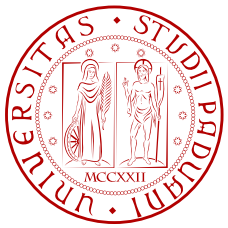

UNIVERSITY OF PADOVA

DEPARTMENT OF INFORMATION ENGINEERING

GMM CLASSIFICATION OF ENVIRONMENTAL SOUNDS FOR SURVEILLANCE APPLICATIONS

Supervisors:
Federico Avanzini
Emanuele Menegatti

Author:
Riccardo Levorato

Master's Degree in Computer Engineering
(Laurea Magistrale in Ingegneria Informatica)

Graduation Date: Padova, October 26th, 2010

Academic Year 2009/2010

Preface

The work reported in this thesis is part of a surveillance system's development that integrates different types of sensors and provides mainly the integration of audio with video. Integrating audio monitoring in surveillance applications is indeed advantageous in many respects: it is "cheap" in terms of both hardware requirements and computational burdens; it complements the video monitoring especially for areas not visible by cameras; it can be of fundamental importance in early detection of awkward situations (burglary, theft, etc.). The algorithms and code developed in this thesis work, focused only on the audio analysis, enable, with the support of a microphone, to detect any type of impulsive sound and to classify with good accuracy four types of impulsive sounds in every surrounding environment: screams, gunshots, broken glasses and barking dogs. Classification is achieved using Gaussian Mixture Models (GMM), trained on several audio features extracted from the audio. The whole system is developed in the Network-Integrated Multimedia Middleware (NMM), which simplifies communications between different types of agents that are organized into interconnected nodes that simultaneously process the data coming from the sensors.

The main innovative approach of this work is the willingness to integrate the experience in the field of telecommunications with intelligent surveillance systems that merge environmental sensors such as cameras or microphones in mobile robotic platforms that can intervene and provide on-site a completely automated, independent and more complete system than a network of fixed sensors. The creation of an integrated system with fixed audio and video sensors will allow mobile robots to propose surveillance systems with more than a mobile unit able to go to a site in order to avoid false alarms.

Phases of the thesis

The thesis started from a project developed for the course *"Informatica musicale"*, titled *"Techniques of identification and automatic classification of environmental sounds for surveillance applications"*, and carried out in collaboration with the master student Carlo Alberto Cazzuffi of the university of Padova. This project, based widely on [1][2], was entirely implemented in Matlab and concerned only the off-line classification of screams and gunshots in a noisy environment. These two papers[1] describe a technique to develop the software, and suggest the most relevant features to be extracted from the audio signals, in order to classify the sounds using

[1]The two papers are similar with small variants on the choice of the parameters. In fact they were published in subsequent moments due to an improvement obtained by the same authors.

GMMs. In our implementation we refer mostly to [2] because it was the last update regarding this work.

The first phase of the project dealt with an implementation of the sound classification using the Matlab language, for the following reasons:

- simple and fast code programming;
- simple audio signals access from file .wav (functions *wavread* and *wavwrite*);
- GMM's training algorithm already implemented in the Statistics Toolbox;
- availability of a Matlab tool that extracts 30 Mel-Frequency Cepstral Coefficients (MFCCs) and other features from an audio signal, developed by Enrico Marchetto, PhD student of the university of Padova.

In a second phase, when I decided to extend the work to my thesis, the code had to be implemented in C++, in order to integrate it over the Network-Integrated Multimedia Middleware (NMM). For this reason, a long period of work has been devoted to porting all the code from Matlab to C++, except for the GMM's training algorithm and the audio test maker. The reason why I choose not to port the GMM's training algorithm is because the classification of the sounds (and of the environmental noise) can be made off-line, as well as the creation of the audio tests, and is compatible with a hypothetical product that does not require on-board training. The porting period lead fortunately to a strong optimization of the code, correction of some bugs/errors and a faster execution of the algorithm. In this period I introduced also two new classes of sounds with respect to the initial project: broken glasses and barking dogs.

In the third period the attention switched to the porting of the software over the NMM architecture following the suggestion of Prof. Emanuele Menegatti, who supervises a large project on surveillance with many analytical nodes (audio and video). In collaboration with Alberto Salamone, a master student of the University of Padova who realized a fall detector with an omnidirectional camera and a steerable PTZ camera, I installed NMM and realized some processing nodes that can analyse and classify the recorded sound in real-time.

In the fourth and last period I assessed the performance of the audio classification in several aspects, both on the off-line C++ implementation and on the real-time NMM realization.

Structure of the thesis

This thesis is structured in six chapters with an additional appendix. Chapter 1 reviews the state of the art on this topic. In Chapter 2, all the mathematical tools used to realize and implement the classification system are listed and explained. Chapter 3 contains the complete description of the original work with project choices, and implementations. Chapter 4 is structured in three parts. The first part is a description of all the development tools used in the project. The second part is written as a "HowTo"guide to simplify the installation and configuration of the environment to work easily with the used middelware (NMM). The third part describes the implementation of the project in the middleware. All the experimental results are explained in Chapter 5, including

both off-line and real-time tests. The final Chapter contains the conclusions of the work, as well as further developments and hints suggested to improve the system. The appendix lists and describes all the libraries needed to install NMM.

Prefazione

Il lavoro riportato in questa tesi fa parte dello sviluppo di un sistema di sorveglianza che integra diversi tipi di sensori e fornisce principalmente l'integrazione dell'audio con il video. Integrare il monitoraggio audio in applicazioni di sorveglianza è molto vantaggioso per molti aspetti: è "economico" sia in termini di requisiti hardware che di onerosità di calcolo; integra il monitoraggio video in particolare per le zone non visibili da telecamere; può essere di fondamentale importanza nella rilevazione preventiva di situazioni scomode (furti, scassi, ecc.). Gli algoritmi e il codice sviluppato in questo lavoro di tesi, concentrato solo sull'analisi dell'audio, consentono, con il supporto di un microfono, di individuare qualsiasi tipo di suono impulsivo e di classificare con esattezza ben quattro tipi di suoni impulsivi in ogni tipo di ambiente circostante: urla, spari, vetri rotti e cani che abbaiano. La classificazione è realizzata usando i Gaussian Mixture Models (GMM), addestrati su diverse feature audio estratte dall'audio. L'intero sistema è sviluppato nel Network-Integrated Multimedia Middleware (NMM), che semplifica le comunicazioni tra i diversi tipi di agenti che sono organizzati in nodi interconnessi i quali elaborano contemporaneamente i dati provenienti dai sensori.

Il principale approccio innovativo di questo lavoro è la volontà di integrare l'esperienza nel campo delle telecomunicazioni con sistemi di sorveglianza intelligenti che fondono sensori ambientali come telecamere o microfoni in piattaforme robotiche mobili che possano intervenire e fornire sul sito un sistema completamente automatizzato, indipendente e più completo di una rete di sensori fissi. La creazione di un sistema integrato con sensori audio e video fissi permetterà a robot mobili di proporre sistemi di sorveglianza con più di una unità mobile in grado di andare in un sito per evitare falsi allarmi.

Fasi della tesi

La tesi ha avuto inizio da un progetto sviluppato per il corso *"Informatica Musicale"*, dal titolo *"tecniche di identificazione e classificazione automatica di suoni ambientali per applicazioni di sorveglianza"*, e realizzata in collaborazione con Carlo Alberto Cazzuffi, studente magistrale dell'università di Padova. Il progetto, basato in gran parte su [1][2], è stato interamente implementato in Matlab e ha riguardato solo la classificazione off-line di urla e spari in un ambiente rumoroso. Questi due documenti[2] descrivono una tecnica per sviluppare il software, e suggeriscono le features più rilevanti da estrarre dai segnali audio, al fine di classificare i suoni me-

[2]I due articoli sono simili ma con piccole varianti sulla scelta dei parametri. In realtà sono stati pubblicati in momenti successivi per un miglioramento ottenuto dagli stessi autori.

diante GMM. Nella nostra implementazione ci riferiamo soprattutto a [2] perché è stato l'ultimo aggiornamento riguardante questo lavoro.

La prima fase del progetto ha riguardato l'implementazione della classificazione del suono utilizzando il linguaggio Matlab per le seguenti ragioni:

- linguaggio di programmazione semplice e veloce;
- semplice accesso ai segnali audio da file .wav (con funzioni *wavread* e *wavwrite*);
- algoritmo di training delle GMM già implementato nello Statistics Toolbox;
- disponibilità di un tool in Matlab che estrae 30 Mel-Frequency Cepstral Coefficients (MFCCs) e altre feature di un segnale audio, sviluppato da Enrico Marchetto, dottorando dell'Università di Padova.

In una seconda fase, quando ho deciso di estendere il lavoro alla mia tesi, si è verificato il bisogno di implementare il codice in C++, al fine di integrarlo nel Network-Integrated Multimedia Middleware (NMM). Per questo motivo, ho dedicato un lungo periodo di lavoro al porting di tutto il codice da Matlab a C++, fatta eccezione per l'algoritmo di training dei GMM e il programma per la creazione di test audio. Il motivo per cui ho scelto di non fare il porting dell'algoritmo di training dei GMM è perché la classificazione dei suoni (e del rumore ambientale) può essere effettuata off-line, così come per la creazione dei test audio, ed è compatibile con un ipotetico prodotto che non richiede il training on-board. Il periodo di porting ha portato fortunatamente ad una forte ottimizzazione del codice, alla correzione di alcuni bug/errori e ad una più veloce esecuzione dell'algoritmo. In questo periodo ho introdotto anche due nuove classi di suoni rispetto al progetto iniziale: vetri rotti e cani che abbaiano.

Nel terzo periodo mi sono occupato del porting del software sull'architettura NMM seguendo il suggerimento del Prof. Emanuele Menegatti, che supervisiona un grande progetto sulla sorveglianza con molti nodi analitici (audio e video). In collaborazione con Alberto Salamone, studente magistrale dell'Università di Padova che ha realizzato un rivelatore di caduta con una telecamera omnidirezionale e una telecamera PTZ orientabile, ho installato NMM e realizzato alcuni nodi di elaborazione in grado di analizzare e classificare il suono registrato in tempo reale.

Nel quarto e ultimo periodo ho testato le prestazioni della classificazione audio in parecchi aspetti, sia nell'implementazione off-line in C++ sia nella realizzazione in tempo reale in NMM.

Struttura della tesi

Questa tesi è strutturata in sei capitoli con un'appendice aggiuntiva. Il Capitolo 1 è la recensione dello stato dell'arte dell'argomento della tesi. Nel Capitolo 2 sono elencati e spiegati tutti gli strumenti matematici utilizzati per realizzare e implementare il sistema di classificazione. Il Capitolo 3 contiene la descrizione completa del lavoro originale con scelte progettuali e implementazioni. Il Capitolo 4 è strutturato in tre parti. La prima parte è una descrizione di tutti gli strumenti di sviluppo utilizzati nel progetto. La seconda parte è scritta come una guida "HowTo" per semplificare l'installazione e la configurazione dell'ambiente per poter lavorare facilmente

con il middelware utilizzato (NMM). La terza parte descrive l'implementazione del progetto nel middleware. Tutti i risultati sperimentali sono spiegati nel Capitolo 5, che include sia i test off-line che quelli in tempo reale. Il Capitolo finale contiene le conclusioni del lavoro, nonché gli ulteriori sviluppi e spunti suggeriti per migliorare il sistema. L'appendice elenca e descrive tutte le librerie necessarie per installare NMM.

Abstract

This thesis describes an audio event detection system which automatically classifies an impulsive audio event as scream, gunshot, broken glasses or barking dogs with every background noise. The classification system uses four parallel Gaussian Mixture Models (GMMs) classifiers each of which decides if the sound belongs to its class or is only noise. Each classifier is trained using different features, chosen from a set of 40 audio features. Simultaneously the system can detect any kind of impulsive sounds using only one feature with very high precision. The classification system is implemented in the Network-Integrated Multimedia Middleware (NMM) for real-time processing and communications with other surveillance applications. In order to validate the proposed detection algorithm, we carried out extensive experiments (both off-line and real-time) on a hand-made set of sounds mixed with ambient noise at different Signal-to-Noise ratios (SNRs). Our results demonstrate that the system is able to guarantee 70% of accuracy and 90% of precision at 0 dB SNR, starting from 100% of both accuracy and precision with clean sounds at 20 dB SNR.

Sommario

Questa tesi descrive un sistema di rilevazione di eventi audio che classifica automaticamente un rumore impulsivo come urla, spari, vetri rotti o cani che abbaiano con qualsiasi rumore di sottofondo. Il sistema di classificazione utilizza quattro classificatori in parallelo, costruiti con i Gaussian Mixture Models (GMMs), ciascuno dei quali decide se il suono appartiene alla propria classe o se è soltanto rumore. Ogni classificatore è addestrato con differenti feature, scelte da un insieme di 40 feature audio. Contemporaneamente il sistema può rilevare qualsiasi tipo di suoni impulsivi utilizzando una sola feature con una precisione molto elevata. Il sistema di classificazione è implementato nel Network-Integrated Multimedia Middleware (NMM) per l'elaborazione in tempo reale e le comunicazioni con altre applicazioni di sorveglianza. Al fine di validare l'algoritmo di rilevazione proposto, sono stati effettuati vari esperimenti (sia off-line sia in tempo reale) su un personale database di suoni, mescolati con rumore ambientale, a diversi rapporti di segnale-rumore (SNR). I nostri risultati dimostrano che il sistema è in grado di garantire il 70% di accuratezza e il 90% di precisione a 0 dB di SNR, a partire da 100% di accuratezza e precisione con suoni puliti a 20 dB di SNR.

Acknowledgements

First I want to thank my parents with a special hug, hoping that this goal would fill them with satisfaction. I would like to thank my professors Federico Avanzini and Emanuele Menegatti to help me and direct my work in the right way. Special thanks go to Carlo Alberto Cazzuffi and Alberto Salamone that worked with me in this project and to Federica Ziliotto that helped me to correct the English version of this thesis. Thanks to Isacco Saccoman, author of the fabulous comics in my final slide of the presentation. Acknowledgements go also to everyone that helped and supported me directly or indirectly in my studies.

I sincerely apologize with Classical Music to have neglected it in order to finish my university studies.

Ringraziamenti

Prima di tutto vorrei ringraziare i miei genitori con un abbraccio speciale, sperando che questo traguardo li riempia di soddisfazione. Vorrei ringraziare i miei professori Federico Avanzini e Emanuele Menegatti per avermi aiutato e aver indirizzato il mio lavoro verso la direzione giusta. Ringraziamenti particolari vanno a Carlo Alberto Cazzuffi e Alberto Salamone che hanno lavorato con me in questo progetto e a Federica Ziliotto che mi ha aiutato a correggere la versione in inglese di questa tesi. Grazie a Isacco Saccoman, autore del favoloso fumetto nella slide finale della mia presentazione. Ringraziamenti particolari vanno anche a tutti coloro che mi hanno aiutato e sostenuto direttamente o indirettamente durante i miei studi.

Mi scuso profondamente con la Musica Classica per averla trascurata per terminare i miei studi universitari.

