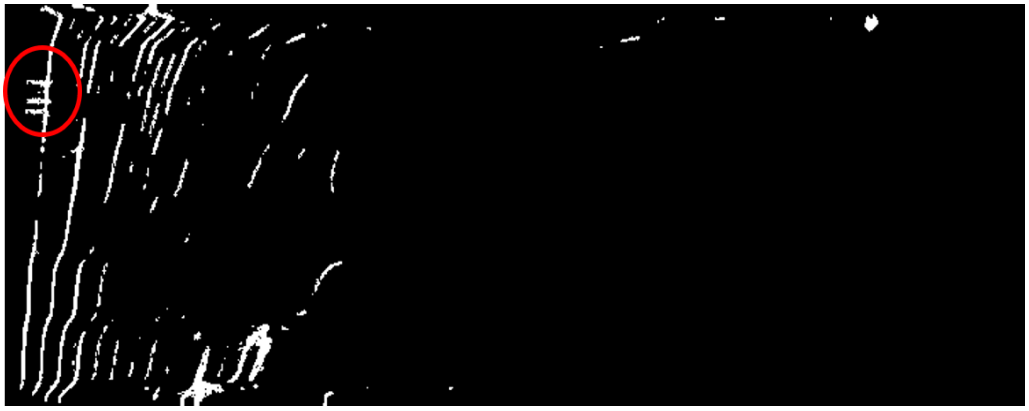
Clusters are groups of elements with similar features. In our case a cluster is a subset of the gray scale range whose an element belongs to. Moreover, the centroids are the points of reference that allow the algorithm to choose which cluster an element belongs to. In our case, the centroid of a cluster is the average colour of its pixels.

To choose the value of $K$, we based on empirical analysis. Different values of $K$ are explored, combined with previous steps, for choosing a good $K$. Values analysed are:

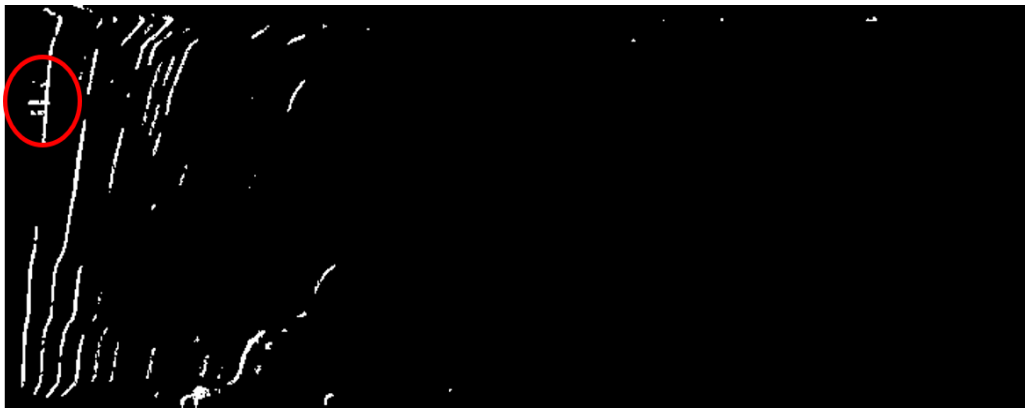
$$K = [5, 6, 7]$$

In most cases, all three values, seeing the last cluster, produce similar good solutions. For this reason we focused on images where noise is stronger in order to find the value of $K$ less affected by noise.

(a) $K = 5$



(b) $K = 6$



(c) $K = 7$

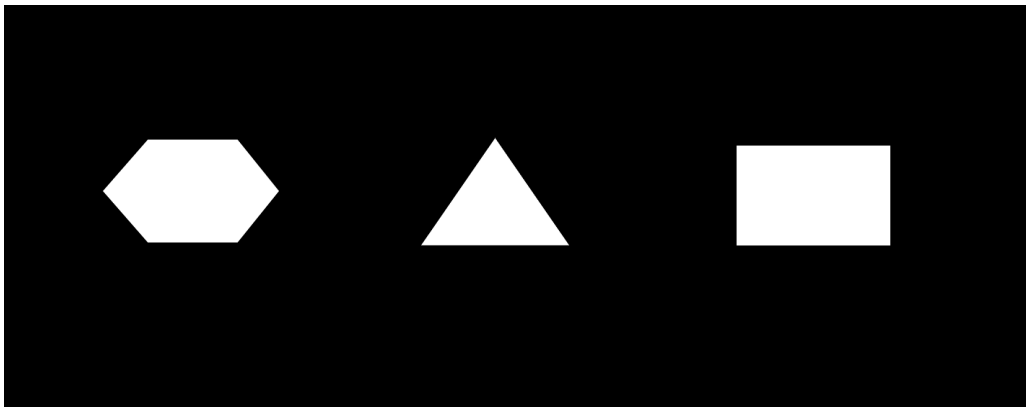
Figure 4.8: Spectrogram with strong noise, discretized with different $K$ values.

Figure 4.8 shows a discretization example made on spectrograms with strong noise (small horizontal white regions marked in red) on main order, with different values

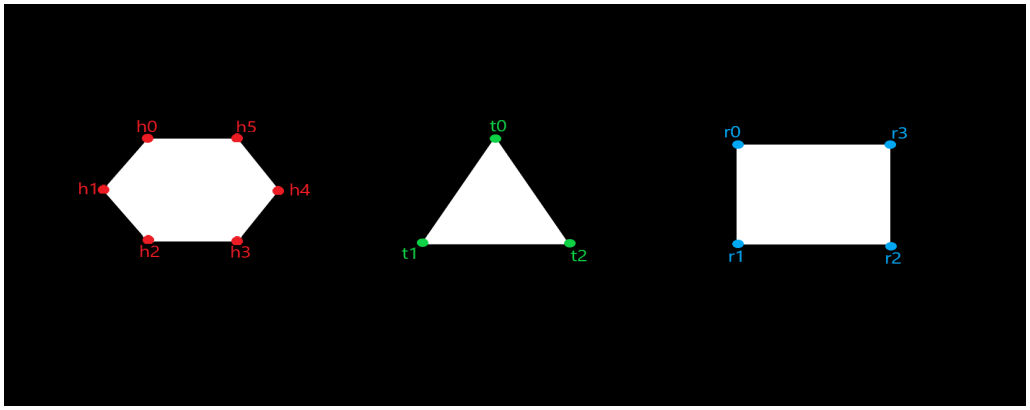
of $K$. When $K = 5$, the noise on the upper part of image, is still too evident. Instead, when $K = 7$, the noise is reduced a lot, but the excessive discretization impacts also on main order contour. Indeed, it presents a great hole in the middle which splits it in two parts. Instead, for $K = 6$, the noise is reduced enough, the holes are still present but they are smaller and the trend of the contour in that area is clearer. Hence, we decided to choose 6 as best value. For what concerns values smaller or bigger than the ones in the set of $K$ analysed before, they would produce respectively noisy (for having too gray scale value in one cluster) or too fine images (that can cause more holes on the main order).

### 4.1.4 Contour detection

Having the discretizated image, the next step is to identify the contours of each white region, where main order can appear in. To do that we exploited the image processing algorithm called Suzuki85 [8]. Given as input the image previously discretized, it produces a list of contours, each represented by a closed polygon separated by other contour by black pixels. A contour is defined as an array of coordinates $(x, y)$. Approximating a polygon with consecutive segments, each coordinate stored in the vector represents connection points between segments.



(a) *Binary image*




(b) *Points saved*

Figure 4.9: Example of points saved by Suzuki85 algorithm

For example, considering the polygons in Figure 4.9 the algorithm will produce a list of three contours

$$contours = [ply1, ply2, ply3]$$

each being a sequence of points

$$ply1 = [h0, h1, h2, h3, h4, h5, h6]$$

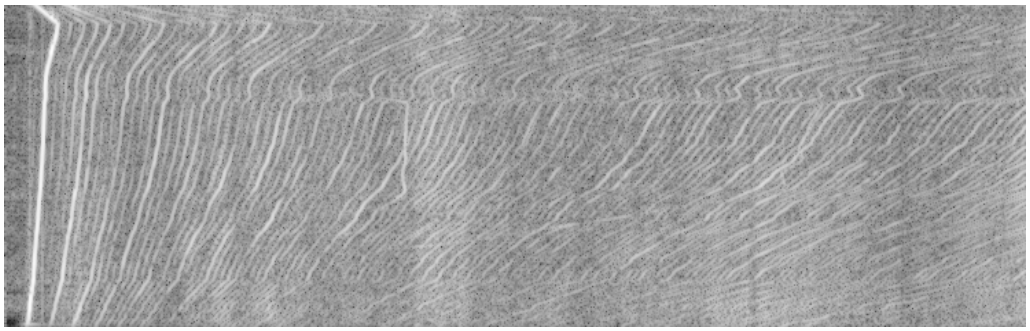
$$ply2 = [t0, t1, t2]$$

$$ply3 = [r0, r1, r2, r3]$$

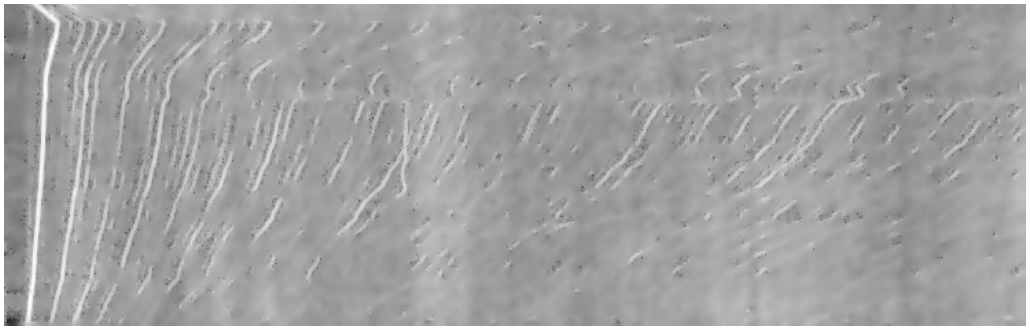
Joining the previous and next point in the sequence, the original contour can be reconstructed. Each point represents a coordinate in the image, for example

$$h0 = (100, 30)$$

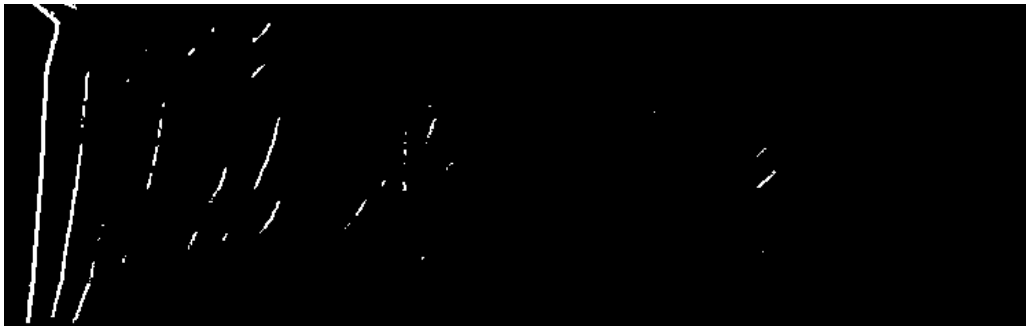
where 100 stands for x-coordinate and 30 for y-coordinate. By applying Suzuki85 algorithm to the discretized image, a set of contours is obtained. This set contains the main order that must be selected in the next phase.



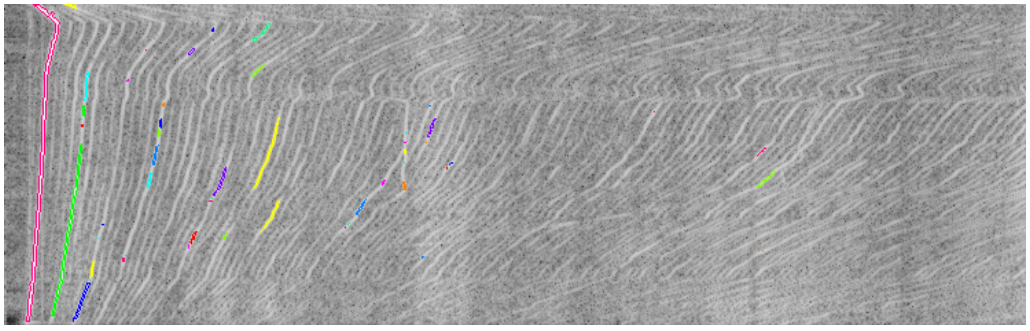
(a) *Original spectrogram*



(b) *Preprocessed spectrogram*



(c) *Discretized spectrogram*



(d) *Contours on original spectrogram*

Figure 4.10: Example of find contour algorithm applied to a spectrogram

Figure 4.10 depicts contours identification in a spectrogram, preprocessed by image filtering (described in Section 4.1.1) and normalization (described in Section 4.1.2), discretized (as described in Section 4.1.3) to obtain the binary image. Each contour shown in Figure 4.10 with different colour, is saved in a list. The red contour is the main order to be found in the next phase of the process.