Contents

List of Figures

List of Tables

Chapter 1

State of the Art

In addition to the traditional video cameras, the use of audio sensors in surveillance and monitoring applications is becoming increasingly important [3]. Audio is useful especially in situations when other sensors, such as video, fails to detect correctly the events. For example, when objects are occluded or in the dark, audio sensors can be more appropriate in detecting a "noisy" presence. Conceptually, there are many events which can be detected better using audio rather than other sensors, e.g. screams or gunshots [4][5], broken glasses, barking dogs, etc. Furthermore, audio sensors are a quite accessible resource for the costs.

Such detection systems can be efficiently used to advert an automated system that an event has occurred with high probability and, at the same time, to enable further processing like automatic video-camera steering. Traditional tasks in the area of the automatic audio classification and matching are speech/music segmentation, classification and audio retrieval. Much of the previous work about audio-based surveillance systems concentrated on the task of detecting some particular audio events. More recently, were developed specific works covering the detection of particular classes of events for multimedia-based surveillance. For example, detection systems specifically designed for impulsive sound[1] recognition consist of a segmentation step, where is detected the presence of an event, followed by a classification step, which refines the result assigning a class label to the event. The results reported in [6], show that these systems fail under real-world conditions reaching less than 50% accuracy at 0 dB SNR (*Signal-to-Noise Ratio*).

Mel-Frequency Cepstral Coefficients (MFCCs) are the most common features representation for non-speech audio recognition. Peltonen et al. in [7] implemented a system for recognizing 17 sound events using 11 features individually and obtained best results with the MFCCs. In a comparison of several feature sets, MFCCs perform well [8], although the classification granularity affects the relative importance of different feature sets. Cai et al. in [9] used a combination of statistical features and labels describing the energy envelope, harmonicity, and pitch contour for each sample. None of these representations, unfortunately, shows clear performance or conceptual advantages over MFCCs.

In the SOLAR system presented in [10], the segmentation step is avoided by decomposing

[1] A definition of *impulsive sound* is a sound with a rapid rise and decay of sound pressure level, lasting less than one second. It is caused by sudden contact between two or more surfaces or by a sudden release of pressure. In other words it is a sound that makes you turn your head and open your ears putting your body in an "alarm" state.

audio tracks into short overlapping audio windows. For each window, a set of 138 features is extracted and evaluated by a series of boosted decision trees. Though efficient in real time computations, the SOLAR system suffers from large differences in classification accuracy from class to class. More recent works showed that a hierarchical classification scheme, composed by different levels of binary classifiers, generally achieves higher performance than a single-level multi-class classifier. In [11] a hierarchical set of cascaded GMMs (*Gaussian Mixture Models*) is used to classify 5 different sound classes. Each GMM is tuned using only one feature from a feature set including both scalar features (e.g. *Zero-Crossing Rate* (ZCR)) or vector features (e.g. *Linear-Log Frequency Cepstral Coefficients* (LLFCC)). Reported results show that the hierarchical approach yields accuracies from 70 to 80% for each class, while single level approaches reach high accuracies for one class but poor results for the others.

The hierarchical approach has also been employed in [4] to design a specific system able to detect screams/shouts in public transport environments. After a preliminary segmentation step, a set of perceptual features such as MFCCs or *Perceptual Linear Prediction* (PLP) coefficients are extracted from audio segments and used to perform a 3-levels classification. First, the audio segment is classified either as noise or non-noise; second, if it is not noise, the segment is classified either as speech or not speech; finally, if speech, it is classified as a shout or not. The authors tested this system using both GMMs and *Support Vector Machines* (SVMs) as classifiers, showing that generally GMMs provide higher precision.

A different technique is used in [5] to detect gunshots in public environments. In this work, the performance of a binary gunshot/noise GMM classifier is compared to a classification scheme in which several binary sub classifiers for different types of firearms run in parallel. A final binary decision (gunshot/noise) is taken evaluating the logical OR of the results of each classifier. In this way, the false rejection rate of the system is reduced by a 50% on average with respect to the original binary classifier.

In a recent work [1][2], Valenzise et al. proposed a system that is able to detect accurately two types of audio events: screams and gunshots. They extract a very large set of features, including some descriptors like spectral slope and periodicity, and innovative features like correlation roll-off and decrease. To the authors knowledge, these features have never been used for the task of sound-based surveillance and it is shown that they provide a significant performance gain using a GMM as classifier.

The current project is a deep study and elaboration of the works [1][2] and can be seen as an implementation and improvement of them. This work is different from the previous ones in the following aspects. First, to manage a multimedia surveillance system and synchronize all processes of sensors and agents, is used NMM (*Network-Integrated Multimedia Middleware*). Second, the system can detect any kind of impulsive sound, with only one feature, using only a single microphone data with very high precision. Finally, over the screams and gunshots impulsive sound classes, the system can detect and classify even broken glasses and barking dogs.

Chapter 2

Principles of audio classification

In this chapter we describe the main mathematical tools used in the project to introduce the reader to the main topics of the thesis.

2.1 Audio Features

A considerable number of audio features was used in the project. Traditionally, these features are classified in:

- **Temporal features** - e.g. *Teager Energy Operator* (TEO) or *Zero-Crossing Rate* (ZCR);
- **Perceptual features** - e.g. loudness, sharpness or *Mel-Frequency Cepstral Coefficients* (MFCCs);
- **Energy features** - e.g. *Short Time Energy* (STE);
- **Spectral features** - e.g. spectral flatness, spectral skewness;

In this work are discarded the audio features which are too sensitive to the SNR conditions, like STE and loudness. In addition to the traditional features listed above, are employed some other features which have been introduced in [1][2], such as spectral distribution (spectral slope, spectral decrease), periodicity descriptors and new features based on the auto-correlation function: correlation decrease and correlation slope. The number of features extracted is 41 and are the following:

- *Teager Energy Operator (TEO)*;
- *Zero Crossing Rate (ZCR)*;
- *30 Mel-Frequency Cepstral Coefficients (MFCCs)*;
- *Spectral Flatness Measure (SFM)*;

- *Spectral Centroid*;
- *Spectral Skewness*;
- *Spectral Slope*;
- *Spectral Decrease*;
- *Whole-Band Periodicity*;
- *Filtered-Band Periodicity*;
- *Correlation Slope*;
- *Correlation Decrease*.

Now it follows a detailed definition of all the features used in the project. Note that an audio frame is a small audio segment of the whole audio signal. In other words, a frame is a fixed-length array containing a part of the values of the audio signal. TEO will be used to detect a generic impulsive sound and all other features will be used for the classification of the impulsive sounds.

2.1.1 Temporal features

2.1.1.1 Teager Energy Operator (TEO)

The *Teager Energy Operator* (TEO) [12] is a powerful non-linear operator that takes trace of the modulation energy and identifies the amplitude and the instantaneous frequency. Compared to the traditional suppression of the noise based on the frequency domain, TEO's noise suppression is based on the time domain. It was demonstrated experimentally that TEO can enhance the discrimination between impulsive sounds and background noise because "it can flatten" the pseudo-constant noisy component of the sound. Furthermore, it will be attenuated the increasing volume of the background signal too (see first part of the signal before the first impulsive sound in figure 2.1).

In continuous time TEO is defined as:

$$\Psi_c[s(t)] = [\dot{s}(t)]^2 - s(t) \cdot \ddot{s}(t) \tag{2.1}$$

where $s(t)$ is a continuous time signal and $\dot{s} = \frac{ds}{dt}$.

In discrete time the operator can be simplified as:

$$\Psi_d[s(n)] = s(n)^2 - s(n-1) \cdot s(n+1) \tag{2.2}$$

where $s(n)$ is a discrete time signal.

Consider now a signal $s(n)$ containing a scream corrupted by uncorrelated additive noise $u(n)$. The resulting signal $y(n)$ is:

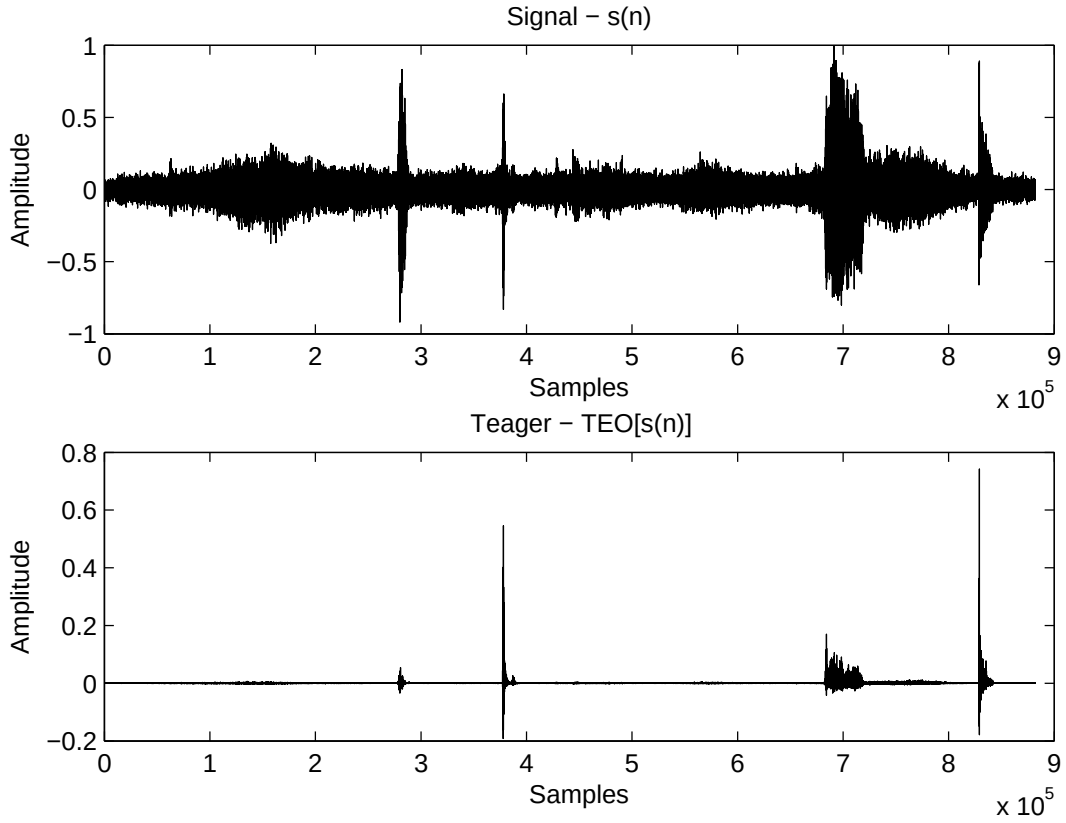

Figure 2.1: *Signal with four impulsive sounds and his TEO.*

$$y(n) = s(n) + u(n) \tag{2.3}$$

Let $\Psi_s[y(n)]$ the TEO of the signal $y(n)$. It is defined as:

$$\Psi_d[y(n)] = \Psi_d[s(n)] + \Psi_d[u(n)] + 2 \cdot \tilde{\Psi}_d[s(n), u(n)] \tag{2.4}$$

where $\Psi_d[s(n)]$ and $\Psi_d[u(n)]$ are the TEO of the scream and of the additive noise respectively. Let $\tilde{\Psi}_d[s(n), u(n)]$ the mutual energy Ψ_d between $s(n)$ and $u(n)$ such that:

$$\tilde{\Psi}_d[s(n), u(n)] = s(n) \cdot u(n) - 0.5 \cdot s(n-1) \cdot u(n+1) + 0.5 \cdot s(n+1) \circ u(n-1) \tag{2.5}$$

where $\circ$ represents the inner product.

As the signals $s(n)$ and $u(n)$ have zero mean and are uncorrelated, the expected value of the mutual energy $\tilde{\Psi}_d[s(n), u(n)]$, is zero. So it is possible to obtain the following equation:

$$E\{\Psi_d[y(n)]\} = E\{\Psi_d[s(n)]\} + E\{\Psi_d[u(n)]\} \tag{2.6}$$

In fact, TEO of the scream is significantly higher than TEO of the noise. So, compared to $E\{\Psi_d[y(n)]\}$, the expected value $E\{\Psi_d[u(n)]\}$ is irrelevant. Finally we obtain the relation:

$$E\{\Psi_d[y(n)]\} \approx E\{\Psi_d[s(n)]\} \tag{2.7}$$

2.1.1.2 Zero-Crossing Rate (ZCR)

ZCR is the rate of sign-changes along a signal [13]. In other words it is a measure of times' number the signal value crosses the zero axis rated by the number of values of the signal. Periodic sounds tend to have small ZCR, while noisy sounds tend to have high ZCR. The formula is:

$$ZCR = \frac{1}{N-1} \cdot \sum_{n=1}^{N-1} \pi(s(n) \cdot s(n-1) < 0) \tag{2.8}$$

where s is a signal of length N and the function $\pi(x)$ is 1 if its argument x is True and 0 otherwise.

2.1.2 Perceptual features

2.1.2.1 Mel-Frequency Cepstral Coefficients (MFCCs)

The mel scale, proposed by Stevens, Volkman and Newman in 1937 [14], is a perceptual scale of pitches judged by listeners to be equal in distance from one another. The reference point between this scale and normal frequency measurement is defined by equating a 1000 Hz tone, 40 dB above the listener's threshold, with a pitch of 1000 mels. Above about 500 Hz, larger and larger intervals are judged by listeners to produce equal pitch increments. As a result, four octaves on the hertz scale above 500 Hz are judged to comprise about two octaves on the mel scale (see Figure 2.2). The name mel comes from the word *melody* to indicate that the scale is based on pitch comparisons.

A popular formula to convert f hertz into m mel is[1]:

$$m = 2595 \log_{10}\left(\frac{f}{700} + 1\right) = 1127 \log_e\left(\frac{f}{700} + 1\right) \tag{2.9}$$

And the inverse:

$$f = 700(10^{m/2595} - 1) = 700(e^{m/1127} - 1) \tag{2.10}$$

The mel-frequency cepstrum (MFC) is a representation of the short-term power spectrum of a sound, based on a linear cosine transform of a log power spectrum on a non-linear mel scale of frequency.

[1]The base-10 formula with 2595 is from O'Shaughnessy (1987) [15]. The natural-log formula with coefficient 1127 is widely used more recently. Older publications typically use the break frequency of 1000 Hz rather than 700 Hz.

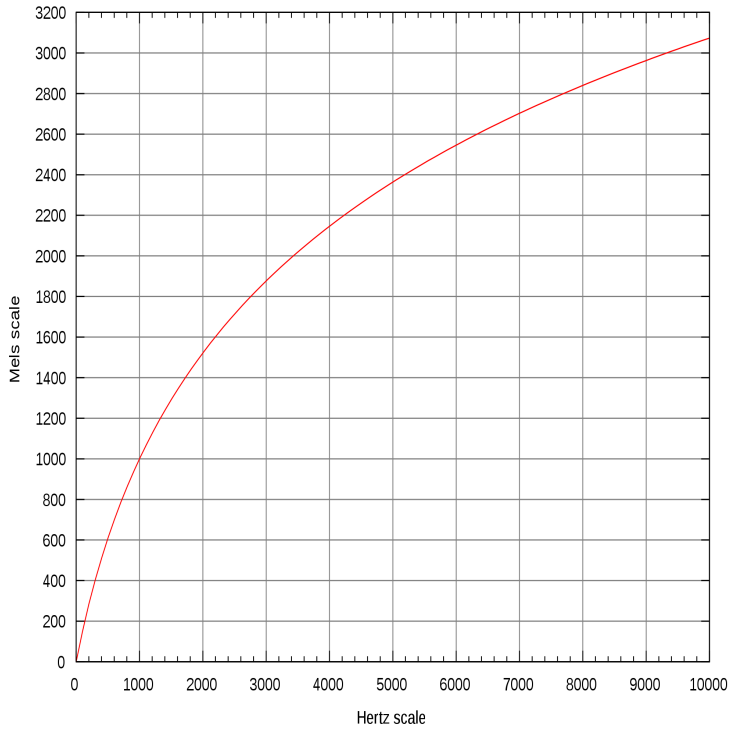

Figure 2.2: *Plots of pitch mels versus hertz.*

MFCCs collectively make up an MFC [13]. They are derived from a type of cepstral representation of the audio clip (a non-linear "spectrum-of-a-spectrum"). In the MFC the frequency bands are equally spaced on the mel scale, which approximates the human auditory system's response more closely than the linearly-spaced frequency bands used in the normal cepstrum. In fact MFCC reflects the energy distribution over the basilar membrane. Due to their perceptually motivated nature, MFCCs are considered to carry a high amount of relevant information related to a sound signal. In fact they are often used to characterize a sound signal in such applications as automatic speech/speaker recognition, and are increasingly used in music information retrieval applications too [16].

The MFCCs are derived as follows:

1. take the Fourier transform of a frame of a signal;
2. map the powers of the spectrum obtained above onto the mel scale, using triangular overlapping windows;
3. take the log of the powers at each of the mel frequencies;
4. take the discrete cosine transform of the list of mel log powers, as if it were a signal;
5. the MFCCs are the amplitudes of the resulting spectrum.

In the present project we extract 30 MFCCs as in [1][2].

2.1.3 Spectral features

2.1.3.1 Spectral Flatness Measure (SFM)

SFM is a measure used in digital signal processing to characterize an audio spectrum [13]. High spectral flatness indicates that the spectrum has a similar amount of power in all spectral bands and this would sounds similar to white noise. Low spectral flatness, instead, indicates that the spectral power is concentrated in a relatively small number of bands and it is typical for tonal sounds.

The spectral flatness is calculated by dividing the geometric mean of the power spectrum by the arithmetic mean of the power spectrum. The spectral flatness used is measured across the whole band.

$$SFM = \frac{\sqrt[N]{\prod_{n=0}^{N-1} x(n)}}{\frac{\sum_{n=0}^{N-1} x(n)}{N}} \tag{2.11}$$

where $x(n)$ represents the magnitude of bin number n of the power spectrum.

2.1.3.2 Spectral Centroid

Spectral centroid is a measure that indicates where is the "center of mass" of the spectrum [13]. Perceptually, it has a robust connection with the impression of "brightness" of a sound. It is calculated as the weighted mean of the frequencies present in the frame, determined using a Fourier transform, with their magnitudes as weights:

$$Centroid = \frac{\sum_{n=0}^{N-1} f(n) \cdot x(n)}{\sum_{n=0}^{N-1} x(n)} \tag{2.12}$$

where $x(n)$ represents the weighted frequency value, or magnitude, of bin number n, and $f(n)$ represents the center frequency of that bin.

2.1.3.3 Spectral Skewness

Spectral skewness is a measure of the asymmetry of the probability distribution of a real-valued random variable that in this context is the spectrum of the signal [13]. The skewness value can be positive or negative, or even zero. Qualitatively, a negative skew indicates that the tail on the left side of probability density function is longer than the right side and the bulk of the values (including the median) lies to the right of the mean. A positive skew indicates that the tail on the right side is longer than the left side and the bulk of the values lies to the left of the mean. A zero value indicates that the values are relatively evenly distributed on both sides of the mean, typically but not necessarily implying a symmetric distribution.

For a sample of N values forming a frame, the skewness is:

$$Skewness = \frac{m_3}{m_2^{3/2}} = \frac{\frac{1}{N} \cdot \sum_{n=0}^{N-1}(x(n) - \overline{x})^3}{\left(\frac{1}{N} \cdot \sum_{n=0}^{N-1}(x(n) - \overline{x})^2\right)^{3/2}} \tag{2.13}$$

where $\overline{x}$ represents the mean of the magnitudes, m_3 is the sample third central moment, and m_2 is the sample variance.

2.1.3.4 Spectral Slope

Spectral slope represents the amount of decreasing of the spectral amplitude [13]. It is computed by linear regression of the spectral amplitude. In other words it is the slope of the line-of-best-fit through the spectral data.

2.1.3.5 Spectral Decrease

Spectral decrease also represents the amount of decreasing of the spectral amplitude. This formulation comes from perceptual studies and it is supposed to be more correlated to human perception. The formula is:

$$Decrease = \frac{1}{\sum_{n=1}^{N-1} x(n)} \cdot \sum_{n=1}^{N-1} \frac{x(n) - x(0)}{N-1} \tag{2.14}$$

where $x(n)$ represents the weighted frequency value, or magnitude, of bin number n.

2.1.3.6 Band Periodicity

Band periodicity is defined in [17] as the periodicity of a sub band and can be derived by sub band correlation analysis. In the current project it was chosen to use two different bands: the first one goes from 300 to 2500 Hz (called *Filtered-Band Periodicity*), the second one is from 0 to 22050 Hz (called *Whole-Band Periodicity*) as suggested in [1][2]. The periodicity property of each sub band is represented by the maximum local peak of the normalized correlation function calculated from the current frame and previous frame.

2.1.4 Correlation Features

These features are similar to spectral distribution descriptors but, in lieu of the spectrogram, they are computed starting from the auto-correlation function of each frame of the audio signal.

2.1.4.1 Correlation Slope

This feature is calculated giving the auto-correlation function of each frame of the audio signal as input to the *slope* function.

2.1.4.2 Correlation Decrease

This feature is calculated giving the auto-correlation function of each frame of the audio signal as input to the *decrease* function.

2.2 Gaussian Mixture Models (GMMs)

Gaussian Mixture Models (GMMs) are among the most statistically mature methods for clustering [18] and are gaining increasing attention in the pattern recognition community. GMMs are widely used in audio applications like speaker recognition and music classification. GMM is an unsupervised classifier which means that the training samples of a classifier are not labelled to show their category membership. More precisely, what makes GMM unsupervised is that during the training of the classifier, we try to estimate the underlying probability density functions (pdf's) of the observations. GMM can be classified as a semi-parametric density estimation method too, since it defines a very general class of functional forms for the density model. In this mixture model, a probability density function is expressed as a linear combination of basis functions. An interesting property of GMMs is that **the training procedure is done independently for the classes** by constructing a Gaussian mixture for each given class separately. So, adding a new class to a classification problem does not require retraining the whole system and does not affect the topology of the classifier making it attractive for pattern recognition applications. While GMM provides very good performances and interesting properties as a classifier, it presents some problems that may limit its practical use in real-time applications. One problem is that a GMM can require large amounts of memory to store various coefficients and complex computations mainly involving exponential calculations.

2.2.1 The multivariate Gaussian pdf

In the GMM classifier, the conditional-pdf of the observation vector is modeled as a linear combination of multivariate Gaussian pdfs, each of them with the following general form:

$$p(x) = \frac{1}{(2\pi)^{\frac{d}{2}} \cdot |\Sigma|^2} e^{\{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\}} \tag{2.15}$$

where :

- d is the number of features in the model;
- x is a d-component feature vector;
- μ is the d-component vector containing the mean of each feature;
- Σ is the d-by-d covariance matrix and $|\Sigma|$ is its determinant. It characterizes the dispersion of the data on the d-dimensions for the feature vector. The diagonal element σ_{ii} is the variance of x_i and the non diagonal elements are the covariances between features. Often

the assumption is that the features are independent, so Σ is diagonal and $p(x)$ can actually be written as the product of the univariate probability densities for the elements of x.

It is important to note that each multivariate Gaussian pdf is completely defined if we know $\theta = [\mu, \Sigma]$.

2.2.2 GMM Training

To classify data using features vectors in a class, GMM needs a training step. At this stage, we have to estimate the parameters of the multivariate Gaussian pdfs: $\theta_i = [\mu_i, \Sigma_i]$ with $i = 1, \ldots, k$ and k the number of multivariate Gaussian pdfs. In literature, the Expectation-Maximization algorithm (EM) is the most often used solution for this problem. EM is an iterative method which starts from a random distribution and alternates between performing an expectation (E) step, which computes the expectation of the log-likelihood evaluated using the current estimate for the latent variables, and a maximization (M) step, which computes parameters maximizing the expected log-likelihood found on the E step. These parameter-estimates are then used to determine the distribution of the latent variables in the next E step. This algorithm is assured to converge to a local optimum. Note that the training set provided to GMM has to be well thought out in order to have a model to be general enough to avoid the common problem of *overfitting*.

A simple Matlab example with real data will explain better the argument. To demonstrate the process, first generate some simulated data from a mixture of four bivariate Gaussian distributions using the *mvnrnd* function[2] (see Figure 2.3).

```
mu1 = [1 2];
sigma1 = [3 0.2; 0.2 2];
mu2 = [-1 -2];
sigma2 = [2 0; 0 1];
mu3 = [-4 2];
sigma3 = [1 0; 0 1];
mu4 = [-6 -2];
sigma4 = [2 -1; -1 2];

X = [mvnrnd(mu1,sigma1,100); mvnrnd(mu2,sigma2,100);
mvnrnd(mu3,sigma3,100); mvnrnd(mu4,sigma4,100)];
scatter(X(:,1),X(:,2),10,'ok')
```

Now we have to fit the data with the GMM training algorithm. Here, we know that the correct number of components to use is $k = 4$. Actually, with real data, this decision would require comparing models with different number of components.

```
gm = gmdistribution.fit(X,k);
```

[2]Here we assume that all data belong to the same class and are independent each other, even if they are not, simply to have a generic data distribution.

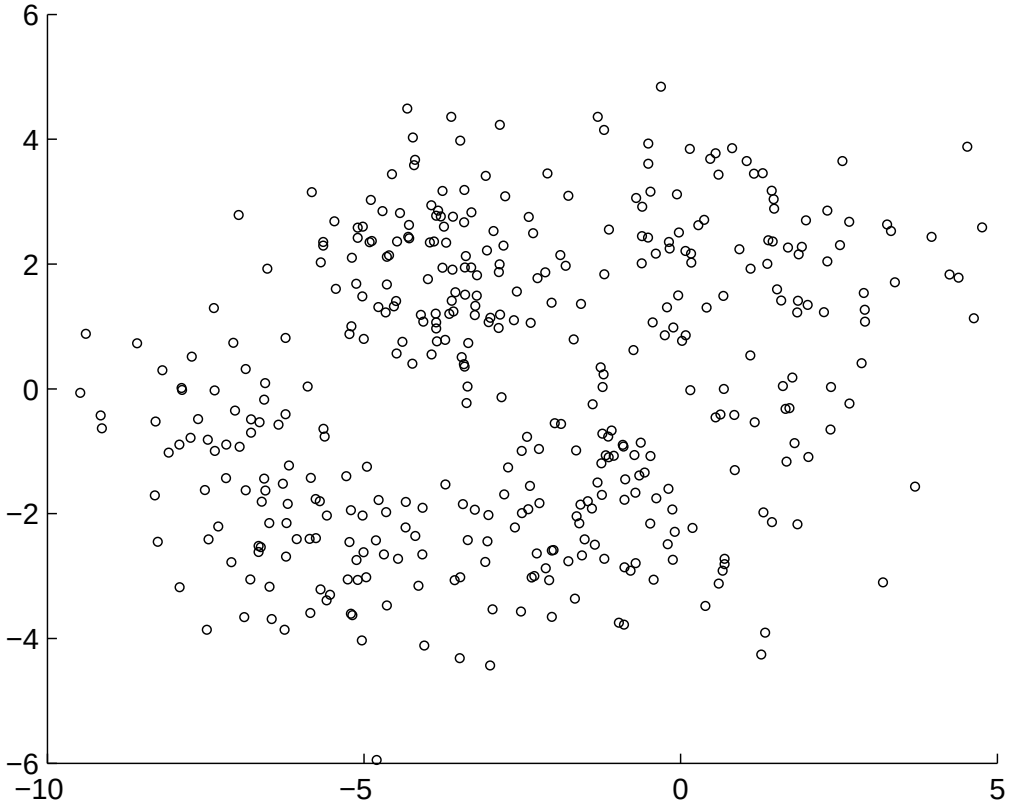

Figure 2.3: *Generic data distribution generated from a mixture of four bivariate Gaussian distributions.*

Fitting the data with a low number of Gaussians can lead to a bad fitting to the data. This is visible looking at the resulting estimated probability density with various values of the number of the components k in Figure 2.4.

2.2.3 Classification test

Once we trained the GMM, we can proceed to the classification test. This step consists in evaluating the log-likelihood of a feature vector x in the model. Now we need a decision step to discriminate if the features vector fits or not the model and say if the observation vector belongs to the model's class[3]. This step will be explained in detail in Chapter 3.

[3]Note that we have to construct a GMM for each class we want to insert in our classifier. So if we have to classify more than one class, we have to build a database in which all data/files must be divided into classes to separate different data coming from different classes. It follows that each model will be trained over all the data referred to a determinate class.