### 4.1.5 Contours filtering

Contours saved, as described in Section 4.1.4, must be filtered in order to recognize easily the main order contour. To do that, we studied the main order features. Except for being the most intense in magnitude, it must be also inside a specific frequency window, defined by both minimum value of RPM of the engine on idle and maximum RPM of engine reached during the audio recording, described in Section 2.2. Given the typically minimum idle speed value of the engine hovers between $iddleRPM_{min}$ and $iddleRPM_{max}$, taking a margin for some exceptions below this threshold, we considered $iddleRPM_{min} - \delta_1$ as minimum value for RPM. Instead, for what concerns the maximum value, the process for recording engine sound described in Section 2.2, expects to reach about $RPM_{max}$. Hence, taking again a margin for inaccurate measures, we chose $RPM_{max} + \delta_2$ as maximum value for RPM. Finally, the conversion for obtaining frequencies values from RPM is done reversing the formula

$$RPM = f_n \cdot \frac{60}{n}$$

where $f_n$ is the frequency and $n$ is the selected order. The reversed formulas become

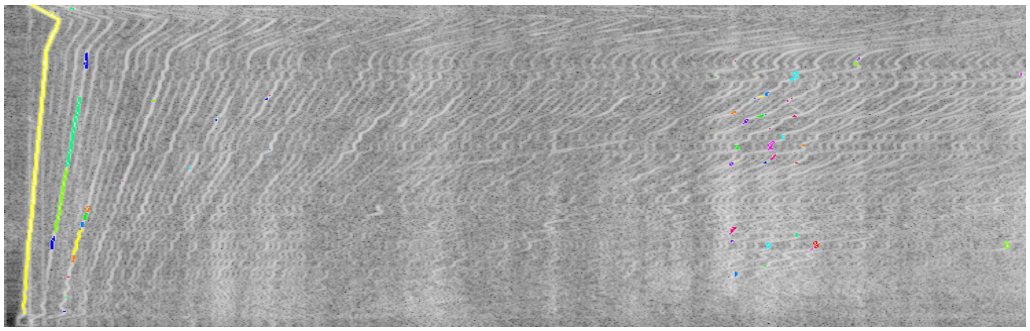
$$f_{nmin} = RPM_{min} \cdot \frac{n}{60}$$
$$f_{nmax} = RPM_{max} \cdot \frac{n}{60}$$

where $RPM_{min} = iddleRPM_{min} - \delta$, $RPM_{max} = RPM_{max} + \delta_2$ and $n = main\ order\ number$. Finding the left and right frequency window limits, each contour, to be a main order, must be included in these limits.

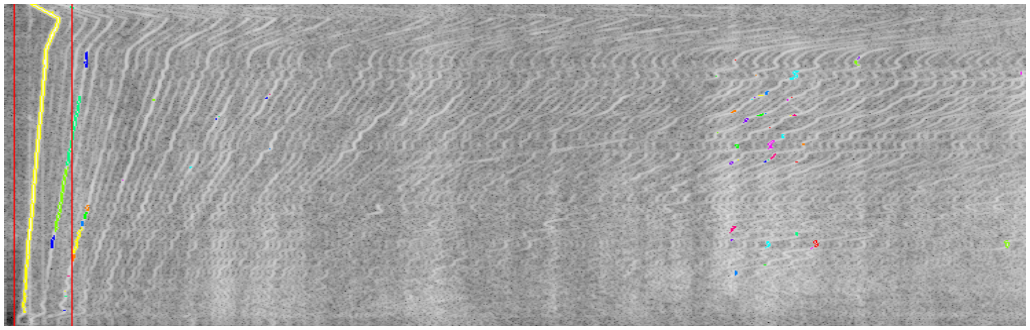
Moreover, to avoid noise points, selected contour must have an $area \geq 2$. The contour area is expressed as total number of pixels inside the contour. The described process can be formalized with the Algorithm 2.

**Algorithm 2** Filter Contours

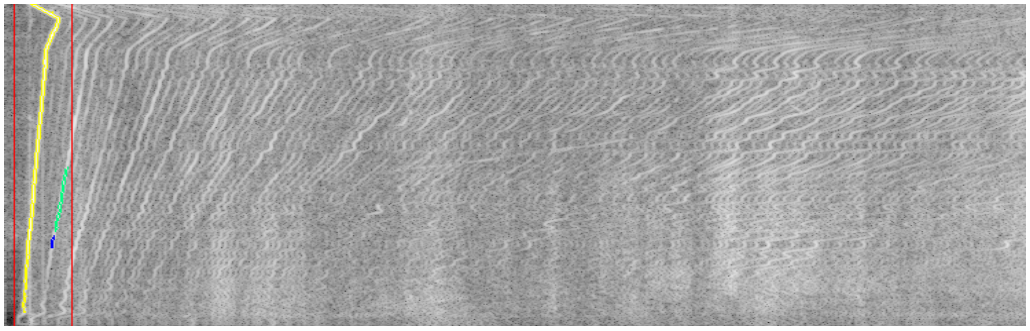
**Require:** candidate contours **and** fnmin **and** fnmax
  create empty list of filtered contour
  **for all** contour $\epsilon$ candidate contours **do**
    $contourBottommostFrequency \Leftarrow$ bottommost frequency of contour
    $contourRightmostFrequency \Leftarrow$ rightmost frequency of contour
    $contourArea \Leftarrow$ contour area
    **if** $contourBottommostFrequency > fnmin$ **and** $contourRightmostFrequency < fnmax\ AND\ contourArea > 2$ **then**
      add contour to filtered contour list
    **end if**
  **end for**
  **return** filtered contours



(a) *Candidate contours*



(b) *Frequencies window*



(c) *Filtered contour*

Figure 4.11: Example of contour filtering

Figure 4.11 shows all the steps of Algorithm 2 applied to an example image. Figure 4.11a shows all the contours found by the contour detection step, described in Section 4.1.4. Figure 4.11b depicts, the frequencies window, drawn with vertical red lines. In the last picture, Figure 4.11c, it is shown how all the contours outside the window are filtered. Small contours inside the window, near the right red line, with area less than 2 pixels are also removed.

Bottommost frequency of a contour is obtainable by taking the point of the contour having the maximum y-coordinate and minimum x-coordinate in case of same values of y. This reasoning assumes that the coordinate $(0, 0)$ is located at top-left corner of the image. Once obtained x value representing the bottommost frequency position in image, it is converted in frequency by taking the value on the vector of frequencies (described in Section 2.4) at position x. Similarly, rightmost frequency of contour corresponds to its maximum x-coordinate. The corresponding frequency is obtained, as for the bottommost one, accessing the vector of frequencies.

Bottommost frequency is compared to $f_{min}$ because it represents the RPM speed ($RPM_{min}$) when the manoeuvre starts, at idle time. We do not use the leftmost frequency to avoid taking possible horizontal noise, such as in Figure 4.8. This one could go beyond on the left than bottommost point and can be erroneously taken as frequency at idle time. Instead, the rightmost frequency is compared to $f_{max}$ because it represents the maximum RPM speed reached during the acceleration manoeuvre for recording the audio track. We do not use the topmost frequency to avoid taking frequencies recorded at the end of the manoeuvre, when the engine starts decelerating. Frequencies at that time, would be instead lower than the $RPM_{max}$ reached.

### 4.1.6 Max intensity selection

The aim of this section is to find the main contour among the filtered contours described in Section 4.1.5. The criterion chosen to distinguish the main order from the others is that, for definition, is the most intense in magnitude. Intensity of each filtered contour must be evaluated and the maximum one should be chosen. To achieve this task, we followed the following steps:

- a black image with the same size of original spectrogram is created for each filtered contour

- each contour is filled with the maximum gray scale value (i.e. 255 for byte-depth images)

The result is a *mask matrix*, for each contour, having 0 outside analysed contour and 255 inside it. By applying a *bitwise and* operator between the original image and the mask, the result will be a matrix containing pixel intensities only where contour is defined, otherwise 0.

$$Image = \begin{bmatrix} 25 & 127 & 135 & 156 & 201 \\ 34 & 112 & 131 & 122 & 222 \\ 44 & 112 & 166 & 122 & 214 \\ 54 & 112 & 154 & 163 & 201 \\ 111 & 137 & 125 & 113 & 180 \end{bmatrix} \quad Mask_c = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 255 & 0 & 0 \\ 0 & 255 & 255 & 255 & 0 \\ 0 & 255 & 255 & 255 & 0 \\ 255 & 255 & 255 & 255 & 255 \end{bmatrix}$$

$$MaskedImage = Image \ \& \ Mask_c = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 131 & 0 & 0 \\ 0 & 112 & 166 & 122 & 0 \\ 0 & 112 & 154 & 163 & 0 \\ 111 & 137 & 125 & 113 & 180 \end{bmatrix}$$

$Image$ is the spectrogram and $Mask_c$ is the mask for contour $c$.

Finally, the contour intensity is just the sum of all pixels values of the masked matrix

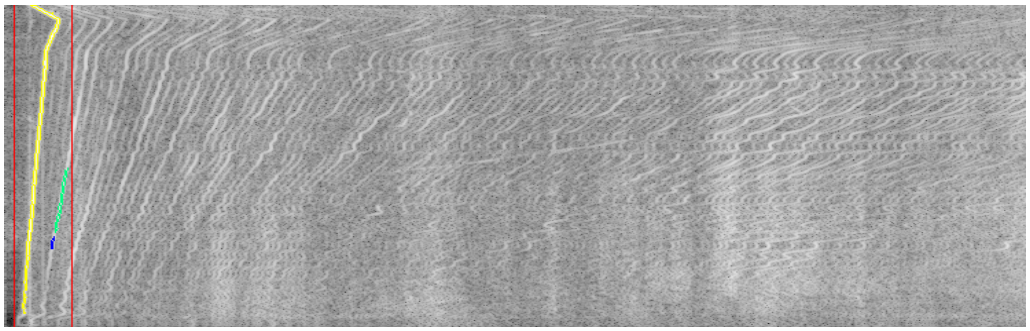
$$MaskedImage = \begin{bmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1,c} \\ p_{2,1} & p_{2,2} & \cdots & p_{2,c} \\ \vdots & \vdots & \ddots & \vdots \\ p_{r,1} & p_{r,2} & \cdots & p_{r,c} \end{bmatrix}$$

$$Intesity = \sum_{i=0}^{r} \sum_{j=0}^{c} MaksedImage_{ij}$$

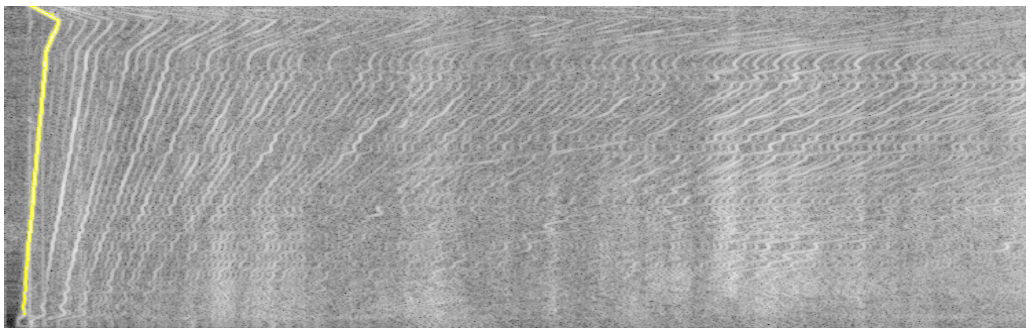
where $MaskedImage$ is the masked matrix.

Using the sum operator, instead of the average intensity allows the process to consider the area of contour meaningful to chose the main order. This avoids taking small contour as main order, that could be just noise or small pieces of main order shape, caused by holes after discetization phase.

At the end of this process, the contour with maximum intensity among filtered contours is chosen and labelled as main order contour.



(a) *Filtered contour*



(b) *Max intensity contour*

Figure 4.12: Max intensity choice

Figure 4.12 shows with a yellow contour the selected main order. Since the small blue contour has few pixels with maximum value (i.e. 255), if the algorithm took into account just the average intensity instead of the sum, it would wrongly select that contour as the main order.

### 4.1.7 Main order parts merging

The discretization phase, described in Section 4.1.3, could introduce holes on the main order, which would split it in distinct contours. This could happen due to a non-homogeneous gray intensity of the whole main order.