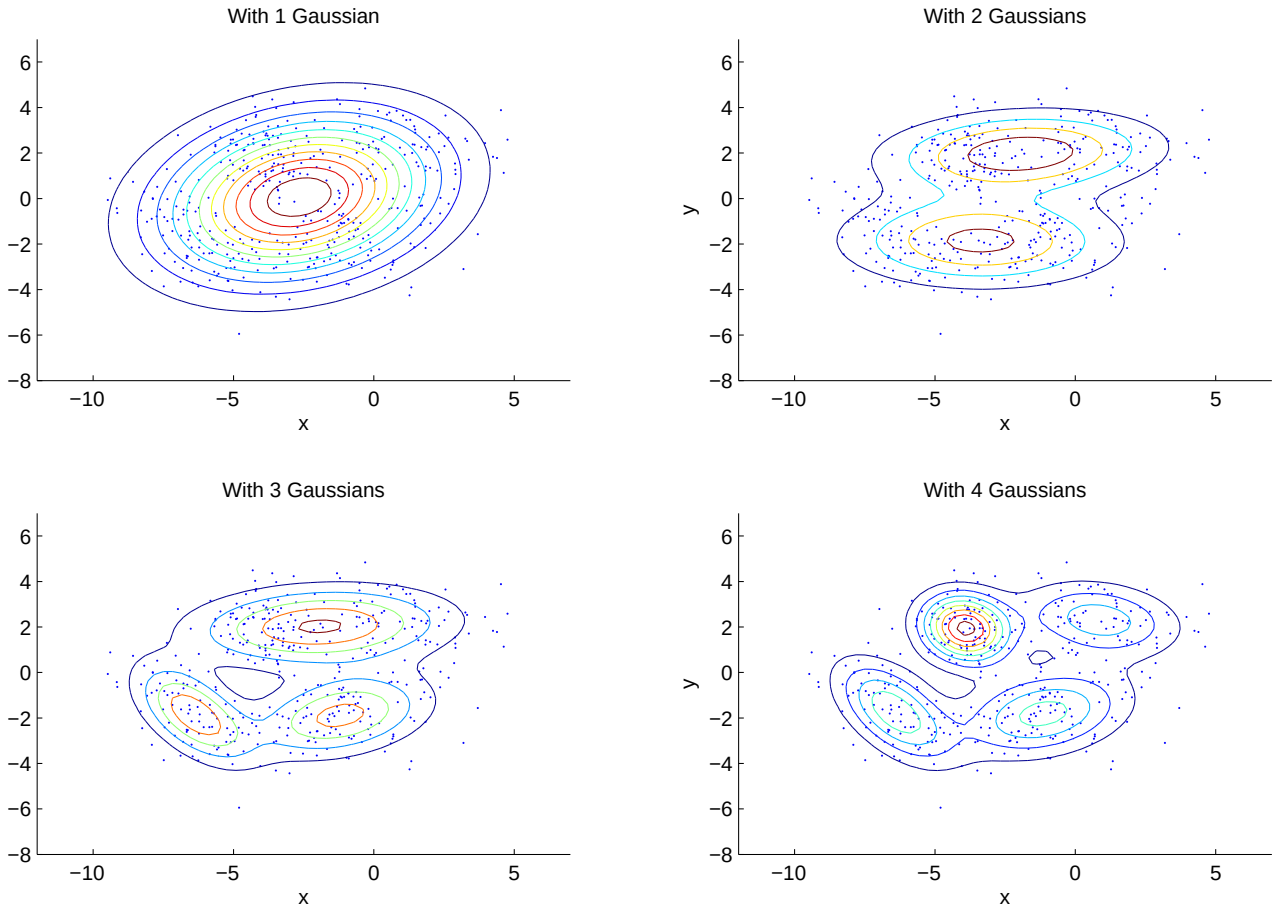

Figure 2.4: *Estimated probability density contours for the distribution with various values of the number of the components k.*

Chapter 3

Surveillance Algorithm for Recognition of Impulsive Sounds (SARIS)

3.1 Sound Classification

3.1.1 Creation of the database of impulsive sounds

In order to create the models of the four classes of sounds (gunshots, screams, broken glasses and barking dogs) we need a database of audio sounds. Unfortunately there are no Web database for the classification of environmental sounds, while there are many for the speech/speaker recognition that, in addition to being large, are made with certain criteria designed for the testing purpose. For example there are the *Timit database*[1] and the *Nist database*[2]. So we have the problem of creating an uniform database at home and the difficulty to deal with the literature because there is a common reference.

Our database has been created downloading audio files from *"The Freesound Project"*[19] without noise environment or eventually with a very high SNR. Every sound signal was stored with some properties that are also the initial conditions and criteria for the well-functioning of the algorithm. Every sound signal:

- has a sampling rate of 44100 Hz and has only one channel (mono)[3];
- has zero mean (the signal is centered on the x axis);
- is not normalized (the maximum absolute value of the signal is not necessarily 1);
- is cleaned from the first and last silence parts to create an homogeneous database and to have a robust training step without frames of sounds that not concern about the "essence" of the sound class.

[1] `http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC93S1`.

[2] `http://www.itl.nist.gov/iad/mig//tests/sre/`.

[3] Audio files with two channels (stereo) were transformed in one channel audio files (mono) summing the two channels arrays and halving the values of the obtained array to avoid clipping (.wav files have a range that goes from -1 to 1).

The number of sounds used in the database is listed in 3.1.

Sound Class	Number of files	Total duration (seconds)
Screams	41	41.3
Gunshots	30	18
Broken Glasses	21	7.2
Barking Dogs	28	2.9
Total	120	69.4

Table 3.1: *Number of files and total duration in seconds for each audio sound class.*

3.1.2 Creation of the classification models

For the classification of every class of the sounds, was selected a limited number of features (10-12) from those extracted (40 excluding TEO) from the signal, described in 2.1, in order to calculate in real-time the likelihood of all the models for each frame. The selection of the features followed mainly [2]. For a lack of a feature selection algorithm which lists in decreasing order the features that discriminate better the data in a determined class using thus only the first 10 or 15 "best" features, it was chosen to use the same features of the *Gunshots* for the *Broken Glasses* and the same features of the *Screams* for the *Barking Dogs*. This arbitrary choice is justified by a sort of similarity between the two couple of sounds, hoping that this would be not so different from the optimal choice of the features. In the table 3.2 are listed all the features used for the classification.

#	Gunshots and Broken Glasses classifiers	Screams and Barking Dogs classifiers
1	MFCC 1	MFCC 2
2	MFCC 2	MFFC 3
3	MFCC 3	MFCC 9
4	MFCC 11	MFCC 12
5	MFCC 28	SFM
6	MFCC 29	Skewness
7	MFCC 30	Spectral Slope
8	ZCR	Correlation Slope
9	Spectral Centroid	Spectral Decrease
10	Filtered Periodicity	Correlation Decrease
11		Periodicity
12		Filtered Periodicity

Table 3.2: *Features selected to create the Gaussian Mixture Models.*

For every class is trained a model from the background noise file with the features of the related class too. This helps to control if the current sound is part of the environmental noise

or a new sound and it is made simply comparing the likelihoods of the sound and noise models and "eliminating" the frames that fit better the noise model instead of the sound model of a determined sound class. This step, suggested in [1] and [2], will be treated with more detail later. All the features that are used to classify the sounds are extracted from 23 milliseconds analysis frames at a sampling frequency of 44100 Hz with 1/3 overlap as in [2] except for the sampling rate that was doubled. After the extraction of all the features for every class we are ready to run the GMM training algorithm of Matlab[4]. Note that the creation of GMMs for all the classes is made off-line. In real-time processing there is an initial step in which the models are loaded from *.txt* files[5] created and stored off-line. In the learning algorithm are used 6 components (Gaussians) because is a good trade off between the fitness of the models to data versus the CPU overload during real-time operations.

During the training and the classification steps, all the frames are considered independent to each other but the overlap between frames helps to correlate them to each other, introducing a sort of causality.

3.1.3 Classification step during audio processing

This step is the core of the algorithm[6]. Once loaded the models, for every frame, are calculated all the selected features and the related likelihood in all the models (sound and noise models). After that, it follows a cascade of three operators:

1. *Noise exclusion*;
2. *Likelihood selection*;
3. *Temporal selection*.

For each class and each frame is selected the largest likelihood between the two models (sound and noise). If the likelihood of the sound model is larger than the likelihood of the noise model, the likelihood is set to the likelihood of the sound model, zero otherwise. This step is called *"Noise exclusion"*.

```
if(likelihoodNoiseClassX > likelihoodSoundClassX)
    likelihoodClassX = 0;
else
    likelihoodClassX = likelihoodSoundClassX;
```

The second operator selects the frames that pass a threshold over the likelihood. If the frame does not pass the condition, its likelihood is set to zero. This operator is also called *"Likelihood selection"* because selects the likelihoods in the chart frames-likelihood (respectively x and y axis).

[4]For the project has being used the Matlab implementation with a maximum of 300 iterations.

[5]In this files are contained all the GMM parameters that permit to evaluate the likelihood of the frames in a model (see 2.2).

[6]This step can be applied both to test signals and in real-time.

```
if(likelihoodClassX < likelihoodThresholdClassX)
    likelihoodClassX = 0;
```

Now that the candidate frames are selected, is applied the third operator that is divided in two steps:

- *Weighted Moving Average* (WMA) of the likelihoods with a fixed number of frames for the window called $numFramesWMA$;
- selection of the likelihoods using a number of frames as threshold $numFramesWMADuration$ larger than $numFramesWMA$.

For the fist step was implemented a WMA defined as:

$$WMA_M = \frac{n \cdot p_M + (n-1) \cdot p_{(M-1)} + \cdots + 2 \cdot p_{(M-n+2)} + p_{(M-n+1)}}{n + (n-1) - \cdots + 2 + 1} \tag{3.1}$$

were p is the array of values and $M = numFramesWMA$ is the window length.

When calculating the WMA across successive values, it can be noted that the difference between the numerators of WMA_{M+1} and WMA_M is $n \cdot p_{M+1} - p_M - \cdots - p_{M-n+1}$. If we denote the sum $p_M + ... + p_{M-n+1}$ by $Total_M$, then

$$Total_{M+1} = Total_M + p_{M+1} - p_{M-n+1} \tag{3.2}$$

$$Numerator_{M+1} = Numerator_M + n \cdot p_{M+1} - Total_M \tag{3.3}$$

$$WMA_{M+1} = \frac{Numerator_{M+1}}{n + (n-1) + \cdots + 2 + 1} \tag{3.4}$$

In the second step, are discarded all the frames that have a number of consecutive *positive* values of the WMA less to $numFramesWMADuration$. In other words is used a binary variable associated to every frame of every sound class that determines if there was or not a sound of a determined class.

```
if(PassTemporalDuration)
    FrameEventClassX = 1;
else
    FrameEventClassX = 0;
```

This passage is made to avoid that sporadic and isolated frames that pass the *Noise exclusion* and the *Likelihood selection* steps detecting an event only for a single frame duration. In fact WMA flats the "spiky" values and permits to have an homogeneous likelihood. This operator is called *"Temporal selection"* because it considers only the frames that overcome the threshold for a determinate number of frames which can be seen as a temporal constraint. The thresholds

	threshold	numFramesWMA	numFramesWMADuration
Screams	0.1	30	35
Gunshots	$3 * 10^{-4}$	10	15
Broken Glasses	$5 * 10^{-6}$	10	13
Barking Dogs	0.1	10	13

Table 3.3: *Thresholds and the lengths of the windows used for selecting the likelihood of the sounds for detection.*

and the lengths of the windows are listed in Table 3.3 and were found manually with repeated test monitoring the accuracy of the classification algorithm.

Note that the cascade of the second and third operator is correct because if there are only zero values coming from the cascade, no frame passes the third selector because all the values after the WMA are zero and so there are no consecutive positive values. When the selection step is done we only have to plot the frames that pass the previous conditions in the temporal axis (see Figure 3.1. The decision that an event of the four classes occurs, is taken by computing the logical OR of the four classifiers (that are independent to each other) as in [2], to take into account the possibility of contemporaneous events.

3.2 Detection of impulsive sounds

The detection of the impulsive sounds is independent from the classification step and is used for the following reasons:

- to find the other impulsive sounds that are not taken in consideration in the classification step. They can be important for surveillance applications;
- to find also the impulsive sounds that are not detected by the classifier but belonging to one of the classes of sounds (miss).

To detect impulsive sounds is used TEO that is extracted over all the samples at a sampling frequency of 44100 Hz.

The idea of detecting the impulsive sound came from the one used for selecting the likelihoods of the models[7]. In fact we used a cascade of a *Threshold selection* followed by a *Temporal selection*. But the fact that TEO is a temporal feature calculated over each sample, led to a heavy processing time for the calculation of a WMA due to the fact that the windows were too large.

The problem was simply avoidable calculating WMA in the faster way for successive values as stated before. However, during the implementation of this detector in real-time, it was simpler, for the writing of the code[8], to aggregate in an unique value, with a simple mean, the values of the TEO belonging to a frame.

[7]This idea turned out to be very interesting because led to build a powerful impulse detector.

[8]The porting from Matlab to C++ suffered from the fact that all calculations were made off-line and for the use of the Matlab filters that were hidden built-in functions that has been ported in C++ using reverse engineering.

Therefore, 23 milliseconds windows were used to calculate the TEO feature with 1/3 overlap as in the classifier. In a second time, when the two detectors were divided in two NMM nodes, were avoided overlapping windows and the TEO's mean was calculated only over a window frame with length equal to the window step (about 7.6 milliseconds). In other words, now, is extracted only one TEO value per frame without overlap; it can be considered actually a sound feature like the others used to classify the sound. The parameters used in the operators are listed in Table 3.3.

threshold	numFramesWMA	numFramesWMADuration
$5 * 10^{-5}$	5	7

Table 3.4: *Threshold and the length of the windows used for selecting the TEO-signal of the sound.*

Summarizing, it follows that the system achieves the **detection of an impulsive sound with only one feature**!

3.3 Audio test creation

Testing was made using two types of audio test signals:

1. test signals for the accuracy calculation;
2. test signals for real-time testing.

In the first type the duration of the test is equal to the duration of the database tests. The new file is created by selecting a random piece of the background noise signal and adding a random signal of the database. The length of the noise audio signal is the equal to that of the pure audio signal. The addition is weighted by the $0 \leq NoiseAmplitudeModulation \leq 1$. To avoid clipping in .wav files is used this formula:

$$newSound = NoiseAmplitudeModulation \cdot cut + (1 - NoiseAmplitudeModulation) \cdot pureSound \quad (3.5)$$

where cut is the piece of the background noise signal and $pureSound$ is the clean signal from the database.

The Signal-To-Noise Ratio (SNR) of a new audio file with noise for the accuracy calculation is:

$$SNR\{dB\} = 10 \cdot \log_{10}(S_e/N_e) \quad (3.6)$$

where S_e is the total energy of the clean signal $\sum_n (pureSound[n]^2)$ and N_e is the total energy of the noise signal $\sum_n (cut[n]^2)$. Note that this formula is correct because the two signals $pureSound$ and cut have exactly the same length.

In the second type, tests are 20 seconds long (duration of the background noise used[9]): in this time lag is inserted in a random position[10], an arbitrary number of sounds taken from the database (as in Figure 3.1).

Figure 3.1: *Visualization of likelihood and detection in time. The upper plot is the detection over the time and in the other plots there are the values of the likelihood of each model. Legend: black - impulsive sound; red - gunshots; green - screams; blue - broken glasses; light blue - barking dogs.*

[9]Extract from an audio file taken in a London square downloaded from [19].

[10]Overlap between sounds can occur so it is possible to see the independence of each classifier in relation to the others

Chapter 4

Development tools and Network-Integrated Multimedia Middleware (NMM)

4.1 Development tools

In this section are listed all the development tools used in the project.

4.1.1 Middleware

- Network-Integrated Multimedia Middleware (NMM) described in deep detail in this chapter.

4.1.2 Software tools

4.1.2.1 Programming languages

The programming languages used for the implementation of the software are:

- Matlab - Octave;
- C/C++.

4.1.2.2 Integrated Development Environment (IDE)

The Integrated Development Environment (IDE) used are:

- Microsoft Visual Studio 2008 (for C++ Windows implementation);
- Eclipse (for C++ Linux implementation).

4.1.2.3 Operative Systems (OS)

The OS used are:

- Windows (XP - 7);
- Ubuntu 9.04 - 9.10.

4.1.2.4 Graphic User Interface (GUI)

For plotting charts of both off-line and real-time tests, it was used the Freeware library *ChartDirector* [20].

4.1.2.5 Subversion (SVN)

Subversion (also known as SVN) is an open source version control system designed by CollabNet Inc used for:

- saving and updating the source files of the project;
- comparing different chronological versions of the project for consideration of possible errors due to changes and updates;
- merging different versions of the same source file.

This tool was so helpful because the files were always protected from data loss and it fastened the update from one computer to another, also with different SO.

4.1.3 Audio tools

These tools are used mainly for the real-time tests:

- external sound card: Edirol UA-101: USB Audio Interface by Roland[1];
- microphone: SM58 by Shure[2];
- loud speaker: MP2-A Compact Amplified Monitor System by Generalmusic.

For the creation of the database of sounds is used the following software:

- Audacity. Sound editor for cutting audio files[3];
- Matlab - Octave. Functions for reading the .wav files.

[1] `http://www.rolandus.com/products/productdetails.php?ProductId=703`.

[2] `http://www.shure.com/americas/products/microphones/sm/sm58-vocal-microphone`

[3] `http://audacity.sourceforge.net/`.

4.2 Introduction to NMM

This section is a description of NMM taken from the NMM documentation [21]. Besides the PC, an increasing number of multimedia devices - such as set-top boxes, PDAs, and mobile phones - already provide networking capabilities. However, today's multimedia infrastructures adopt a centralized approach, where all multimedia processing takes place within a single system. The network is, at best, used for streaming predefined content from a server to clients. Conceptually, such approaches consist of two isolated applications, a server and a client (see Figure 4.1). The realization of complex scenarios is therefore complicated and error-prone, especially since the client has typically no or only limited control of the server.

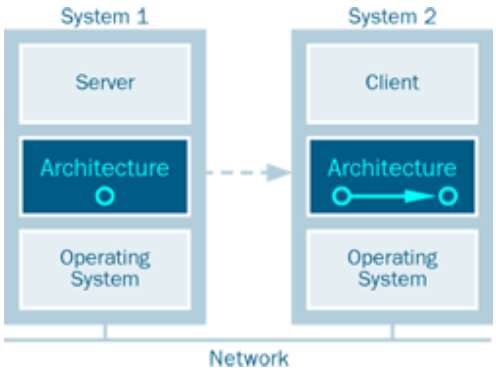

Figure 4.1: *Client/server streaming consists of two isolated applications that do not provide fine-grained control or extensibility.*

The Network-Integrated Multimedia Middleware (NMM) [21] overcomes these limitations by enabling access to all resources within the network: distributed multimedia devices and software components can be transparently controlled and integrated into an application. In contrast to all other multimedia architectures available, NMM is a *middleware*, i.e. a distributed software layer running in between distributed systems and application (see Figure 4.2).

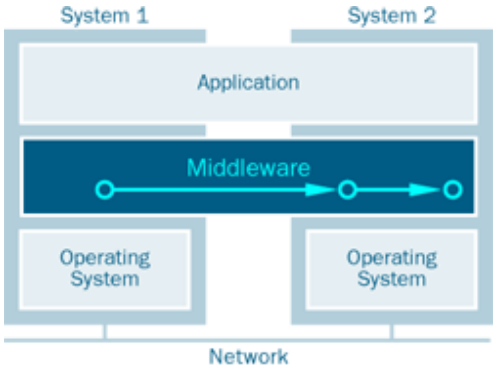

Figure 4.2: *A multimedia middleware is a distributed software layer that eases application development by providing transparency.*

As an example, this allows the quick and easy development of an application that receives TV from a remote device – including the transparent control of the distributed TV receiver. Even a PDA with only limited computational power can run such an application: the media conversions needed to adapt the audio and video content to the resources provided by the PDA can be distributed within the network. While the distribution is transparent for developers, no overhead is added to all locally operating parts of the application. To this end, NMM also aims at providing a standard multimedia framework for all kinds of desktop applications.

NMM is both an active research project at Saarland University in Germany and an emerging Open Source project. NMM runs on a variety of operating systems and hardware platforms. NMM is implemented in C++, and distributed under a dual-license: NMM is released under "free" licenses, such as the GPL, and commercial licenses.

4.2.1 Nodes, Jacks, and Flow Graphs

The general design approach of the NMM architecture is similar to other multimedia architectures. Within NMM, all hardware devices (e.g. a TV board) and software components (e.g. decoders) are represented by so called *nodes*. A node has properties that include its input and output ports, called *jacks*, together with their supported multimedia *formats*. A format precisely defines the multimedia stream provided, e.g. by specifying a human readable type, such as "audio/raw" for uncompressed audio streams, plus additional parameters, such as the sampling rate of an audio stream. Since a node can provide several inputs or outputs, its jacks are labelled with tags. Depending on the specific kind of a node, its innermost loop produces data, performs a certain operation on the data, or consumes data.

The system distinguishes between different types of nodes: a *source* produces data and has one output jack. A *sink* consumes data, which it receives from its input jack. A *filter* has one input and one output jack. It only modifies the data of the stream and does not change its format or format specific parameters. A *converter* also has one input and one output jack but can change the format of the data (e.g. from raw video to compressed video) or may change format specific parameters (e.g. the video resolution). A *multiplexer* has several input jacks and one output jack; a *demultiplexer* has one input jack and several output jacks. Furthermore, there is also a generic mux-demux node available. In section 4.4 will be explained step-by-step how to develop a new node.

These nodes can be connected each other to create a *flow graph*, where every two connected jacks need to support a "matching" format, i.e. the formats of the connected input jack respectively output jack need to provide the same type and all parameters and the respective values present in one format need to be available for the other and vice versa. The structure of this graph then specifies the operation to be performed, e.g. the decoding and playback of an MP3 file (see Figure 4.3).

Together, more than 60 nodes are already available for NMM, which allows for integrating various input and output devices, codecs, or specific filters into an application.

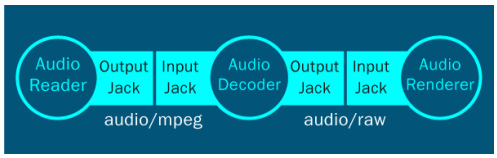

Figure 4.3: *A flow graph for playing back MP3 files.*

4.2.2 Messaging System

The NMM architecture uses a uniform messaging system for all communication. There are two types of messages. Multimedia data is placed into *buffers*. *Event* forward control information such as a change of speaker volume. Events are identified by a name and can include arbitrary typed parameters.

There are two different types of interaction paradigms used within NMM. First, messages are streamed along connected jacks. This type of interaction is called *instream* and is most often performed in *downstream* direction, i.e. from sources to sinks; but NMM also allows for sending messages in *upstream* direction.

Notice that both buffers and events can be sent instream. For instream communication, so called *composite events* are used that internally contain a number of events to be handled within a single step of execution. Instream events are very important for multimedia flow graphs. For example, the end of a stream (e.g. the end of a file) can be signalled by inserting a specific event at the end of a stream of buffers. External listener objects can be registered to be notified when certain events occur at a node (e.g. for updating the GUI upon the end of a file or for selecting a new file).

Events are also employed for the second type of interaction called *out-of-band*, i.e. interaction between the application and NMM objects, such as nodes or jacks. Events are used to control objects or for sending notifications from objects to registered listeners.

4.2.3 Interfaces

In addition to manually sending events, object-oriented *interfaces* allow to control objects by simply invoking methods, which is more type-safe and convenient then sending events. These interfaces are described in *NMM Interface Definition Language* (NMM IDL) that is similar to CORBA IDL. According to the coding style of NMM, interfaces start with a capital "I". For each description, an IDL compiler creates an interface and a implementation class. While an implementation class is used for implementing specific functionality within a node, an interface class is exported for interacting with objects. During runtime, supported events and interfaces can be queried by the application. Notice that interfaces described in NMM IDL describe out-of-band and instream interaction.

4.2.4 Distributed Flow Graphs

What is special about NMM is the fact that **NMM flow graphs can be distributed across the network**: local and remote multimedia devices or software components encapsulated within nodes can be controlled and integrated into a common multimedia processing flow graph, a *distributed flow graph*. While this distribution is transparent for application developers, no overhead is added to all locally operating parts of the graph: In such cases, references to already allocated messages are simply forwarded - no networking is performed at all.

The following shows an example for a distributed flow graph for playing back encoded (compressed) files, e.g. MP3 files. A source node for reading data from the local file system (*AudioReader*) is connected to a node for decoding audio streams (*AudioDecoder*). This decoder is connected to a sink node for rendering uncompressed audio using a sound board (*AudioRenderer*). Once the graph is started, the source nodes reads a certain amount of data from a given file, encapsulates it into buffers, e.g. of size 1024 bytes, and forwards these buffers to its successor. After being decoded to uncompressed "raw" audio, the converter node forwards data buffers to the sink node (see Figure 4.4).

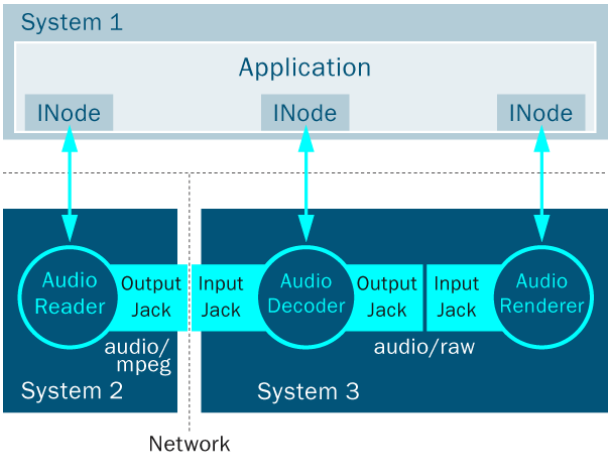

Figure 4.4: *A distributed flow graph for playing back MP3 files.*

The application controls all parts of this flow graph using interfaces, e.g. INode for controlling the generic aspects of all instantiated nodes. Notice that three different hosts are present in our example. The application itself runs on host1, the source node on host2, and the decoder and sink node on host3. Therefore, NMM automatically creates networking connections between the application and the three distributed nodes (out-of-band interaction), but also between the source and the decoder node (instream interaction). Therefore, compressed MPEG audio data is transmitted over the network.

Notice that such a simple but distributed flow graph already provides many benefits. First, it allows an application to access files stored on distributed systems without the need for a distributed file system, such as NFS. Second, the **data streaming between connected distributed nodes is handled automatically by NMM**. Third, the application acts as "remote control" for all

distributed nodes. As an example, this allows for transparently changing the output volume of the remote sound board by a simple method invocation on a specific interface, e.g. IAudioDevice.

4.2.5 Distributed Synchronization

Since NMM flow graphs can be distributed, they allow for rendering audio and video on different systems. For example, the video stream of an MPEG2 file can be presented on a large screen connected to a PC while the corresponding audio is played on a mobile device. To realize synchronous playback of nodes distributed across the network, NMM provides a generic distributed synchronization architecture. This allows for achieving lip-synchronous playback as required for the above described setup. In addition, media presentations can also be performed on several systems simultaneously. A common application is the playback of the same audio stream using different systems located in different rooms of a household - a home-wide music system.

The basis for performing distributed synchronization is a common source for timing information. It is used a static clock within each address space. This clock represents the system clock that is globally synchronized by the Network Time Protocol (NTP) and can therefore be assumed to represent the same time basis throughout the network. With the current setup, it is found the time offset between different systems to be in the range of 1 to 5 ms, which is sufficient for the purpose of the current thesis project.

4.2.6 Registry Service

The registry service in NMM allows discovery, reservation, and instantiation of nodes available on local and remote hosts. On each host a unique *registry server* administrates all NMM nodes available on this particular system. For each node, the server registry stores a complete *node description* that includes the specific type of a node (e.g. “sink”), its name (e.g. “PlaybackNode”), the provided interfaces (e.g. “IAudioDevice” for increasing or decreasing the output volume), and the supported input and output formats (e.g. an input format “audio/raw” including additional parameters, such as the sampling rate).

The application uses a *registry client* to send requests to registry servers running on either the local or remote hosts. Registry servers are contacted by connecting to a well-known port. After successfully processing the request the server registry reserves the requested nodes. Nodes are then created by a factory either on the local or remote host. For nodes to be instantiated on the same host, the client registry will allocate objects within the address space of the application to avoid the overhead of an interprocess communication.

To setup and create complex distributed flow graphs, an application can either request each node separately or use a *graph description* as query. Such a description includes a set of node descriptions connected by edges.

For an application to be able to create a distributed flow graph, the NMM application called *serverregistry* needs to be running on each participating host. For purely locally operating applications this is not required. Then, a server registry is running within the application itself but not accessible from remote hosts.

Before a server registry can be used, it needs to determine which devices and software components are available on a particular host. Therefore, the registry needs to be initialized once using following command:

```
user@linux:~/nmm/bin> ./serverregistry -s

Create config file with plugin information ...
Loading plugins...
AC3DecodeNode                                          available
AVDemuxNode                                            available
AVIReadNode                                            available
... and many further nodes

Config file successfully written.
```

4.2.7 Clic - An Application for Setting up NMM Multimedia Flow Graphs

The NMM framework is used to build up multimedia applications. The basic components in this framework are nodes which perform a certain functionality, like reading a video file or decoding it. Such nodes can be connected to a flow graph to perform a certain task like watching a film or transcoding it into another format.

Especially transcoder jobs do not need any additional user interaction if the application is running. Writing such an application without any user interaction is currently straightforward. First, the application requests these nodes from the registry, connects them to a flow graph and finally starts the entire flow graph.

To use the features provided by NMM we can follow two procedures. The first and easiest is discussed in this section and is based on the use of the executable *clic*; the second is to write, compile and run C++ code that uses libraries, namespaces and interfaces of NMM as can be seen in the *helloworld* examples located in the folder *examples* of NMM.

The application *clic* is used to build such a standard NMM application automatically from a textual description. *clic* stands for *Command Line Interaction and Configuration* and provides a tool for using the existing nodes with the possibility to configure some of their parameters. Starting from the knowledge of the functions of each node, textitclic can create a flow graph from its textual description which is called “graph description”. With *clic* becomes very easy to write and run an application that does not require interaction with a user; just correctly create a file with extension .gd (graph description) and run it with *clic*.

Will now follow the details of the syntax of the graph description and the usage of the *clic* application.

A graph description consists of two major parts. The first part is an (optional) comment introduced by the character %. The second part specifies the flow graph which describes how to connect the nodes. The following example describes a simple graph to play a .wav audio file.

```
% This graph description realizes a simple WAV player.
% Use the -i option of clic to specify an WAV file
```

```
WavReadNode ! ALSAPlaybackNode
```

To run this graph description, copy it into a file, e.g. wavplay.gd, and start clic with the following command:

```
./clic wavplay.gd -i /home/username/audio/song.wav
```

The option "*-i*" is used to specify the input file. To exit the application, enter "*q*" at the command line and press enter.

The flow graph section of a graph description describes how to connect the nodes. A node is identified by its name, which is (normally) the C++ class name. The exclamation mark (!) is used to denote a connection between two nodes as seen in the graph description for playing WAV files. In this example the statement:

```
WavReadNode ! ALSAPlaybackNode
```

results in clic to request the nodes WavReadNode and ALSAPlaybackNode. If all nodes can be successfully requested, the WavReadNode is connected to the ALSAPlaybackNode. Then, clic sets the specified parameters from the command line, for example an input file, and starts the entire flow graph.