Since our final aim is the constant tone identification, the RPM range of the detected main order contour, must cover at least the range between idle speed ($RPM_{min}$) and $RPM_{max}$ that is the speed that must be reached during the recording phase, described in Section 2.2. Indeed, in addition to select the contour having the maximum sum of all pixels intensity, the process checks the range of RPM that it covers. We decided to consider that the detected main order has not to be merged if it covers at least the range $[RPM_{min}, RPM_{max}]$ Hence, constant tones can be found in this range if they are present, as pointed out by domain experts. Algorithm 3 formalizes the procedure for choosing whether the maximum intensity contour should be merged with other contours, to obtain the main order.

**Algorithm 3** Is max intensity to be merged

**Require:** max intensity contour **and** main order n **and** RPMmin **and** RPMmax
  $bottommostFrequency \Leftarrow$ bottommost frequency of max intensity contour
  $rightmostFrequency \Leftarrow$ rightmost frequency of max intensity contour
  $bottommostRPM = bottommostFrequency \cdot 60$/main order n
  **if** $bottommostRPM \leq RPMmin$ **and** $rightmostRPM \geq RPMmax$ **then**
    **return false** {contour not to be merged}
  **else**
    **return true** {contour to be merged}
  **end if**

In Algorithm 3, bottommost frequency and rightmost frequency of max intensity contour are computed in the same way done for Algorithm 2 in Section 4.1.5.

If the maximum intensity contour must be merged, other parts of main order can be searched in contours filtered, as explained in Section 4.1.5. In order to perform this operation, filtered contours are split into contours *above* and *below*

the maximum intensity contour. Remembering that the $(0, 0)$ coordinate is on top-left position of the image, above contours have the bottommost y-coordinate smaller than topmost y-coordinate of maximum intensity contour, while below contours have the topmost y-coordinate below than the bottommost y-coordinate of maximum intensity contour. Algorithms 4 and 5 explain more formally the selection of below and above contours respectively.

**Algorithm 4** Below contours

**Require:** filtered contours **and** max intensity contour
  belowContours = empty list
  $maxContourBottommostY \Leftarrow$ bottommost y-coordinate of max intensity contour
  **for all** contour $\epsilon$ filtered contours **do**
    $contourTopmostY \Leftarrow$ topmost y-coordinate of contour
    **if** $contourTopmostY > maxContourBottommostY$ **then**
      $belowContours.add(contour)$
    **end if**
  **end for**
  **return** belowContours

**Algorithm 5** Above contours

**Require:** filtered contours **and** max intensity contour
  aboveContours = empty list
  $maxContourTopmostY \Leftarrow$ topmost y-coordinate of max intensity contour
  **for all** contour $\epsilon$ filtered contours **do**
    $contourBottomostY \Leftarrow$ bottommost y-coordinate of contour
    **if** $contourBottomostY < maxContourTopmostY$ **then**
      $aboveContours.add(contour)$
    **end if**
  **end for**
  **return** aboveContours

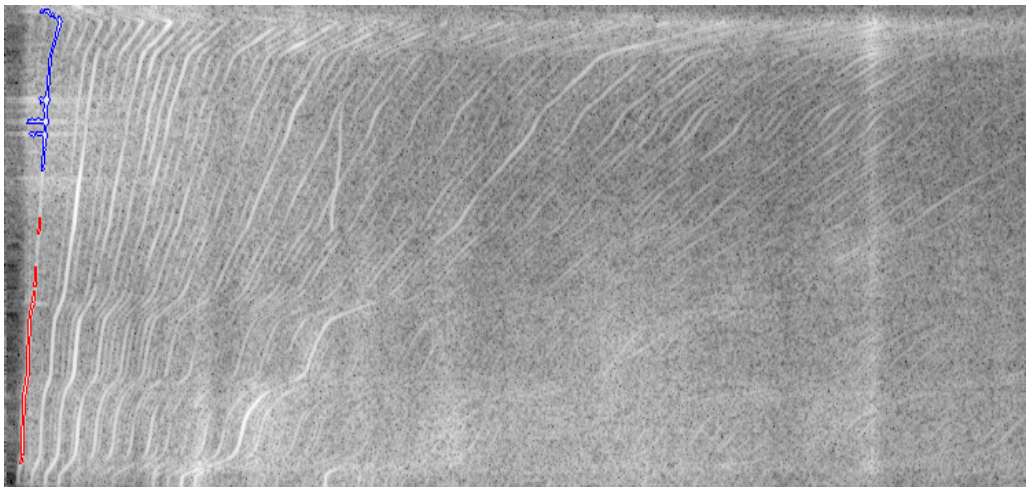


Figure 4.13: Example of below contours each coloured in red. The maximum intensity contour is shown in blue

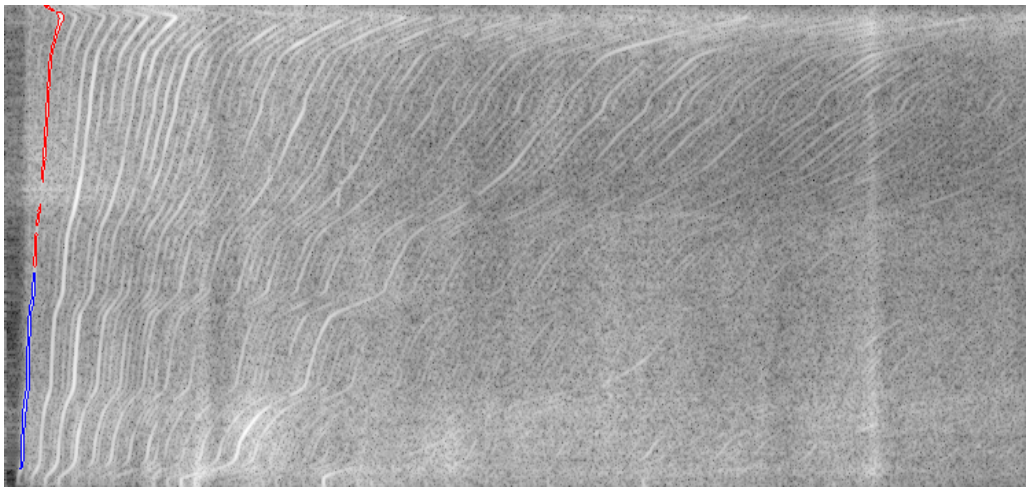


Figure 4.14: Example of above contours each coloured in red. The maximum intensity contour is shown in blue

Figure 4.13 and 4.14 depict the obtained lists of contours, drawn in red, respectively below and above the max intensity contour shown in blue. Having the *below* list, the algorithm as first step, sorts all below contours from top to bottom and creates a chain of contours starting from the maximum intensity one. To do this, it joints the contours starting from the first one of the sorted list. While building the chain, for each contour of the list, it evaluates if its topmost y-coordinate is below than

bottommost y-coordinate of last contour in the chain and if its bottommost x-coordinate is on the left than bottommost x-coordinate of last contour in the chain. This is done because each portion of the same order, should have an incremental trend, given the acceleration performed when recording data. Hence, the starting point of a contour to be added to the chain, has to be on the left side than the starting point of upper contours belong to the same order.

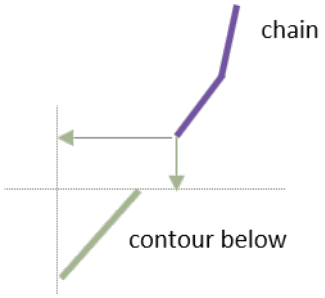



Figure 4.15: Example of below contour

Figure 4.15 shows an example of a contour (in green) below the chain (in purple) for which the algorithm decides whether it should be added. Green arrows represents the constraints to be satisfied. This is repeated for all below contours. At the end, the final chain will represent the below contours to be joined. Algorithm 6 shows the complete procedure.

**Algorithm 6** Below chain

**Require:** below contours **and** max intensity contour
  sort below contours from bottom to top
  *belowChain* = empty list
  *belowChain.addTop*(max intensity contour)
  **for all** contour $\epsilon$ below contours **do**
    *contourTopmostY* $\Leftarrow$ topmost y-coordinate of contour
    *contourBottommostX* $\Leftarrow$ bottommost x-coordinate of contour
    *chainBottommostX* $\Leftarrow$ bottommost x-coordinate of chain
    *chainBottommostY* $\Leftarrow$ bottommost y-coordinate of chain
    **if** *contourTopmostY* $>$ *chainBottommostY* **and** *contourBottommostX* $<$ *chainBottommostX* **then**
      *belowChain.addBottom*(*contour*)
    **end if**
  **end for**
  **return** belowChain

For what concerns the above list, the algorithm as first step, sorts all the above contours from bottom to top and creates a chain starting from the maximum intensity contour, similarly to below contours. Then, starting from the first contour of the sorted list, it evaluates if its bottommost y-coordinate is above than topmost y-coordinate of last contour in the chain and if its rightmost x-coordinate is on the right than topmost x-coordinate of last contour in the chain. This is done for the same reason explained for below contours, taking into account the incremental trend of an order, from the bottom to the top of the image.

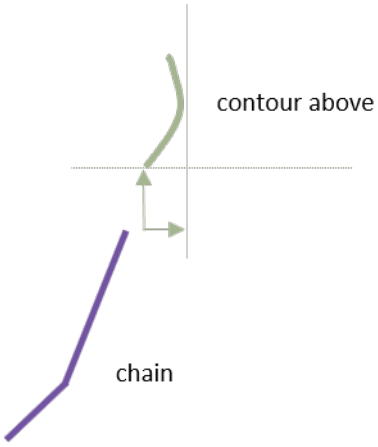



Figure 4.16: Example of above contour

Figure 4.16 shows an example of a contour (in green) above the chain (in purple) and the way our algorithm decides if it is valid or not. If it is valid, it will be added to the chain, otherwise not. This is repeated for all above contours and the final chain represents the above contours to be joined to form the upper part of the main order. The algorithm is formalized in Listing 7.

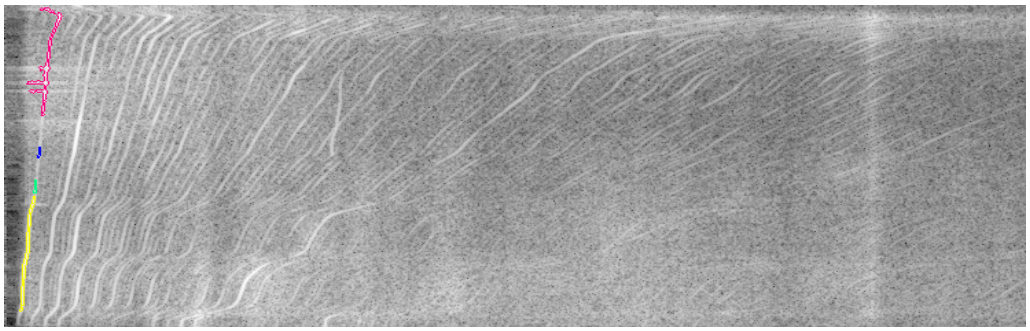
**Algorithm 7** Above chain

**Require:** above contours **and** max intensity contour
  sort above contours from top to bottom
  aboveChain = empty list
  *aboveChain.addBottom*(max intensity contour)
  **for all** contour $\epsilon$ contours above **do**
    $contourBottommostY \Leftarrow$ bottommost y-coordinate of contour
    $contourRightmostX \Leftarrow$ rightmost x-coordinate of contour
    $chainTopmostX \Leftarrow$ topmost x-coordinate of chain
    $chainTopmostY \Leftarrow$ topmost y-coordinate of chain
    **if** $contourBottommostY < chainTopmostX$ **and** $contourRightmostX > chainTopmostX$ **then**
      *aboveChain.addTop*(*contour*)
    **end if**
  **end for**
  **return** aboveChain

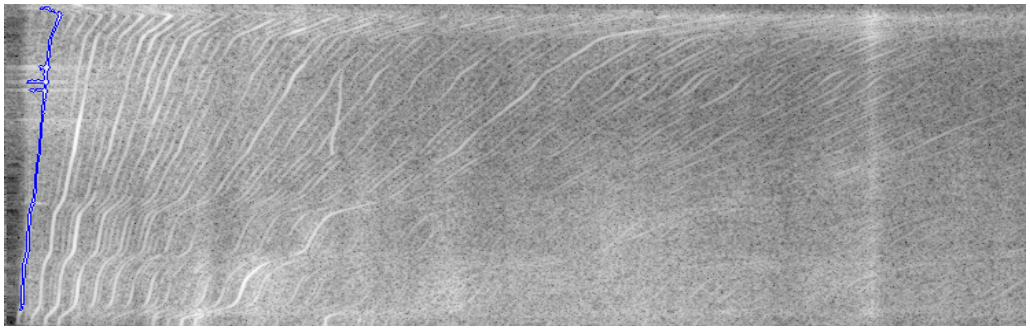
Both chain lists created are then concatenated in order to obtain the final chain

of contours to be merged.

The main contour may have a cut at the bottom, at the top or both of them. Moreover, left and right points in the cut part/s to be merged must be decided. Sorting the points of all contours counter-clockwise, left and right points in the bottom cut are decided taking first the bottommost point. Then, the left point to be merged will be the previous point than the bottommost one, if it is on the left otherwise, the bottommost itself is chosen. Instead, right point will be the next point than the bottommost one if it is on the right otherwise, the bottommost itself is chosen. For what concern a top cut, the topmost point is taken and the left point to be merged will be the next point than the topmost one, if it is on the left otherwise, the topmost itself is chosen. Instead, right point will be the previous point than the topmost one if it is on the right otherwise, the topmost itself is chosen. To know where cuts are, the chain list of contours to be merged is sorted, top to bottom or bottom to top. Taking for example the first sorting type, the first contour of the sorted list will present a cut in the bottom part, the last on top part and if there are other contours in the middle of the list, they will present cut in both parts. Once deciding all the left and right points to be joined, the final merged contour represents the whole main order, which will be used in the next phase of our process.



(a) *Chain lists to merge*



(b) *Chain list merged*

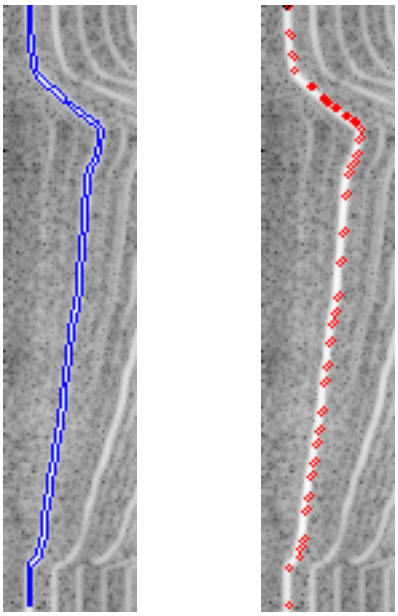
Figure 4.17: Contours merged

Figure 4.17a shows the chain list of contours to merge, coloured with different colours, while Figure 4.17b the final merged contour in blue. It reconstructs the main order, solving the problem of interrupting holes created by discretization phase.

## 4.2 Anchors identification

Having completed the main order identification, the next step is to choose its anchors, that are points typically drawn manually by domain experts, useful for the time-to-RPM conversion. They must follow the trend of the main order, as shown in Figure 1.3 of Section 1.2.2. To do that, our process exploits the main order contour points obtained by the contour detection algorithm described in Section 4.1.4. Indeed, following the contour shape, they chase the variation of the main order. Then, our process considers, as valid anchors, the right margin of the main

order contour, starting from bottommost point and going to the topmost (counter-clockwise order).



(a) (b)

Figure 4.18: Contour (a) to anchors (b)

However, before proceeding to time to RPM process, anchors must be processed to remove useless segments and approximate the main order with the smallest set of points.

### 4.2.1 First constant frequencies removal

The first points, starting from the bottommost point of the anchors, may represent constant frequencies, due to recording idle speed before the acceleration.

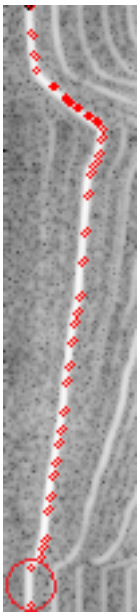
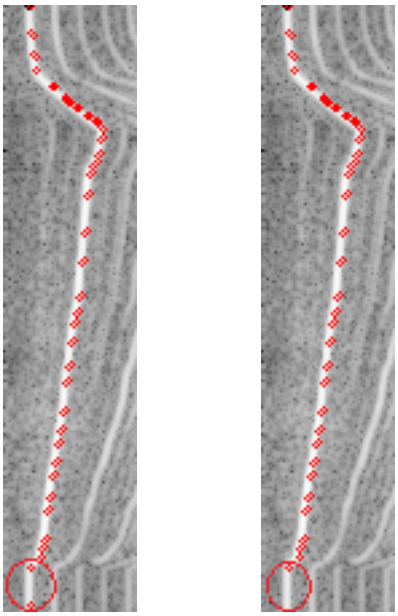


Figure 4.19: Constant anchors

As Figure 4.19 depicts, the first two anchors have the same x-coordinate, representing the idle speed. This area of the image is useless, because only points representing an acceleration are meaningful for the conversion to RPM domain. Indeed, the conversion will consist of straightening the main order shape, starting from those points. For this reason, our procedure, scanning the anchors from bottom to top, removes them until the next one has a different x-coordinate.



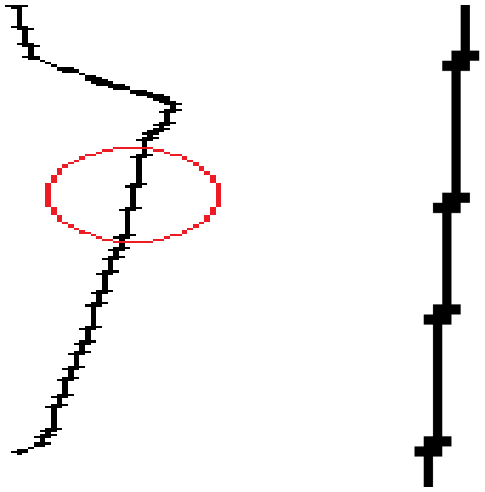
(a) (b)

Figure 4.20: Constant anchors (a) removed (b)

For example, as shown in Figure 4.20, the first anchor is removed, because the anchor after that one is on the same x-coordinate.

### 4.2.2 Anchors approximation

The anchors extracted from the right contour, chase its variations too finely, due to the high number of points that follow the contour shape.



(a) (b)

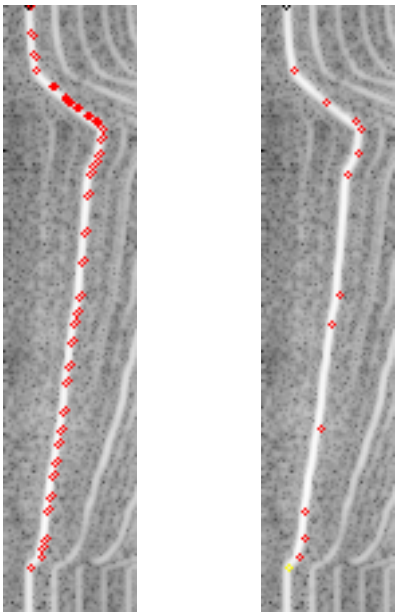
Figure 4.21: Right contour shape (a) zoomed (b)

For example, as Figure 4.21 shows, lines created by joining consecutive anchors are collinear segments. Instead, a single diagonal line would better approximate the acceleration phase, decrementing the number of vertices that depict the right contour shape.

Figure 4.22: Three collinear lines approximated with a single segment

Figure 4.22 clarifies with an example the approximation described before. Indeed, the three collinear lines are approximated with just one diagonal line, the blue one in the picture.

In order to perform this, we exploited the Douglas-Peucker Algorithm [9]. It is an heuristic method that complies with our target, indeed it approximates a sequence of points with a segment, starting from the first and ending to the last point of that sequence. The criterion for deciding if a line approximates a set of points, is an $\epsilon$ parameter, that indicates the minimum distance that a line must have to approximate two points. If the line between the two extreme points has a distance grater than $\epsilon$, the algorithm will proceed recursively by including other intermediate points in the approximation. For deciding $\epsilon$ parameter in our case, we adopted an heuristic approach, trying different values. We established 1 as $\epsilon$ value. It gives a good approximation without loosing the quality of the contour shape.



(a) (b)

Figure 4.23: Anchors before (a) and after (b) approximation, with $\epsilon = 1$

Figure 4.23 shows the reduction of points performed by Douglas-Peucker Algorithm using $\epsilon = 1$ and how approximating points are follow correctly the shape of the main order.

### 4.2.3 Decrementing frequencies removal

The last anchors on the top of the image, may present segments increasing from bottom-right to top-left. These lines represent a deceleration recorded during the collection process described in Section 2.2. This could be happen by continuing recording the engine sound for some moments after the acceleration is finished. As said previously in Section 4.2.1, only anchors representing the acceleration are meaningful for the conversion to RPM domain, therefore those anchors must be removed.

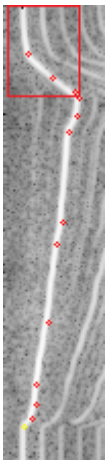
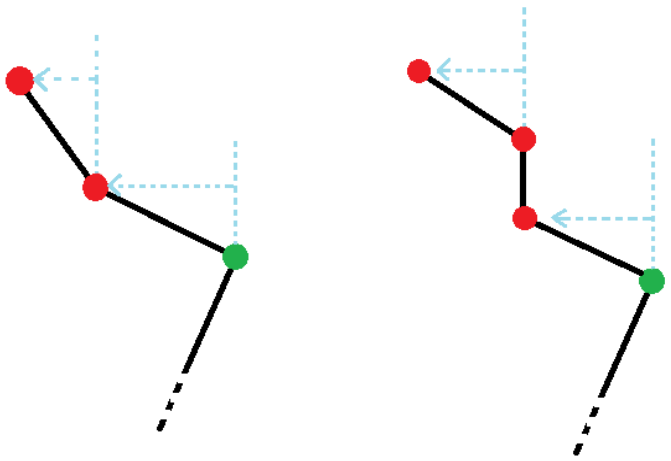


Figure 4.24: Anchors of the deceleration phase, highlighted with a red rectangle.

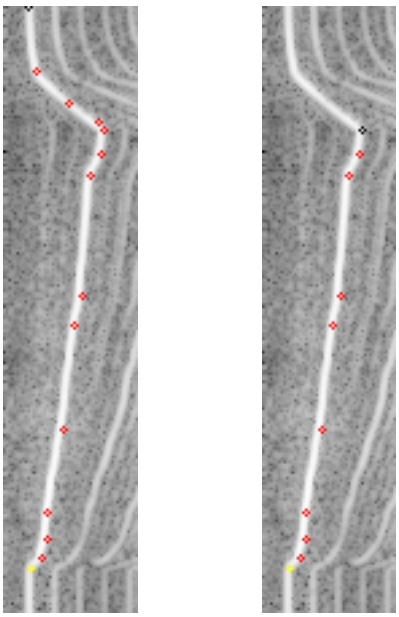
Anchors describing the deceleration phase, shown inside the red rectangle in Figure 4.24, must be recognized and removed. Sorting all anchors in a list from bottom to top, in our process an anchor is considered as the starting point of deceleration phase if the next two anchors are both on the left of their bottom anchors. When checking the next two anchors, our process starts counting, skipping those which describe vertical segment, as they do not represent a deceleration. This means selecting either two consecutive lines increasing from bottom-right to top-left, skipping some possible vertical segments between them, or a line increasing from bottom-right to top-left just followed by a single vertical segment. Once the procedure finds the staring anchor of deceleration phase, all the next one are deleted. The reason why the procedure considers at least two anchors on the left and not just one is that during the acceleration phase of the process for collecting data described at Section 2.2, the operator can involuntarily do a little deceleration, causing a small lines increasing right to left. However, this line does not represent the phase to remove that is when the operator take his foot off the gas. The only case when just one anchor on the left is enough to stop the procedure is when it is the last one. In this case just that anchor will be removed.



(a) (b)

Figure 4.25: Example of anchors removed by the process

Figure 4.25 depicts examples of starting points of the deceleration phase (marked in green) and anchors that will be removed (red points) by our process.



(a) (b)

Figure 4.26: Anchors representing deceleration zone removed by the process

The black anchor in Figure 4.26b represents the beginning of the deceleration.

## 4.3 Time to RPM

Having the anchors extracted and processed, as described in Section 4.2, the process must mimic the time to RPM process described in Section 1.2.3. In summary, it consists in straightening the main order and, as a consequence, all the other orders, to let domain experts apply known analyses, applicable only to RPM domain.

Therefore, our process exploits the anchors extracted so far. Since they represent the main order trend, straightening anchors means, as a result, straightening also the main order itself. Remembering that adjacent anchors can be seen as the starting and ending point of a segment, accomplishing the alignment needs all segments to have the same angular coefficient. In order to make this, the image is divided in horizontal sections defined by the anchor positions. Each section represents a *time frame* where the shape of the main order is approximately a straight line with a specific angular coefficient. In our process, sections are reshaped vertically to modify the angular coefficient of their associated main order segment. In this way the time spent to perform the acceleration ramp inside each section is compressed or extended.