In several cases a node must be configured before it can be used. In the previous examples, the name of a .wav file needs to be specified for the WavReadNode to read data from this file. Even though the clic-application offers a command line arguments for such a scenario, other nodes might require additional parameters, which can not be specified in this way. In this case, you ust call a node's specific method, offered by an NMM interface specified in NMM-IDL (Interface Definition Language). Each method of an NMM interface can be called from graph description as well. For this purpose you have to write the corresponding IDL method with all arguments prefaced by the character "$". Furthermore you have to specify the state in which the method must be executed. For example the WavReadNode offers the method *setFilename* in one of its interfaces to set a WAV file, which must be called in the state INITIALIZED. Furthermore you might want to change the queue of the ALSAPlaybackNode for incoming buffers, which can be done using the method *setDownstreamMaxSize* which must be called in the state CONSTRUCTED. The following example shows how to call these two methods from a graph description.

```
% This graph description describes a simple WAV player and shows
% how to call interface methods from a graph description. It is
% not required to set the input file using the option -i of clic.
  WavReadNode      $ setFilename("/home/username/audio/song.wav")
   INITIALIZED
! ALSAPlaybackNode $ setDownstreamMaxSize(1, "default") CONSTRUCTED
```

As can be seen in this example, the methods to be called are specified with their common name and additional parameters followed by the state of the node in which they should be executed.

Be careful using this feature. The given arguments are read as strings and converted to the expected type. Thus, it can cause errors if you set invalid arguments. Furthermore, you must specify all arguments: default arguments are currently not supported.

4.3 Installation of NMM

The middleware was installed using the 9.04 and 9.10 Ubuntu releases. It is also available to work in virtual machines because the emulation of the hardware peripherals like the sound card or video card does not create problems for the manage of the middleware.

4.3.1 Hardware prerequisites

For the Linux installation is needed a rightly configured video card with the Xv (X-Video) extension. This extension indicates a mechanism of output video for the X-Window system which is a windows visualization protocol used mainly in Unix-like systems. To know if it is all right run the following command:

```
xvinfo
```

This command prints all the informations of the video card for the X-Video extension. If it prints the settings of the video card then no changes are necessary, otherwise it returns an error. In this case you have to verify the correct installation of the *xvinfo* package on the Linux distribution used.

NMM also need a sound card or an audio chip able to reproduce sounds at different sampling rates (e.g. at 44.1 KHz). At present each personal computer has a sound card with this functionality integrated. By the way the precondition implies the proper recognition of the hardware by the operating system.

4.3.2 Network configuration

To configure the network must be opened all ports 22801 and the interval 5000-6000, that's to say that they should not be blocked by firewalls or other protection systems. Then it is necessary to set the hosts' file which are usually located in the /etc/hosts folder. Here we have to enter the domain, the hostname, and then, if there is a DNS (Domain Name System), we must also enter the IP addresses because NMM is unable to recognize them directly.

4.3.3 Software configuration

Before installing NMM we have to prepare the working environment. Specifically, working with Ubuntu, many of the packages required for the proper functioning of the middleware were absent and therefore it was required a prior installation. There are three ways to install:

- download the archive available by Motama containing the precompiled libraries and unpack it in a folder. These libraries do not necessarily cover all requirements and also does not guarantee the correctness of the operations in all system. The method is simple, but can sometimes cause problems redirecting thus to one of the following two options;
- download, compile and install the libraries using the source codes. All the informations and descriptions of the libraries needed for the right functioning are listed in A;
- install the package on the Linux distribution used. Beware that many distributions provide the header files, divided by library files called *developer*[4].

It is recommended to use the first solution, which involves a simple download of a package of already precompiled libraries (e.g. *nmm-2.2.0-optional-external-libs-linux.tar.gz*) and install it by following these instructions:

```
cd /home/username/
tar xvfz nmm-2.2.0-optional-external-libs-linux.tar.gz
```

Despite the advice, this alternative did not achieve the desired results, because after the installation of NMM many nodes were still not available. The second alternative, instead, was successful.

There are libraries for the management of audio such as "*asoundlib*", for the manage of the video as "*Xlib*" and for the interaction with various devices. Of course, for a complete installation, all the libraries should be installed; but if, for example, the firewire cameras are not used, the libraries "*l1394*" may be omitted.

Once identified the libraries, the installation can be done simply by using the *Synaptic Package Manager* in Ubuntu, making sure to install from the list all the files **-dev*. Some libraries, however, are not available through Synaptic and then require manual installation (for example, the ALSA libraries for managing audio).

Once prepared the environment, you can proceed with the actual installation of NMM. From site Motama you need to download the latest release (e.g. the file *nmm-2.2.0.tar.gz*) and unzip it to a folder (e.g. /home/username/nmm). From this new folder give these commands in sequence:

```
./configure
make
sudo make install
```

[4]If a package is called libABC, the package *developer file* will call most likely *libABC-developer* or *libABC-dev*.

The *./configure* permits to see which basic functions of NMM are enabled. At the beginning there should be entries marked *disabled*, but after the installation of all external packages, the items concerned should all be marked *enabled*. After performing these steps, if no errors occurs, NMM should be properly installed.

As a first test you can try the command:

```
serverregistry -s
```

that returns the list of all available nodes. The following is an example of the result of this command:

```
serverregistry and Network-Integrated
Multimedia Middleware (NMM) Version 2.2.0

Copyright (C) 2005-2010
Motama GmbH, Saarbruecken, Germany
http://www.motama.com

See licence for terms and conditions of usage

No plugin information available! If you start NMM for
the first time this information is created automatically.
Note: If you ever change your hardware configuration or
update the NMM version you must delete the file
/home/username/.nmm/plugins.2.2.0.xps._usr_local
or run 'serverregistry -s' again.

Create config file with plugin information ...
Loading plugins...
AC3DecodeNode                                   available
AC3ParseNode                                    available
ALSAPlaybackNode                                available
ALSARecordNode                                  available
AVDemuxNode                                     available
AVMuxNode                                       available
AnalyseDataIdNode                               available
AudioMuxNode                                    available
BrightnessNode                                  available
BufferDropNode                                  available
BufferShapingNode                               available
BufferTimestampControlNode                      available
BufferTimestampNode                             available
CopyNode                                        available
DVBS2ReadNode                                   not available
DVBSimulatorNode                                available
DevNullNode                                     available
```

```
DummyAudioSinkNode                      available
DummyVideoSinkNode                      available
FFMPEGDeinterlaceNode                   available
FFMpegAVIReadNode                       available
FFMpegAVIWriteNode                      available
FFMpegAudioDecodeNode                   available
FFMpegAudioEncodeNode                   not available
FFMpegDecodeNode                        available
FFMpegEncodeNode                        available
FLACReadNode                            available
FramerateConverterNode                  available
GenericReadNode                         available
GenericWriteNode                        available
H264DecodeNode                          available
IVTVReadNode                            not available
IcecastNode                             available
IdNode                                  available
JPEGDecodeNode                          available
JPEGEncodeNode                          available
LogoNode                                available
M4AReadNode                             available
MP3ReadNode                             available
MPEGAudioDecodeNode                     available
MPEGAudioEncodeNode                     available
MPEGDemuxNode                           available
MPEGReadNode                            available
MPEGTSDemuxNode                         available
MPEGTSReadNode                          available
MPEGTimeShiftingNode                    available
MPEGTimeShiftingNode2                   available
MPEGVSHDetectionNode                    available
MPEGVideoDecodeNode                     available
MagickManipulationNode                  available
MagickReadNode                          available
MagickWriteNode                         available
MessageSeekNode                         not available
NetSourceNode                           not available
OGMDemuxNode                            available
OSDManagerNode                          available
OggVorbisDecodeNode                     available
OverlayNode                             available
PCMDecodeNode                           available
PCMEncodeNode                           available
PNGReadNode                             available
PNGWriteNode                            available
```

```
PlaybackNode                                    available
RGBtoRGBConverterNode                           available
RGBtoYV12ConverterNode                          available
RawNode                                         available
RecordNode                                      available
ScopeNode                                       available
TSDemuxNode                                     available
TVCardReadNode                                  available
TVCardReadNode2                                 available
TimeDisplayNode                                 available
TimedBufferDropNode                             available
URLNode                                         available
VISCACameraReadNode                             not available
VideoCropNode                                   available
VideoGrabNode                                   available
VideoMuxNode                                    available
VideoScalerNode                                 available
VoiceDetectorNode                               not available
WavReadNode                                     available
WavWriteNode                                    available
WhiteNoiseNode                                  available
XDisplayNode                                    available
YUVDeInterlaceNode                              available
YUVtoRGBConverterNode                           available
YUVtoYUVConverterNode                           available
Finished loading plugins ...

Config file successfully written.
```

This list can be taken as an example of a first inspection of a NMM's correct installation.

4.3.4 Testing the middleware

After the installation, is required a testing phase that verifies the NMM's correct functioning. A first simple test can be the execution of a simple audio player using *clic*. First enter in the following NMM's directory path:

```
../nmm-2.2.0/apps/clic/gd/linux/playback/audio/
```

In this folder there are files with extension .gd ready for use. You can launch this example:

```
clic wavplay.gd -i <file audio .wav>
```

that will execute the file *wavplay.gd* with input any audio file *.wav*. This should allow you to hear the audio file.

In a second test you can try to view a video. The procedure is the same as before. From the folder:

```
nmm-2.2.0/apps/clic/gd/linux/playback/video/
```

give:

```
clic noise.gd
```

which should show a display window with video noise.

4.4 Software Developer Kit (SDK)

Motama provides a kit called NMM-SDK for software's development. Actually, this kit consists of a simple interface in NMM-IDL, a plugin that implements this interface and a flow graph using these two components, namely providing an example of what a developer should do to create his own plugins recognizable from the registry service (usable in a flow graph) and how to use them.

Let's see how to install the NMM-SDK. The most important prerequisite is the correct installation of NMM. After NMM's installation, download the SDK-package from the web (e.g. nmm-sdk-2.2.0) and unpack it. Now from the SDK folder give:

```
./configure --with-nmm=/usr/local
make
sudo make install
```

After the installation, go inside the folder:

```
...nmm-sdk-2.2.0/nmm/plugins/audio/filter
```

in which there is an example node, *AudioVolumeFilterNode*, that can open a file .wav and adjust the volume from the shell. The recommended way to learn how to build new nodes is to create an own node or otherwise modify the *AudioVolumeFilterNode* to take inspiration from it.

Watching .cpp files it is easy to understand the NMM's syntax but the first problem is trying to build the project including other external files (.cpp and .hpp) with the file AudioVolumeFilter.cpp.

To do this we must include in *make.am* the files used:

```
libnmmMyNode_la_SOURCES = \
    MyNode.cpp \
    file1.cpp \
    file2.cpp

treepkginclude_HEADERS = \
    $(IDLHEADERS) \
    MyNode.hpp \
    file1.hpp \
    file2.hpp
```

After editing the make.am file from the SDK directory we need to use the autotools functions[5] to generate the *make.in* and subsequently the *makefile*:

```
chmod +x autogen.sh && ./autogen.sh
```

Now we have everything ready. Now just give a reconfiguration, remembering that the SDK requires the NMM installation path:

```
./configure --with-nmm=/usr/local
```

Now that we generated the makefiles, we have to write from the SDK root:

```
make
sudo make install
```

If all ends without errors, writing:

```
serverregistry -s
```

you should see the name of our new node in the list:

```
MyNode          available
```

Note that if you use external libraries in your code, they have to be added into make.am.

4.5 Audio Surveillance Graph

The Audio Surveillance Graph is a cascade of five nodes, simply described by the *clic* syntax:

```
RecordNode $setDevice("/dev/audio1") CONSTRUCTED
!ImpulseDetectorNode
!AudioClassificationNode
!AudioEventReceiverNode
!XDisplayNode
```

The *RecordNode*, a SourceNode, records the sound at a fixed framerate and bitrate (respectively 44100 Hz and 16 bit/value) and sends the signal to the *ImpulseDetectorNode* that is a FilterNode like *AudioClassificationNode*. FilterNodes are nodes that have the output signal that is the same of the input signal and they only make an analysis and processing step of the signal. When they found an event, they notify it with an event-message that is propagated downstream in the graph. The *AudioEventReceiverNode*, a GenericFilterNode, is a listener of the graph events and catch all the events launched by the ImpulseDetectorNode and the AudioClassificationNode. GenericFilterNodes are nodes that have an input format different from the output one. In fact the

[5]For those not familiar with autotools or those who want to know more, take a look at this on-line guide `http://sourceware.org/autobook/autobook/autobook_toc.html`.

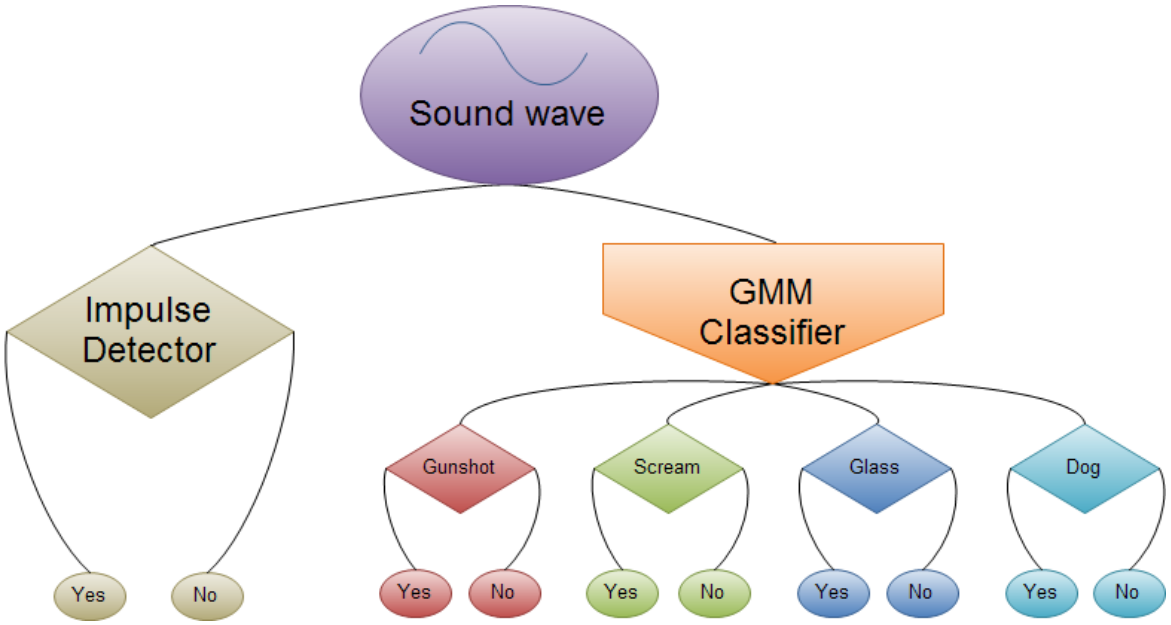

Figure 4.5: *The Audio Surveillance Graph.*

AudioEventReceiverNode receives in input an audio signal and sends in output an image that is the real-time plot of the analysis of the audio signal constructed with the timestamps of the events generated from the two filter nodes. Finally, to see the real-time plot, we need a connection to the *XDisplayNode*, a SinkNode, that opens a window with the plot refreshed every second. Note that using NMM has simplified a lot the managing of the audio stream that involves the mutual exclusion of the buffer containing the audio samples implemented as a circular array.

In a first implementation the three FilterNodes were unified in a single node that processed the audio signal and generated the plot. This, however, went in opposition with the modular way of thinking of NMM. So, the main functions of the old node were selected and the node was divided in three single processing nodes with distinct and independent aims. In fact classification and impulse detection are two distinct and separated processes (see Figure 4.5). In this way, a future user will have more freedom in locating the nodes in the graph, adding some new nodes between them or simply removing one of them if it is needless. we found a problem due to the integrated sound card that applied a DC offset (an additive component to the amplitude of the audio signal)[6] that caused a bad classification. In fact all the training files are stored in the database at zero mean by and so the real-time recorded sound from the microphone has to be at zero mean. To overcome this problem, was used an external usb audio card, the *Edirol UA-101: USB Audio Interface* by *Roland*, that sets by default the recorded sound at zero mean.

All files are included in a subdirectory called “audio” contained in the NMM-SDK “plugins” folder. To build the C++ project we used these makefiles:

[6]This problem has been observed in Ubuntu in which the recorded audio signal is translated in the amplitude axis

- ImpulseDetectorNode:

```
# Don't forget to add Makefile to configure.in (in project root directory)

# helper rules for nasm
include $(top_srcdir)/Makefile.am.extra

# this is a IDL Makefile, so include the according rules!
include $(top_srcdir)/Makefile.am.idlextra

# Add any subdirectories, where another Makefile.am is located
# "." is a cheap trick to force building of . first
SUBDIRS =

# library name
pluginlib_LTLIBRARIES = libnmmImpulseDetectorNode.la


# ---------- libnmmImpulseDetectorNode.la ----------

# all sources for this library in this directory
libnmmImpulseDetectorNode_la_SOURCES = \
ImpulseDetectorNode.cpp

# additional linker flags for the library
# version-info means the library compatibility number,
# see documentation for details
# do not remove, change only if you know what you do
libnmmImpulseDetectorNode_la_LDFLAGS = \
$(all_libraries) \
--version-info 0:0:0 -module

# libraries against which the library should be linked
# (example links against libtrallala.so and so on)
libnmmImpulseDetectorNode_la_LIBADD =

# all header files, that are to install in the current directory
treepkginclude_HEADERS = \
$(IDLHEADERS) \
ImpulseDetectorNode.hpp

# all extra things you want to be included in the tar-ball of the distribution
# (standard files like *.cpp, *.hpp, README or Makefile.am will be included
# anyway)
EXTRA_DIST = \
$(IDLS)

# additional include paths needed to compile the librar{y,ies}
INCLUDES = $(USER_INCLUDES) \
$(all_includes)
```

- AudioClassificationNode:

```
# Don't forget to add Makefile to configure.in (in project root directory)

# helper rules for nasm
include $(top_srcdir)/Makefile.am.extra

# this is a IDL Makefile, so include the according rules!
include $(top_srcdir)/Makefile.am.idlextra

# Add any subdirectories, where another Makefile.am is located
# "." is a cheap trick to force building of . first
SUBDIRS =

# library name
pluginlib_LTLIBRARIES = libnmmAudioClassificationNode.la


# ---------- libnmmAudioClassificationNode.la ----------

# all sources for this library in this directory
libnmmAudioClassificationNode_la_SOURCES = \
AudioClassificationNode.cpp \
gmm.cpp \
mathPlus.cpp \
    feature.cpp

# additional linker flags for the library
# version-info means the library compatibility number,
# see documentation for details
# do not remove, change only if you know what you do
libnmmAudioClassificationNode_la_LDFLAGS = \
$(all_libraries) \
--version-info 0:0:0 -module

# libraries against which the library should be linked
# (example links against libtrallala.so and so on)
libnmmAudioClassificationNode_la_LIBADD =

# all header files, that are to install in the current directory
treepkginclude_HEADERS = \
$(IDLHEADERS) \
AudioClassificationNode.hpp \
    gmm.h \
    mathPlus.h \
    feature.h

# all extra things you want to be included in the tar-ball of the distribution
# (standard files like *.cpp, *.hpp, README or Makefile.am will be included
# anyway)
EXTRA_DIST = \
$(IDLS)

# additional include paths needed to compile the librar{y,ies}
INCLUDES = $(USER_INCLUDES) \
$(all_includes)
```

- AudioEventReceiverNode:

```
# Don't forget to add Makefile to configure.in (in project root directory)

# helper rules for nasm
include $(top_srcdir)/Makefile.am.extra

# this is a IDL Makefile, so include the according rules!
include $(top_srcdir)/Makefile.am.idlextra

# Add any subdirectories, where another Makefile.am is located
# "." is a cheap trick to force building of . first
SUBDIRS =

# library name
pluginlib_LTLIBRARIES = libnmmAudioEventReceiverNode.la


# ---------- libnmmAudioEventReceiverNode.la ----------

# all sources for this library in this directory
libnmmAudioEventReceiverNode_la_SOURCES = \
AudioEventReceiverNode.cpp

# additional linker flags for the library
# version-info means the library compatibility number,
# see documentation for details
# do not remove, change only if you know what you do
libnmmAudioEventReceiverNode_la_LDFLAGS = \
$(all_libraries) \
--version-info 0:0:0 -module \
-L/home/riccardo/nmm-sdk-2.2.0/nmm/plugins/audio/AudioEventReceiverNode \
-Wl,-R/home/riccardo/nmm-sdk-2.2.0/nmm/plugins/audio/AudioEventReceiverNode \
-lchartdir

# libraries against which the library should be linked
# (example links against libtrallala.so and so on)
libnmmAudioEventReceiverNode_la_LIBADD = \
/home/riccardo/nmm-2.2.0/nmm/utils/png/libnmmpngutils.la

# all header files, that are to install in the current directory
treepkginclude_HEADERS = \
$(IDLHEADERS) \
AudioEventReceiverNode.hpp \
chartdir.h \
bchartdir.h \
memblock.h \
FinanceChart.h

# all extra things you want to be included in the tar-ball of the distribution
# (standard files like *.cpp, *.hpp, README or Makefile.am will be included
# anyway)
EXTRA_DIST = \
$(IDLS)

# additional include paths needed to compile the librar{y,ies}
INCLUDES = $(USER_INCLUDES) \
$(all_includes)
```

Chapter 5

Experimental Results

5.1 Testing Off-line

To know the performance of the system, testing off-line is made over the whole database, working on audio files registered with a microphone. Three test variables are taken into account. The former two are boolean, the third is numerical.

1. *Noise exclusion on/off*. This variable chooses whether to use or not the Noise exclusion (if set to "off" is not required a noise classification step);

2. *Technique of evaluating accuracy smooth/sharp*. The simultaneous detection of different classes of the same file belonging to one determinate class is evaluated as a correct detection in the smooth accuracy mode, a wrong detection in the sharp accuracy mode;

3. *Signal-to-Noise Ratio (SNR)*. This variable goes from -20 dB to 25 dB with step of 5 dB.

Let N the number of the classes used in the system and C the set of all the classes such that $c_{1,\dots,N} \in C$. The performance parameters used are explained in the following:

- *True positive*: A sound file belonging to a class $c_i \in C$ and correctly classified in a sound belonging to the same class c_i. TP is the number of true positive files tested;

- *False positive*: A generic sound file not belonging to any of the classes of C that is wrongly classified in a sound belonging to some class $c_i \in C$. Note that in a multi class system with N classes, this parameter incorporates both the prediction that the file that belongs to a class $c_i \in C$ is wrongly classified in a sound belonging to a class $c_j \in C$ with $j \neq i$, and the prediction that the audio file that does not belong to any class is wrongly classified in a sound belonging to some class $c_i \in C$. FP is the number of false positive files tested;

- *True negative*: A sound file that does not belong to any class $c_i \in C$ and is not detected by the system. TN is the number of true negative files tested;

- *False negative*: A sound file belonging to a class $c_i \in C$ and not classified in any class. FN is the number of false negative files tested.

The accuracy is the proportion of true results (both true positives and true negatives) in a database:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{FN} + \text{TN}} \tag{5.1}$$

On the other hand, precision is defined as the proportion of the true positives against all the positive results (both true positives and false positives):

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{5.2}$$

Accuracy is the degree of veracity while precision is the degree of reproducibility. The analogy used here to explain the difference between accuracy and precision is the target comparison. In this analogy, repeated measurements are compared to arrows that are shot at a target. Accuracy describes the closeness of arrows to the bullseye at the target center. Arrows that strike closer to the bullseye are considered more accurate. The closer a system's measurements to the accepted value, the more accurate the system is considered to be (see Figure 5.1).

Figure 5.1: *High accuracy but low precision.*

To continue the analogy, if a large number of arrows are shot, precision would be the size of the arrow cluster. (When only one arrow is shot, precision is the size of the cluster one would expect if this were repeated many times under the same conditions.) When all arrows are grouped tightly together, the cluster is considered precise since they all struck close to the same spot, even if not necessarily near the bullseye. The measurements are precise, though not necessarily accurate (see Figure 5.2).

Figure 5.2: *High precision but low accuracy.*

However, it is not possible to reliably achieve accuracy in individual measurements without precision: if the arrows are not grouped close to one another, they cannot all be close to the

bullseye (their average position might be an accurate estimation of the bullseye, but the individual arrows are inaccurate).

To increase almost linearly the noise over the clean signals, the background noise was added to the clean signal changing the $NoiseAmplitudeModulation$ from 0 to 0.9 with 0.1 step.

Unfortunately there is not a good definition of SNR for a set of signals because the audio files in the database have different loudness. Our definition of the mean SNR makes sense to characterize a real situation where there are audio streams with almost stationary noise, with a certain energy, and various signals with different energies.

SNR's Average value makes sense because it can be representative of a real situation. Every SNR value reported in the graphs is the mean of the SNRs of all the files referring to the same $NoiseAmplitudeModulation$ value (the variance has a constant range of ± 10 dB). All the following plots are structured in this way: in the x axis there is the mean SNRs and in the y axis there are the statistical results of accuracy and precision. All data are the mean of 10 iterations of the *accuracy & precision algorithm* because the training and the audio test-files construction are random based.

5.1.1 Global classification performance

Here we show the global performance of the systems with all the four classes of sounds. The values are calculated using all the files of the database for the training of the GMMs. Although here the training set and the test set are the same, the purpose of this analysis is to look at the effect of noise over the performance. In Figure 5.3 the global accuracy decreases as the SNR decreases. Using the Noise Subtraction it seems that the accuracy does not improve; this means that we can **avoid to train the GMMs of the noise** that varies according to the place in which we want to install the system.

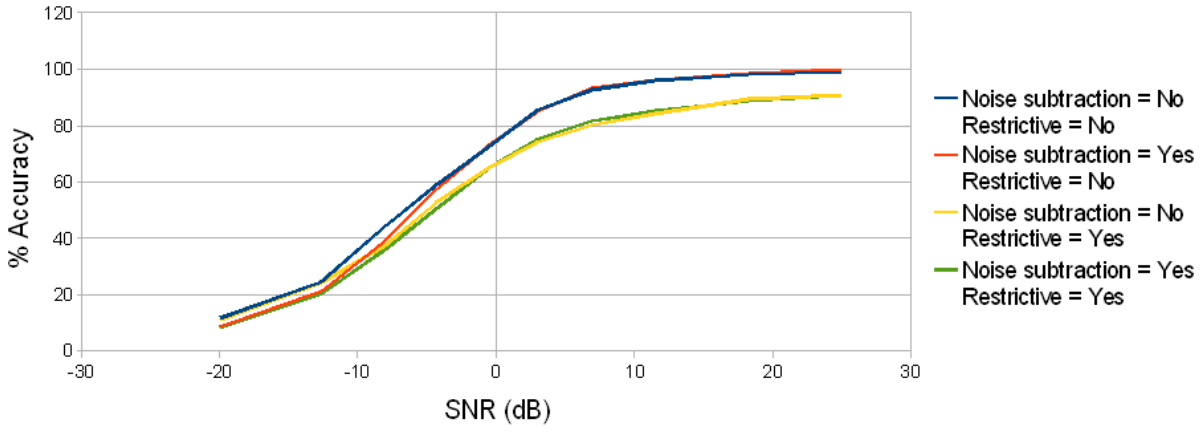

Figure 5.3: *Accuracy of all the classes.*

From Figure 5.4 we can observe that the whole system is very precise also when there is a low SNR. Obviously, using the restrictive mode, the accuracy and the precision are lower; this is

due to simultaneous detection of different classes of the same file belonging to one determinate class.

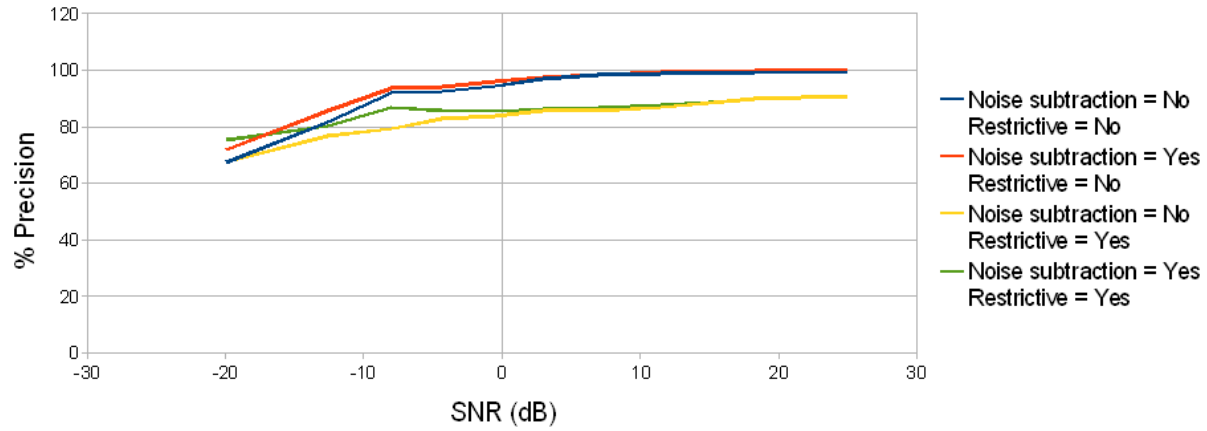

Figure 5.4: *Precision of all the classes.*

5.1.2 Single class classification performance

Now it follows a single performance evaluation of the four classifiers. In this section are tested only the sounds belonging to a determinate class and are taken into account the results of all the classifiers. This will show if there are some "interferences" between classes and if some classes performs better than others. These tests are also a way to determine whether the features used to classify the sounds of a given class are discriminatory for that class in relation with the other classes.

For the evaluation of the performance we followed two methods:

1. use of all the database files both for the training and for the test (training set = test set) as in the global performance;

2. *leave-one-out cross-validation method* (LOOCV).

The second approach is more accurate than the first one because it achieves more robust performance values. As the name suggests, LOOCV involves using a single observation from the original sample as the validation data, and the remaining observations as the training data. This is repeated such that each observation in the sample is used once as the validation data. This is the same as a K-fold cross-validation with K being equal to the number of observations in the original sample. LOOCV, however, is usually very expensive from a computational point of view because of the large number of times the training process is repeated.

5.1.2.1 Gunshots

Simple validation

For the gunshot class we can see (Figures 5.5 and 5.6) that the behaviour is the same as in the overall performance but it worses as the SNR decreases.