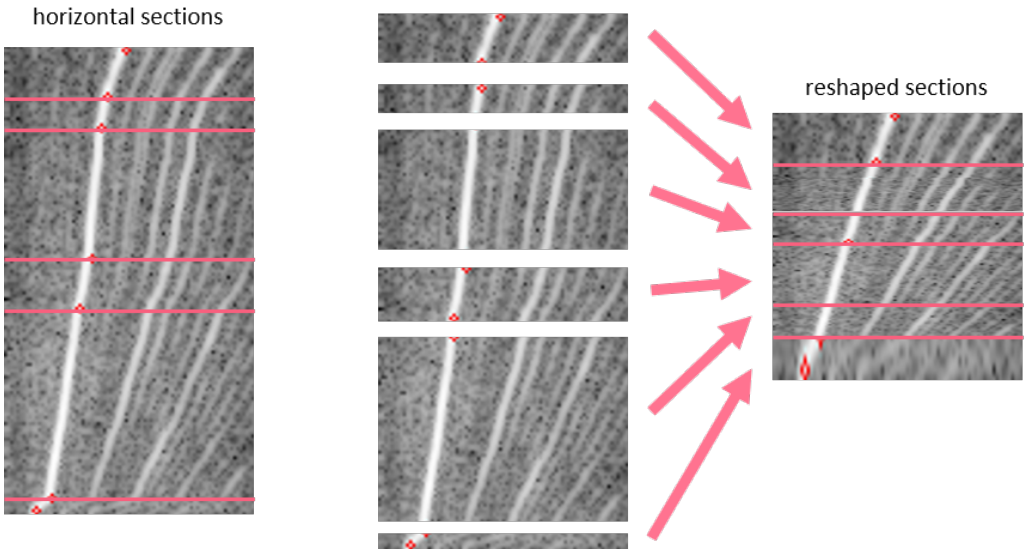



Figure 4.27: Sections reshaping

Figure 4.27 shows a piece of main order divided in sections, and how they must be reshaped to obtain a straightened out line.

### 4.3.1 Target angular coefficient

To make all segments have the same angular coefficient, we analysed two possibilities: decide a priori a value for all images or compute it ad hoc for each image. The first approach is faster, not requiring any computation, but it can badly affect the quality of images by stretching or compressing more or less than needed. Instead, for what concerns the second approach, an easy method to minimize the reshaping of images is to choose the mean of angular coefficients of all the segments, defined by anchors, as target angular coefficient. Describing this with a procedure:

**Algorithm 8** Target angular coefficient

**Require:** $anchorsList$
  $sum = 0$
  $count = 0$
  **for** $i = 0$ **to** $anchorsList.size - 2$ **do**
    $anchorBottom \Leftarrow anchorsList.get(i)$
    $anchorTop \Leftarrow anchorsList.get(i + 1)$
    **if** $anchorBottom.X \neq anchorTop.X$ **then**
      $angularCoefficient = (anchorTop.Y - anchorBottom.Y)/(anchorTop.X - anchorBottom.X)$
      $sum = sum + angularCoefficient$
      $count = count + 1$
    **end if**
  **end for**
  $targetAngularCoefficient = sum/count$
  **return** $targetAngularCoefficient$

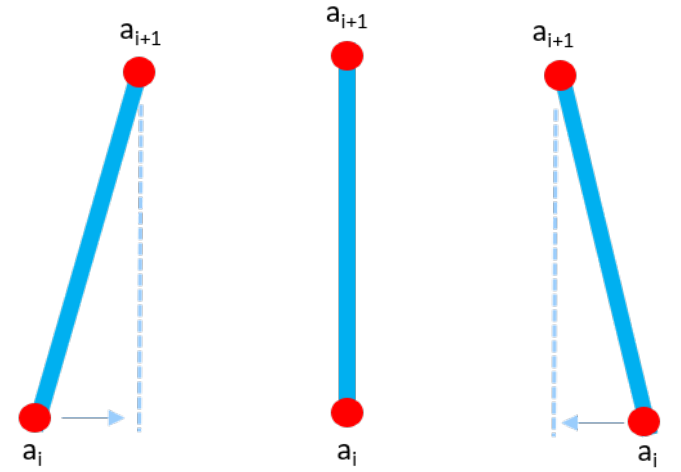
Then, Algorithm 8 is the approach chosen for our procedure.

### 4.3.2 Conversion process

Once having target angular coefficient, described in Section 4.3.1, the main process of time to RPM conversion consists in reshaping vertically each horizontal section delimited by two adjacent anchors by inspecting their coordinates. Two adjacent anchors can have the top x-coordinate:

- on the right than the bottom one
- on the same position
- on the left

Each case must be processed differently. Figure 4.28 depicts the three kind of segments that two anchors can generate.



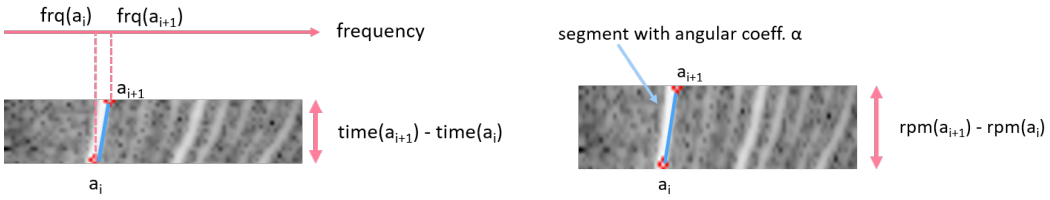

(a) $a_{i+1}$ on right than $a_i$

(b) $a_{i+1}$ on same x-position than $a_i$

(c) $a_{i+1}$ on left than $a_i$

Figure 4.28: Type of segments obtained by different position of anchors

Starting from sections having the top anchors on the right, they suggest an incrementing speed in a time frame.




(a) *Section to reshape (time domain)* (b) *Section reshaped (RPM domain)*

Figure 4.29: Example of section reshaping

Figure 4.29 shows a segment having the top point on the right, and how it must be reshaped. In particular, the segment in RPM domain, between $a_{i+1}$ and $a_i$ must match the target angular coefficient ($\alpha$) computed. Hence, the height of the reshaped section will be in RPM domain:

$$RPM(a_{i+1}) - RPM(a_i) = \alpha \cdot (frq(a_{i+1}) - frq(a_i))$$

Continuing with the second type of section, having top anchors on the same x-coordinate of the bottom one, it indicates a constant speed (or irrelevant variation, approximated with a vertical line) in a time frame. These sections are meaningless for straightening the main order, since the frequency actually remains constant in time. Hence, our process completely compresses these vertical sections to 1 pixel height.

Finally, top anchors on the left than the bottom one represent decrementing speed in a time frame. These sections are also useless, as the previous sections, because only increasing frequencies allow main order to be straightened out. However, differently from constant frequencies, compressing these sections to 1 pixel is not enough. Indeed, also a portion of the next section having frequencies less than frequency reached before the deceleration must be compressed.

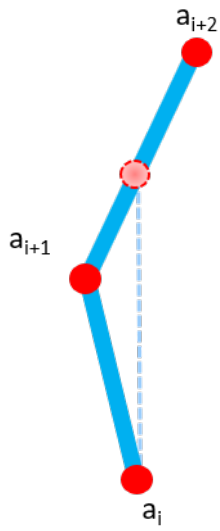



Figure 4.30: Compression due to decrementing speed

Figure 4.30 shows the section where frequencies decrease, between anchors $a_i$ and $a_{i+1}$, and the next increasing section, between anchors $a_{i+1}$ and $a_{i+2}$. The portion of image to compress is included between $a_i$ and the light red anchor between anchors $a_{i+1}$ and $a_{i+2}$. The light red anchor has the same x position of $a_i$ and selects the point where the main order starts to increase again.

For reproducing these three methods on images, our process exploits the remap function, available in image processing libraries. Remapping consists of deforming an input image by stretching its pixels between some reference points, whose source and target position is known. Pixels among reference points are computed with linear interpolation. For example, if two reference points $p1$ and $p2$, taken at the extreme horizontal positions of the image and on the same $y$-coordinate, were be remapped in the target image on the same $x$-coordinates but on $(y + \delta)$-coordinate, the portion of image under $(y + \delta)$ would be compressed if $\delta > 0$, stretched up if $\delta < 0$, unchanged if $delta = 0$.

Therefore, a list sorted from bottom to top containing all the $y$ positions of main

order anchors is created:

$$ySource = [a_0.y, ..., a_i.y, ..., a_{N-1}.y]$$

where $a_i.y$ is the y value of the anchor in position $i$ and $N$ the total number of anchors. This is the list of the original position of pixels to remap. In addition, an other list that contains the destination position of these pixels is created too. Its first element is $image.height - 1$. It is the bottommost part of the image, having:

$$a_0.y \neq image.height - 1$$

due to first constant frequencies removal, described at Section 4.2.1. This let the remap function cut the portion of image where frequency is constant in time. Instead, the next elements are computed as the sum of the previous one by a $\delta$. Hence, the destination list is:

$$yDest = [image.height-1, yDest.get(0)+\delta_1, ..., yDest.get(i)+\delta_{i+1}, ..., yDest.get(N-2)+\delta_{N-1}]$$

To decide the $\delta$ value, the process scans all the anchors, considering each section of the image composed by two anchors at a time. Its value is based on the three kind of section described before.

For sections having the top anchor on the right, $\delta_{i+1}$ is computed as:

$$\delta_{i+1} = \alpha_{target} \cdot (a_{i+1}.x - a_i.x)$$

where $\alpha_{target}$ is the target angular coefficient described in Section 4.3.1, $a_{i+1}.x$ the x-coordinate of the top anchor in the considered section analysed and $a_i.x$ the bottom one.

For sections having the $x$ coordinate of the top anchor on the same position than the bottom one

$$\delta_{i+1} = 1$$

for compressing vertical segments to 1 pixel, as described previously in this section.

Finally, for the last kind of section, having top anchor on the left of the bottom one,

$$\delta_{i+1} = 1$$

as done for the previous type section. However, what changes here is the $ySource$ list. Once a section of this type is detected, between $a_{i+1}$ and $a_i$, the $ySource_{i+1}$ is not equal to $a_{i+1}.y$, but it will be:

$$ySource_{i+1} = \alpha \cdot (a_i.x - a_{i+1}.x) + a_{i+1}.y$$

where $\alpha$ is the angular coefficient of anchors $a_{i+1}$ and $a_{i+2}$ computed as:

$$\alpha = \frac{a_{i+2}.y - a_{i+1}.y}{a_{i+2}.x - a_{i+1}.x}$$

Once having the two list $ySource$ and $yDest$, the final step is to generate the reference points for the remapping function by adding the horizontal extremes to each $ySource_i$ and $yDest_i$ coordinate. The result are two other lists such as:

$$pixelSource = [po_0, ..., po_i, ..., po_{N-1}]$$

$$pixelDest = [pd_0, ..., pd_i, ..., pd_{N-1}]$$

where

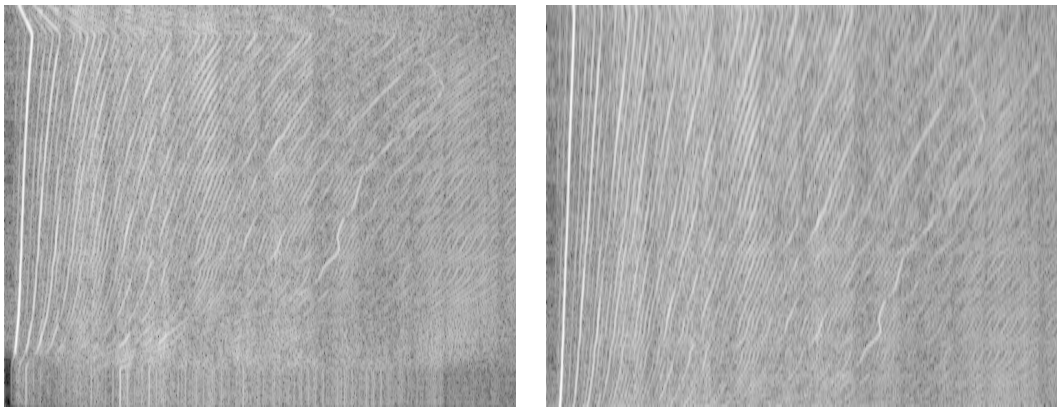
$$po_i = [(0, ySource(i)), ((image.width - 1), ySource(i))]$$

$$pd_i = [(0, yDest(i)), ((image.width - 1), yDest(i))]$$

Then, $pixelSource$ and $pixelDest$ are given to the remap function as reference points. The Algorithm generates the remapped image, which represents the RPM conversion. The described procedure for extracting reference points for the remap function is summarized in Algorithm 9.

**Algorithm 9** Time to RPM

**Require:** $anchorsList$ **and** $alphaWanted$ **and** $imageTime$
  $ySource = []$
  $yDestination = []$
  $pixelSource = []$
  $pixelDestination = []$
  $ySource.add(anchorsList.get(0))$
  $yDestination.add(anchorsList.get(0))$
  **for** $i = 0$ **to** $anchorsList.size - 2$ **do**
    $anchorBottom \Leftarrow anchorsList.get(i)$
    $anchorTop \Leftarrow anchorsList.get(i + 1)$
    **if** $anchorTop.X < anchorBottom.X$ **then**
      $alpha = (anchorsList.get(i+2).y - anchorTop.y)/(anchorsList.get(i+2).x - anchorTop.x)$
      $yNew = alpha \cdot (anchorBottom.x - anchorTop.x) + anchorTop.y$
      $ySource.add(yNew)$
    **else**
      $ySource.add(anchorTop.y)$
    **end if**
  **end for**
  **for** $i = 0$ **to** $anchorsList.size - 2$ **do**
    $anchorBottom \Leftarrow anchorsList.get(i)$
    $anchorTop \Leftarrow anchorsList.get(i + 1)$
    **if** $anchorTop.X > anchorBottom.X$ **then**
      $delta = alphaWanted \cdot (anchorTop.x - anchorBottom.x)$
      $yDest = yDestination(i) + delta$
      $yDestination.add(yDest)$
    **else**
      $delta = 1$
      $yDest = yDestination(i) + delta$
      $yDestination.add(yDest)$
    **end if**
  **end for**
  **for** $i = 0$ **to** $yDestination.size - 1$ **do**
    $pixelS = [(0, ySource.get(i).y), (imageTime.width - 1, ySource.get(i).y)]$
    $pixelD = [(0, yDestination.get(i).y), (imageTime.width - 1, yDestination.get(i).y)]$
    $pixelSource.add(pixelS)$
    $pixelDestination.add(pixelD)$
  **end for**
  $imgRPM = remap(imageTime, pixelSource, pixelDestination)$
  **return** $imgRPM$



(a) *Time domain spectrogram* (b) *RPM domain spectrogram*

Figure 4.31: Example of time to RPM conversion

Figure 4.31 shows Algorithm 9 applied to a time domain spectrogram. As a result, the main order and all the orders are straightened out due to remapping (Figure 4.31b).

### 4.3.3 Axes computation

After the spectrogram computation process, two lists are obtained. One contains for each x-coordinate the corresponding frequency value (Hz) and one contains for each y-coordinate the corresponding time value (s), as described at Section 2.4. Once spectrogram is converted from time to RPM domain, the list of frequencies is still valid, having just vertically scaling the image. Instead for what concern the y-coordinates, they must indicate the RPM values. A new list containing for each $y$-coordinate the corresponding RPM value must be created.

In order to perform this task, our process exploits the straightened out main order. Knowing the $\alpha_{target}$ and its bottommost point ($(x_{bottom}, y_{bottom})$), the process can compute for each $y$-coordinate the $x$-coordinate of the point defined by the intersection of $y$-coordinate and the line passing through $x_{bottom}$ having $\alpha_{target}$ as angular coefficient.

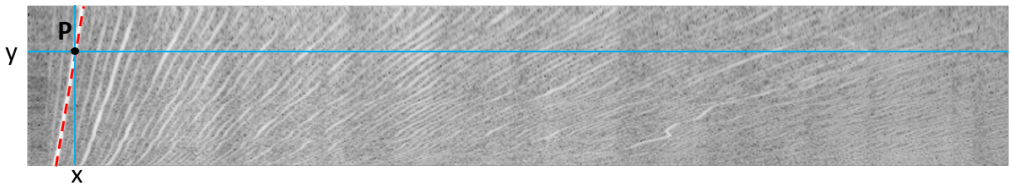



Figure 4.32: Example of intersection

Figure 4.32 depicts the point P, generated by the intersection between dotted red line and $y$. The $x$-coordinate is computed with the formula:

$$x = x_{bottom} + \frac{y - y_{bottom}}{\alpha_{target}}$$

Having this value, the frequency value ($f_x$) of x-coordinate is simply taken by the frequencies list mentioned before. Instead, the RPM value ($RPM_y$) of y-coordinate is obtained with the usual formula:

$$RPM_y = f_x \cdot \frac{60}{n}$$

where $n$ is the known main order number. This operation is performed for all y-coordinates for creating the whole y-to-RPM list.

# Chapter 5

# Constat tone detection

## 5.1 Introduction

In this chapter, we will present the way our process achieves the *constant tone detection*, the main objective of this thesis work. In order to obtain this result, we explore *machine learning techniques* [10]. These techniques aim to identify, analysing selected features, regularities in data called *patterns*, which allow performing data classification. Once having a set of data labelled with a class label, the classification, through patterns, addresses the assignment of unlabelled data to the correct class. Hence, translating this to our work means:

- extracting features from the labelled images, described in Chapter 3. In our process, feature are obtained from all the contours present on those images. However, as for the main order identification phase described in Section 4.1, the images must be first preprocessed to ease the classification
- exploiting the extracted features to *train* a classification model
- *testing* trained model, thanks to the labelled contours, validating how the model predicts a class label for a contour

## 5.2 Image preprocessing

To have good quality patterns, *data preprocessing* is an essential operation. It allows reducing the effect of noise that would negatively impact the result of the classification.

As for easing the main order identification, described in Section 4.1, our process performs the same step for preprocessing the images. In summary, it performs:

- image filtering by applying the *mean shift filter* described in Section 4.1.1. It allows removing noise and highlighting useful informations, such as constant tone contour
- *min-max normalization* technique to have a fixed range of the pixel values in the image, useful also for the discretization step, as described in Section 4.1.2
- *image discretization* by using the *K-means* algorithm, described in Section 4.1.3. It reduces the cardinality of the possible pixel values, in K clusters. However, for the main order identification, being the most intense element in magnitude of the image, our process analyses only the last cluster, containing the highest pixel values of the gray scale. Instead, for taking the feature of more contours for our classification model and trying to include the constant tone ones, our process does not select only the last cluster but a k-subset of the last ones. This helps removing noise, present more in the first clusters, and highlighting more intense contours, where constant tone can be present.
- find all the *contours* of the k-last clusters, by applying Suzuki85 [8] algorithm described at Section 4.1.4, and save them in a set.
- *filter* found contours, based on a frequency window as performed for main order identification, described at Section 4.1.4. Given the frequency range, established by domain experts, in which constant tone appears is $[f_{min}, f_{max}]$Hz.

Choosing a larger frequency range of $[f_{min} + \delta_{min}, f_{max} + \delta_{max}]$Hz, due to possible exceptions, our process considers only contours having leftmost frequency point and rightmost one inside this threshold. Moreover, also contours having area $\leq 2$, are discarded, to remove noise.

## 5.3 Building the ground truth labels

Having the set of filtered contours, our process must label each of them. A contour can be associated to a constant tone label (string "CT") or to a label that represents the other contours (string "OT"). In order to evaluate if a contour has to be classified with the label "CT" or "OT", we exploit constant tone contours, labelled manually thanks to LabelMe tool, described at Section 3.1. Indeed, all the contours of an image are scanned by our evaluation process and, for each of them, it computes the *intersection over union* (IoU) between the analysed contour and the constant tone contour taken from the annotated image. The intersection over union between two contours is typically used as an evaluation metric to evaluate the accuracy of an object detector. It is computed between two contours diving their *intersection* area by their *union* area. It gives as result the percentage of the accuracy.

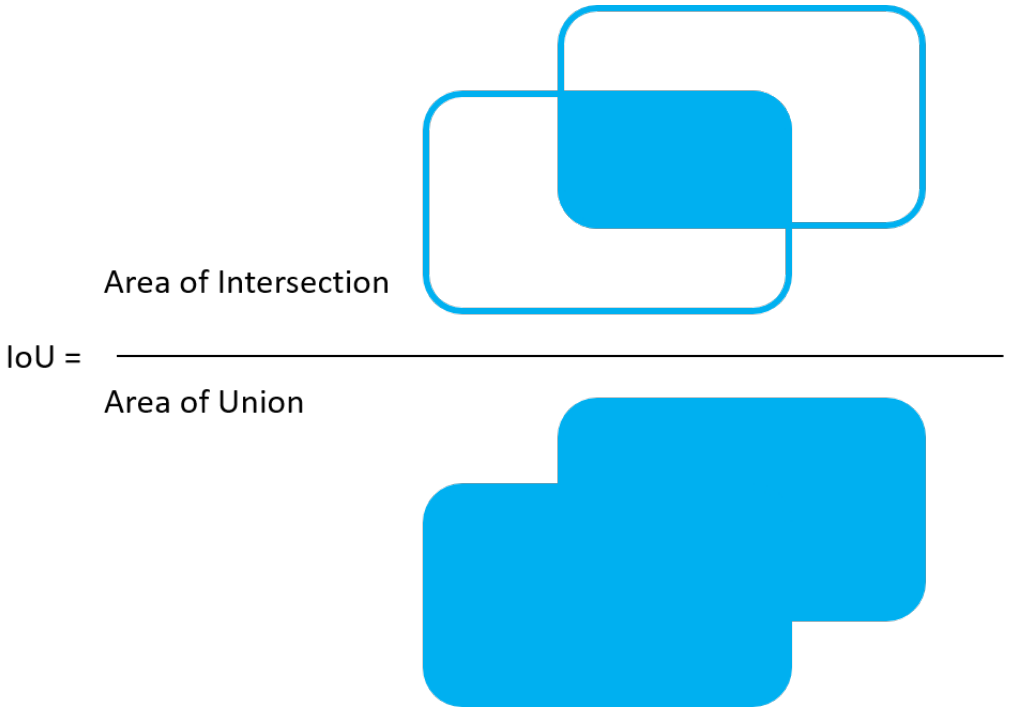



Figure 5.1: Intersection over union between two figures.

Figure 5.1 depict the intersection over union metric, where the upper blue area represents the intersection, while the bottom one the union. A contour has to be labelled as "CT" if the result of the IoU with a manually labelled constant tone is higher than 10% otherwise it is considered as "OT". Hence, at the end of this labelling phase, every contour of all the images will be associated to a label. The set of labelled contour built so far is considered as ground truth for training and evaluating the classifier.

## 5.4 Feature extraction

Before proceeding to the classification process, a procedure to extract features from a contour must be implemented. The extracted features must be chosen in order to obtain valid patterns for discerning a constant tone from other contours. Moreover, remembering that a constant tone is a noise almost constant in frequency,

it typically appears as a vertical line, differently to orders that come out as lines increasing from bottom-left to top-right, following the acceleration speed. Indeed, the features extracted are related to contour shape, its frequency position and also its intensity. Being noise, constant tone could also have an intensity different than other contours have. The set of extracted features is:

- approximating rectangle height
- approximating rectangle width
- min(height, width) / max(heigh, width)
- contour area over approximated rectangle area
- atan2(y, x) (contour orientation)
- contour maximum pixel intensity
- contour intensity sum
- contour intensity average
- contour intensity average over approximated rectangle intensity average

where approximating rectangle is a rotated rectangle with minimum area enclosing the contour as shown in Figure 5.2, while $atan2(y, x)$ is the function

$$atan2(y,x) = \begin{cases} arctan(\frac{y}{x}) & \text{if } x > 0 \\ arctan(\frac{y}{x}) + \pi & \text{if } x < 0 \text{ and } y \geq 0 \\ arctan(\frac{y}{x}) - \pi & \text{if } x < 0 \text{ and } y < 0 \\ +\frac{\pi}{2} & \text{if } x = 0 \text{ and } y > 0 \\ -\frac{\pi}{2} & \text{if } x = 0 \text{ and } y < 0 \\ undefined & \text{if } x = 0 \text{ and } y = 0 \end{cases}$$

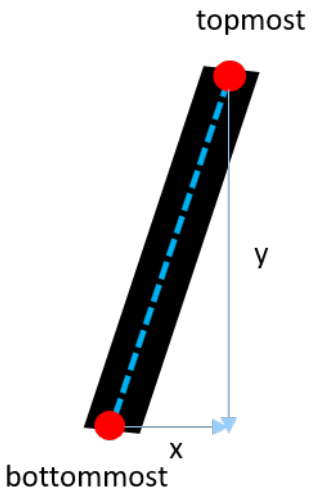



Figure 5.2: Example of $x$ and $y$ value, using the segment cutting the black contour.

Having a segment that cuts in half the contour, we choose as $y$ value, the vertical distance between the rightmost point and the leftmost one of that segments. Moreover, we chose as $x$ value, the horizontal distance between the rightmost point and the leftmost one of the same segment. Hence, $feature(contour)$ procedure, having as input a contour, will return as result all these features, stored in an array.

## 5.5 Classification process

Once having extracted the ground truth contours of all the images, the main classification process consist in three main phases:

- choosing a *classification* algorithm that will perform the classification phase
- *training* the chosen classifier with a subset of images, using the ground truth labelled contours
- *testing* the classifier with an other subset of images, different from the one chosen for the training phase.

### 5.5.1 Classification method

The classification method analysed is the *Decision-tree* [7]. It adopts a greedy strategy based on a tree, as name could suggest. The tree is built, through the training set, by *Hunt's algorithm* [7].

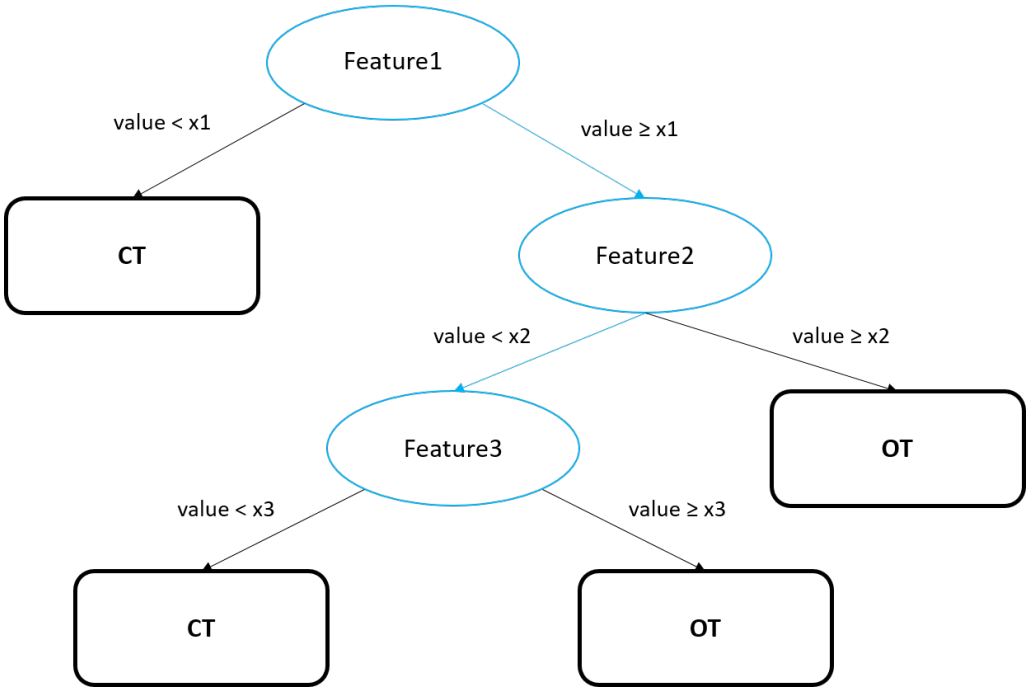



Figure 5.3: Example of decision tree.

Figure 5.3 shows an example of a decision tree built based on features of the contours. Each blue rectangle represents an intermediate node, containing the feature to be analysed. Moreover, each intermediate node is split with two arrows, based on feature values. Splitting can be performed by *binary split*, as made in Figure 5.3, or by *multi-way split*, dividing values in many partitions. The last nodes in the tree having no more children, shown in Figure 5.3 as black rectangles, are called *leaf nodes*. They indicate the class label to be assigned to a contour, having the features following that specific path. Hence, having an array of feature extracted from a contour, our process visits the tree in order to assign a label to that contour. For

example, given the tree of Figure 5.3, a contour will be classified with label "OT" if it has $Feature1 \geq x1$, $Feature2 < x2$ and $Feature3 \geq x3$.

The advantages of this method are:

- inexpensive to construct
- fast classification of record to be test
- good accuracy for simple data sets

Having a relative small data sets, we decided to choose this method to perform the classification.

### 5.5.2 Training phase

As described in Section 5.5.1, a classifier needs a collection of labelled data, defined as *training set*, in order to build the model. In our process, this set is composed by a subset of our images, each of them associated to a list of ground truth labelled contours, as described in Section 5.3. Moreover, labelled contours must be processed before training the classifier. Firstly, the features described in Section 5.4 must be extracted for each contour by $feature(contour)$ procedure. Secondly, once having all the features, they are *normalized*. This is an important step useful to facilitate classification. Indeed, it reduces the cardinality of all the attributes to a fixed range (e.g. $[0, 1]$), in order to avoid an attribute domain to be dominant than others. For example, referring to our features, contour intensity sum will have a variation scale higher than contour intensity average. This would cause contour intensity sum feature to be dominant than contour intensity average if the normalization has not been performed. Hence, in order to avoid that, our process computes the *z-score* function, described in Section 2.4, as normalization function applied to extracted features. Recalling it, its formula is:

$$z_i = \frac{x_i - \overline{x}}{S}$$

where $\overline{x}$ represents the mean function and $S$ the standard deviation. This normalization function allows having data with zero main and unitary variance.

Only after performing the two operations just described, the decision tree is fit with the training set in order to build our model.

### 5.5.3 Testing phase

In order to evaluate the classification model just created, an other subset of our images must be chosen, different from that one used for the training phase described in Section 5.5.2. It is called *test set* and it contains, such as for the training set, labelled contours associated to each image. For each contour of the image to be tested, our process performs the same steps used to process contours in the training set:

- extract features from all the contours by $feature(contour)$ procedure
- normalize them with z-score function, using the mean and the standard deviation computed for normalizing training data

Then, the processed test set is given as input to the classification model, that will return, as result, the predicted class labels for each contours inside the set. Finally, our process is able to classify the global image as "CT" if it contains at least one contour labelled as "CT" otherwise as "OT". This is done for all the images manually labelled and predicted by the decision-tree. The evaluation of the classification is performed by the ground truth labels with the predictions. A *confusion matrix* shows the distribution of the correct prediction across the different classes. It can be obtained as:

| | Predicted class | | |
|---|---|---|---|
| Actual class | | Class="CT" | Class="OT" |
| | Class="CT" | TP | FN |
| | Class="OT" | FP | TN |

Table 5.1: Confusion matrix

In Table 5.1, TP, FN, FP and TN respectively mean *true positive*, *false negative*, *false positive* and *true negative*. Starting from these values, the accuracy of a model is defined as:

$$Accuracy = \frac{\text{Number of correctly classified objects}}{\text{Number of classified objects}} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

Other two important measures for evaluating the classification model are *recall*, that is the percentage of correct predictions among objects with a specific actual class and *precision*, that identifies the percentage of correct samples among those predicted with a specific class. They are evaluated separately for each class (C) and they are defined as:

$$Precision(p) = \frac{\text{Number of objects correctly assigned to C}}{\text{Number of objects belonging to C}} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

$$Recall(r) = \frac{\text{Number of objects correctly assigned to C}}{\text{Number of objects assigned to C}} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

All these formulas are used as evaluation criteria for establishing how good the classification model is able to detect when a constant tone is present or not.

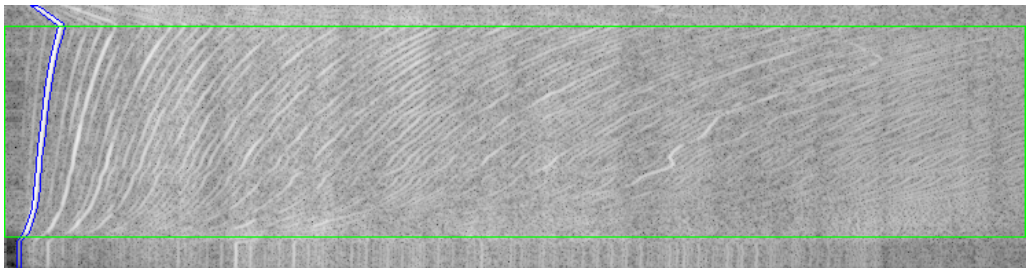
# Chapter 6

# Results

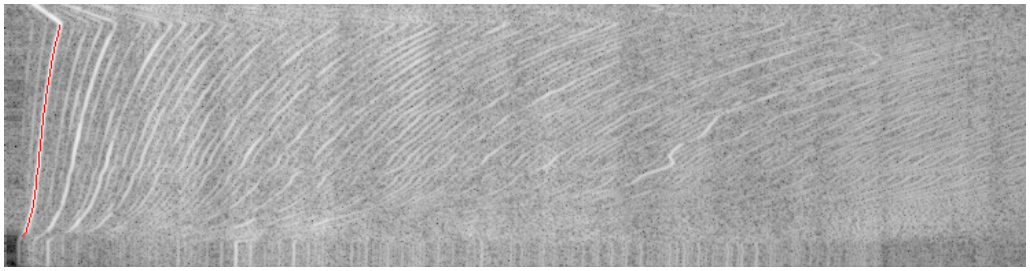
## 6.1 Evaluation of the main order identification

In Chapter 4, we described how, starting from a recorded file audio, our process automatically obtains a spectrogram in frequency (x-axis) and RPM (y-axis) domain through intermediate steps. One of the most difficult and important steps for obtaining a good result, is the *main order identification*. Its necessary to correctly recognize its shape, characterized by the most intense in magnitude pixels in the middle of the contour. Correctly including those pixels inside the found contour, will be also needed for performing future analysis on RPM domain by known formulas, especially to analyse a particular order n. Hence, we have to evaluate how many of that pixels are included in our main order identification. For doing this, we exploited the ground truth of the main order, found in Chapter 3. Then, the most intense pixels must be extracted from either the ground truth and the main order identified by our procedure. By extracting them, we isolate the contour from others, by applying the same mask described in Section 4.1.6, that will produce a matrix having only the pixel intensity of the contour isolated. Moreover, pixels constant in

frequency in the bottom part of the image and pixels in the upper part representing the deceleration phase are not considered for the evaluation and they must be discarded. The range $[y_{low}, y_{up}]$, where the most important part of the main order is present, can be taken by the *engine* ground truth, described in Section 3.2.1. Then, our procedure, analysing each matrix raw between position $[y_{low}, y_{up}]$, extracts the $x$-coordinate of the most intense pixel of that raw. Performing this, we obtained a list of coordinates $(x, y)$ representing the set of most intense horizontal pixel for a contour.



(a) *Ground truth*



(b) *Most intense pixels*

Figure 6.1: Example of most intense pixels inside the main order, drawn with red colour, extracted from the ground truth.

Figure 6.1 depicts an example of the most intense pixels inside the main order contour, drawn in red. The pixels are extracted from the inner part of the blue contour, taking into consideration only the pixels inside the green rectangle.

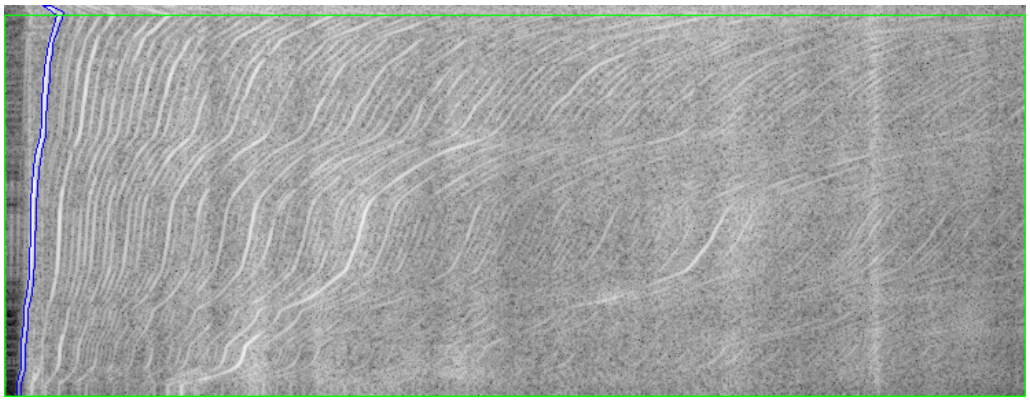
Extracting these pixels from both the ground truth contour and that one identified by our procedure as main order, to evaluate the accuracy of identification we performed the intersection over union (IoU), explained in Section 5.3, between the

two set of pixels. IoU evaluates the accuracy of an object detector, having as output the percentage of the accuracy (from 0 to 1). In the following, we present the result obtained with our model for the main order identification phase, according to this metric.

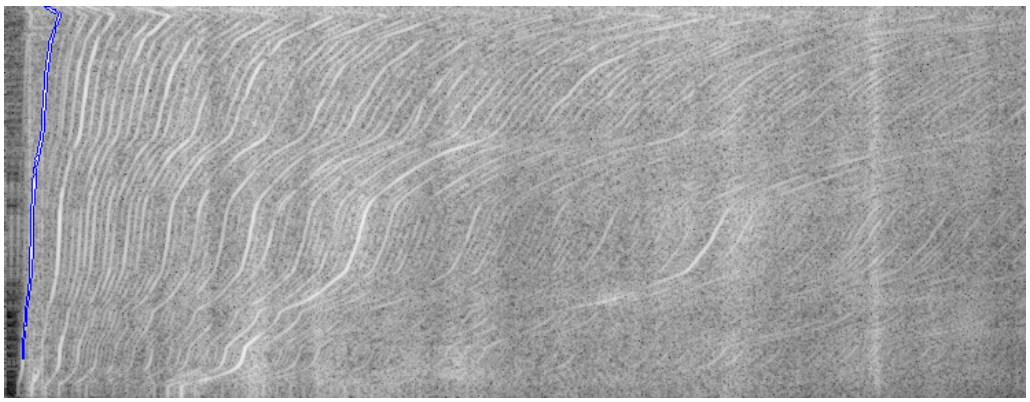
| | **IoU value** |
|---|---|
| **Min** | 0.76 |
| **Max** | 1.0 |
| **Avg** | 0.96 |

Table 6.1: IoU statistics

Table 6.1 shows IoU statistics. We obtained an average value of 0.96 pixels matched and minimum of 0.76, which means that in all images the main order is properly recognized, with small errors. Indeed, Figure 6.2, shows an example where the IoU presents a low value (i.e. 0.76). Despite being the minimum one, only a non significant part of low main order is not caught well. Histogram in Figure 6.3 shows the complete distribution of the IoU values for main order detection.



(a) *Ground truth*



(b) *Min IoU spectrogram*

Figure 6.2: Spectrogram with minimum IoU.

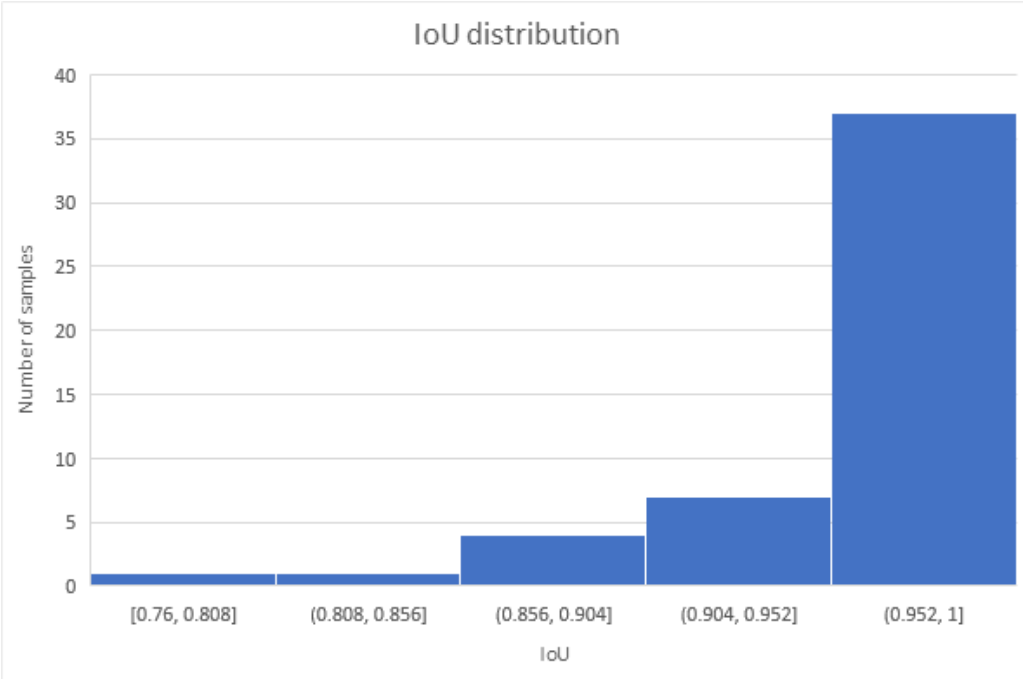



Figure 6.3: Distribution of the IoU values for the main order.

## 6.2 Evaluation of the constant tone detection

Constant tone detection, described in Chapter 5, performed by decision tree algorithm, must be evaluated to understand the accuracy of the classification model. In order to validate the model, training and test data must be split. The partitioning technique chosen is the *cross validation*. It consists in:

- partitioning data into k disjoint subsets
- training on k-1 partitions
- testing the remaining one

Having a relative small dataset, we adopted a particular case of cross validation, only appropriate for small dataset, called *leave-one-out*. It chooses the total number of data as k. Moreover, in our case, each image of the dataset will be tested, having

as training set the others. This guarantees a reliable accuracy estimation of the model.

Then, we analysed the decision tree technique, comparing it with a more complex model called *random forest classifier* [10]. It uses more than one decision tree classifiers, each one fitted by a random subset of training set. The final class is decided by combining the different results of each tree.

Moreover, we performed cross validation to the two methods by varying also the *max depth* parameter of the trees. It indicates the upper limit for the longest path (from the root to a leaf) reachable by a tree during its creation. Using the metrics described in Section 5.5.3, the results of our analysis can be summarized with the following tables.

Total number of images analysed = 50
Partitioning technique: leave-one-out

Table 6.2: Decision tree, max depth = 8

| | **Predicted class** | | |
|---|---|---|---|
| **Acual class** | | Class="CT" | Class="OT" |
| | Class = "CT" | 18 | 12 |
| | Class = "OT" | 5 | 15 |

Table 6.3: Random forest, max depth = 8

| | **Predicted class** | | |
|---|---|---|---|
| **Acual class** | | Class="CT" | Class="OT" |
| | Class = "CT" | 11 | 19 |
| | Class = "OT" | 2 | 18 |

Table 6.4: Decision tree, max depth = 10

| | **Predicted class** | | |
|---|---|---|---|
| **Acual class** | | Class="CT" | Class="OT" |
| | Class = "CT" | 17 | 13 |
| | Class = "OT" | 7 | 13 |

Table 6.5: Random forest, max depth = 10

| | **Predicted class** | | |
|---|---|---|---|
| **Acual class** | | Class="CT" | Class="OT" |
| | Class = "CT" | 13 | 17 |
| | Class = "OT" | 1 | 19 |

Table 6.6: Decision tree, max depth = 12

| | **Predicted class** | | |
|---|---|---|---|
| **Acual class** | | Class="CT" | Class="OT" |
| | Class = "CT" | 18 | 12 |
| | Class = "OT" | 5 | 12 |

Table 6.7: Random forest, max depth = 12

| | **Predicted class** | | |
|---|---|---|---|
| **Acual class** | | Class="CT" | Class="OT" |
| | Class = "CT" | 14 | 16 |
| | Class = "OT" | 1 | 19 |

Table 6.8: Decision tree metrics

| | | **Accuracy** | **Precision true** | **Recall true** | **Precision false** | **Recall false** |
|---|---|---|---|---|---|---|
| **Max depth** | 8 | 0.62 | 0.78 | 0.6 | 0.56 | 0.75 |
| | 10 | 0.56 | 0.71 | 0.57 | 0.5 | 0.65 |
| | 12 | 0.62 | 0.78 | 0.6 | 0.56 | 0.75 |

Table 6.9: Random forest metrics

| | | **Accuracy** | **Precision true** | **Recall true** | **Precision false** | **Recall false** |
|---|---|---|---|---|---|---|
| **Max depth** | 8 | 0.67 | 0.85 | 0.37 | 0.49 | 0.9 |
| | 10 | 0.73 | 0.93 | 0.43 | 0.53 | 0.95 |
| | 12 | 0.73 | 0.93 | 0.47 | 0.54 | 0.95 |

Looking at these tables, random forest allow obtaining higher accuracy, thanks to higher precision of the true class. Precision of the false class is not so good in both approaches, due to higher number of images having a label "CT" classified

as "OT". However, we are able to detect correctly most of the samples, obtaining accuracies till 73%.

# Chapter 7

# Conclusions

## 7.1 Summary

The aim of this thesis was to develop a procedure for the automatic detection of a specific noise signature in a spectrogram, obtained by a file audio recording a car engine. This is the last operation of a standard process, manually performed by domain experts. In the first part, we have presented this process and the main challenges to deal with. The existing process can be split in two main parts. The first one is related to the conversion, through intermediate steps, from the audio to a spectrogram in the frequency and RPM domain. The default domain in frequency and time, obtainable just performing Fourier transforms on recorded file audio, is not enough. Indeed, after an engine fault is detected by a domain expert, already developed formula can be applied to spectrogram in RPM domain, to understand the gravity of the problem. Hence, even if an engine fault can be detected in time domain, for evaluating its intensity, RPM domain is necessary. Then, in the second part of the thesis, we dealt with the conversion from the audio to a spectrogram in RPM domain. However, before proceeding with the task, a dataset must be created. By the standard procedure, we collected 50 audio. To deal with the conversion

from the collected audio to spectrograms in time domain we exploited the Fourier transforms. Moreover, for what concern the time to RPM conversion, we handled image processing techniques for emulating what the operators done manually thanks to industrial tools. The resulting images have an appreciable quality, indicating the achievement of an automatic conversion.

The second main part of the original process is the constant tone identification. In the last chapters of the thesis, we presented our approach that consists in creating patterns that try to automatically recognize the typical features of the constant tone and detect its presence or absence. As a first approach, we performed this task on the spectrogram in time domain by image processing and machine learning techniques. The results provide a good quality of the detection. Shifting the detection on RPM domain, we think to obtain also better result than time domain.

## 7.2 Future works

The quality of spectrogram RPM images are good, but industrial tools still present a higher quality. A future work could be investigating a signal processing technique, called zoom FFT, typically used to analyse a portion of a spectrum with higher quality. Since we are interested only in the range $[f_{min}, f_{max}]$Hz, it could help to improve the spectrogram quality in time domain and, as a consequence, obtain a better quality in the conversion to RPM. An other improvement that would allow having a better conversion from time to RPM, would be using the most intense pixels inside the main order shape as the guideline for the conversion. Since we used the right contour of the main order shape, it could be more affected by noise. An other significant improvement will be collecting more data, that would expand knowledges about the problems and will let us apply more complex machine learning techniques based on larger amounts of data, such as *neural networks*. Once improving the spectrogram in RPM, the detection can be performed directly on RPM domain in

order evaluate also the intensity of the detected fault. Having also the evaluation of gravity of the problem, the process could be shifted to a mobile environment. This could be done by a client-server approach, where mobile environment acts as the client, performing just the acquiring of the data, the conversion to the spectrogram and send it to the server. Then, the server could analyse the spectrogram and answer to the client indicating the percentage of an engine fault to be present or not. This could also allow collecting a huge amount of data, useful for improving the whole process.

# Bibliography


[1] Atkinson, D., *The Sound Production Handbook.* Taylor & Francis, p 23, 2013.

[2] Oppenheim, Alan V., Ronald W. Schafer, John R. Buck, *Discrete-Time Signal Processing*, Prentice Hall, 1999.

[3] B.C. Russell, A. Torralba, K.P. Murphy, and W.T. Freeman., *Labelme: A Database and Web-Based Tool for Image Annotation.*, International journal of computer vision, 77(1):157173, 2008.

[4] Sylvain Paris, Pierre Kornprobst, Jack Tumblin and Frdo Durand, *Bilateral Filtering: Theory and Applications*, Foundations and Trends in Computer Graphics and Vision: Vol. 4: No. 1, pp 1-73, 2009.

[5] D. Comaniciu and P. Meer, *Mean shift: A robust approach toward feature space analysis*, IEEE Trans. Pattern Anal. Machine Intell., vol. 24, no. 5, pp. 603619, 2002.

[6] Comaniciu, D. and Meer, P. *Mean shift Analysis and Applications.* In Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, pp. 11971203, 1999.

[7] P. N. Tan, M. Steinbach and V. Kumar, *Introduction to data mining.* Boston, Pearson Addison-Wesley 2006.

[8] Suzuki, S. and Abe, K., *Topological Structural Analysis of Digitized Binary Images by Border Following.* CVGIP 30 1, pp 32-46, 1985.

[9] David H. Douglas and Thomas K. Peucker, *Algorithms for the reduction of the*

*number of points required to rep- resent a digitized line or its caricature.* The Canadian Car- tographer, 10(2):112122, 1973.

[10] Witten, I. H., Frank, E., Hall, M. A., and Pal, C. J. *Data Mining: Practical machine learning tools and techniques.* Morgan Kaufmann, 2016.