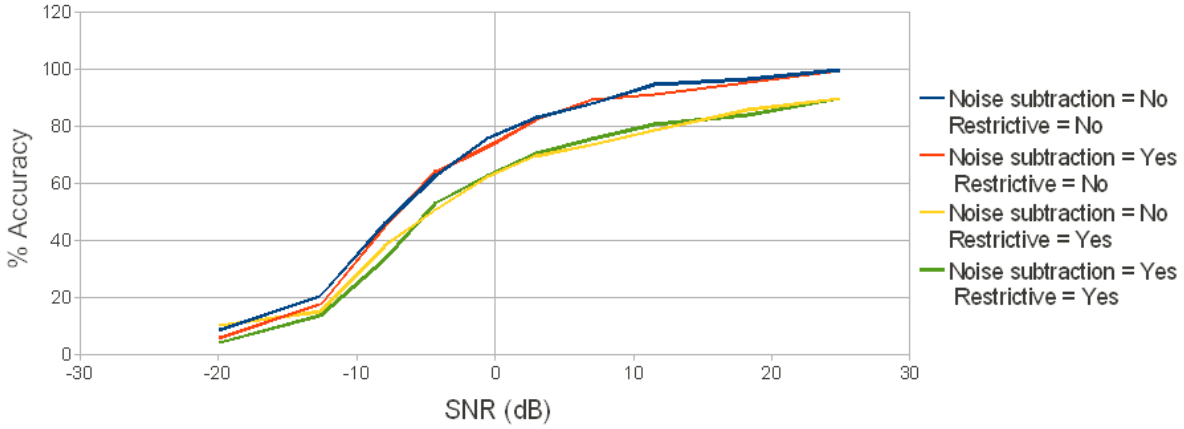

Figure 5.5: *Accuracy of the Gunshot class with simple validation.*

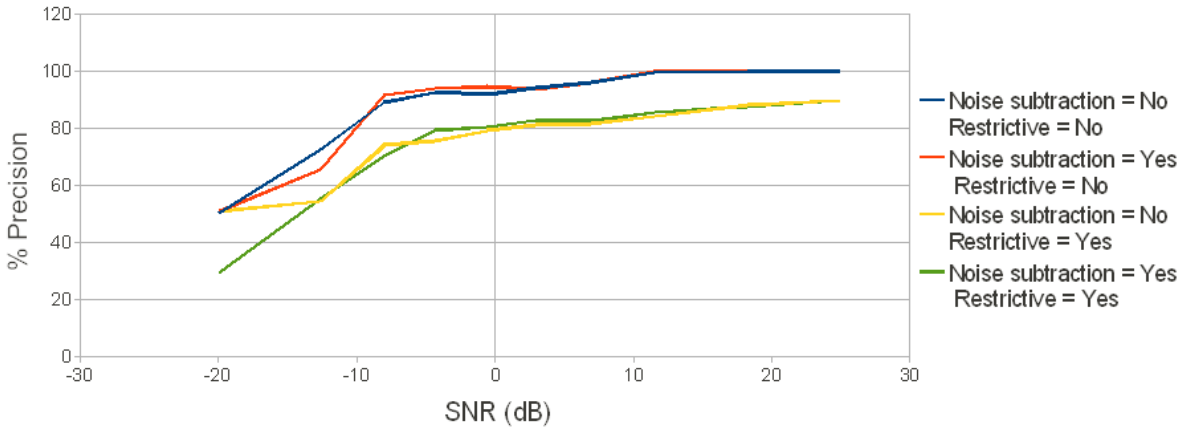

Figure 5.6: *Precision of the Gunshot class with simple validation.*

Leave-one-out cross-validation

The accuracy (Figure 5.7) decreases slowly from 80% at 25 dB SNR remaining at a high value of 40% at -20 dB SNR. The precision (Figure 5.8) is very good and is quite constant to 90% in all the SNR's tested range. Very little difference in using the noise exclusion.

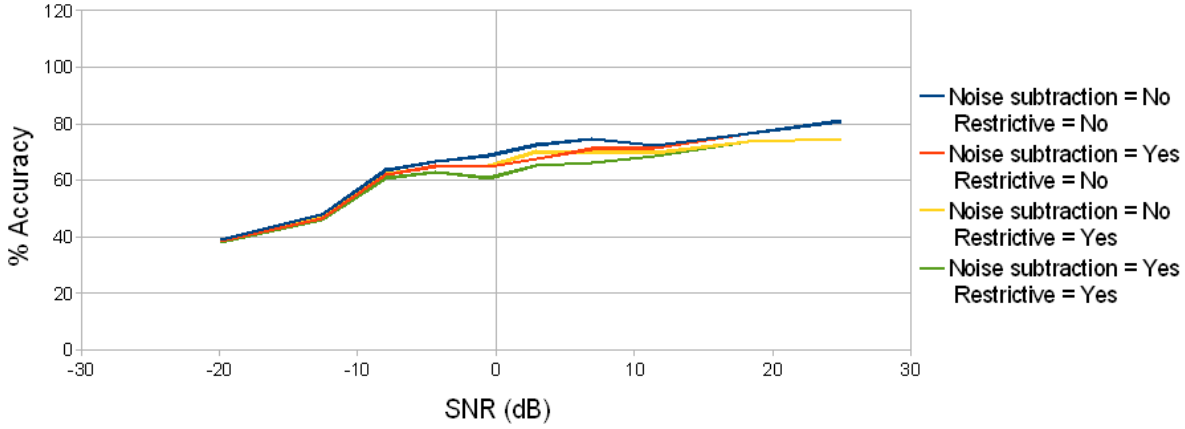

Figure 5.7: *Accuracy of the Gunshot class with LOOCV.*

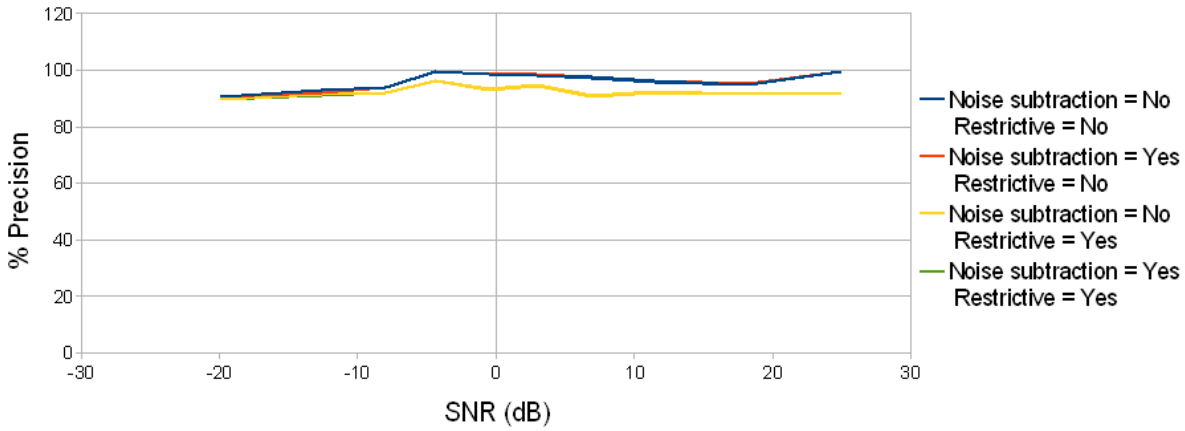

Figure 5.8: *Precision of the Gunshot class with LOOCV.*

5.1.2.2 Screams

Simple validation

In this class we can see (Figures 5.9 and 5.10) that there is a larger gap between the two technique for evaluating the accuracy (smooth/sharp) than in the other plots. This may mean that the model of the screams classify well the sounds of its class (red and blue curves), but other models have a lot of false positives because some screams are classified as belonging also to other classes (yellow and green curves). This may be because we use the same features to classify different classes of sounds (e.g. the screams and barking dogs classes). To understand which classes confuse the sounds we should build a confusion matrix.

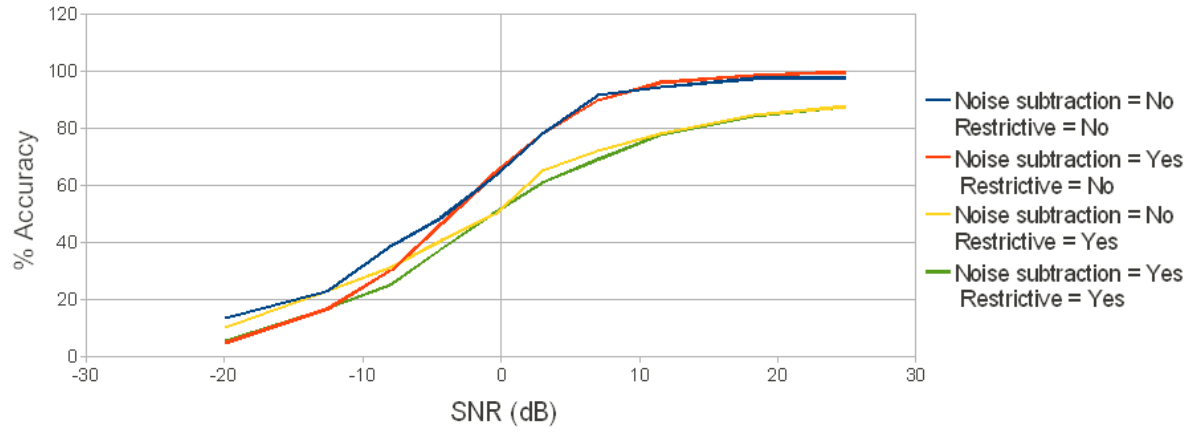

Figure 5.9: *Accuracy of the Scream class with simple validation.*

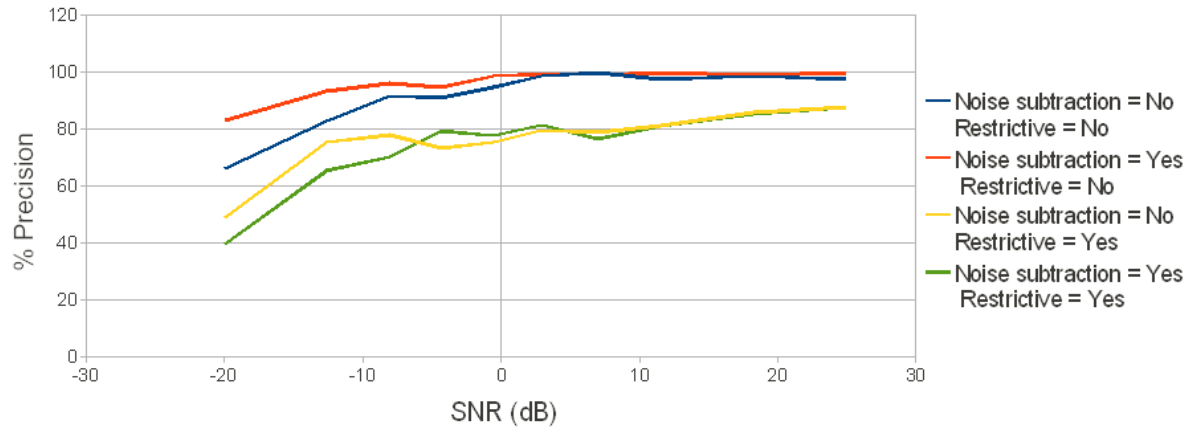

Figure 5.10: *Precision of the Scream class with simple validation.*

Leave-one-out cross-validation

In the Figures 5.11 and 5.12 the gap is larger than in Figures 5.9 and 5.10 (same and strengthened inferences as before). The accuracy now start from less than 80% and there is no difference in using the noise exclusion.

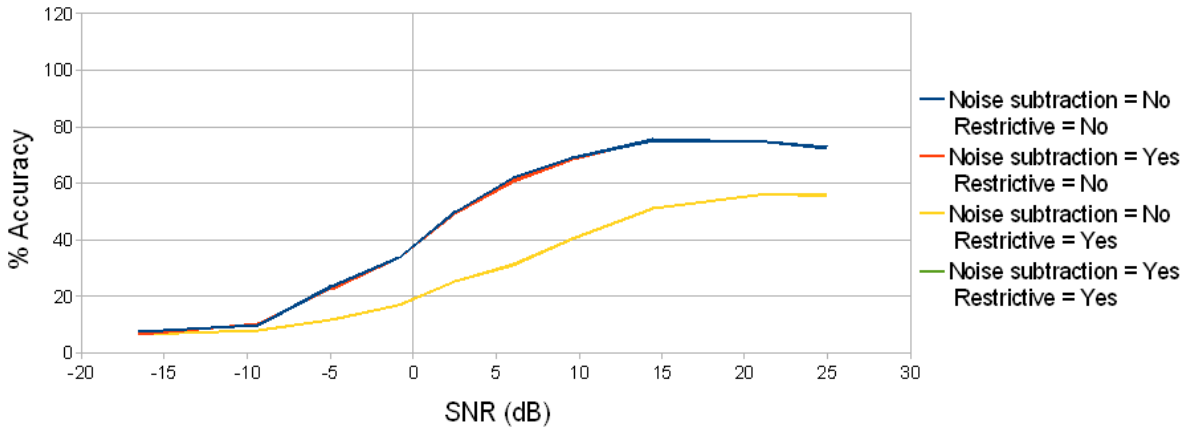

Figure 5.11: *Accuracy of the Scream class with LOOCV.*

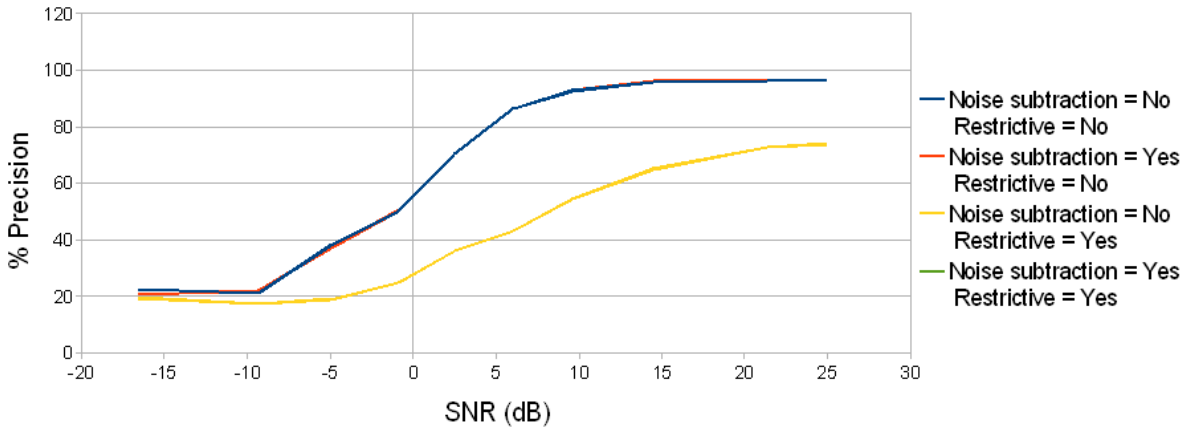

Figure 5.12: *Precision of the Scream class with LOOCV.*

5.1.2.3 Broken Glasses

Simple validation

This class performs better than the others (very high accuracy and precision) and looking to the Figures 5.13 and 5.14 we can see that there is no difference in using the noise subtraction or not. We can also see that there are no false positives and so no interferences with the other classes (the yellow and green lines lie on the blue and red curves). This is probably due to the small database regarding this class that is correlated also to the fact that precision is very high, suggesting that the model does not generalizes too much and suffers of overfitting. Furthermore we see that the precision is almost constant although the SNR decreases, perhaps because the sounds of this class are so similar to white noise.

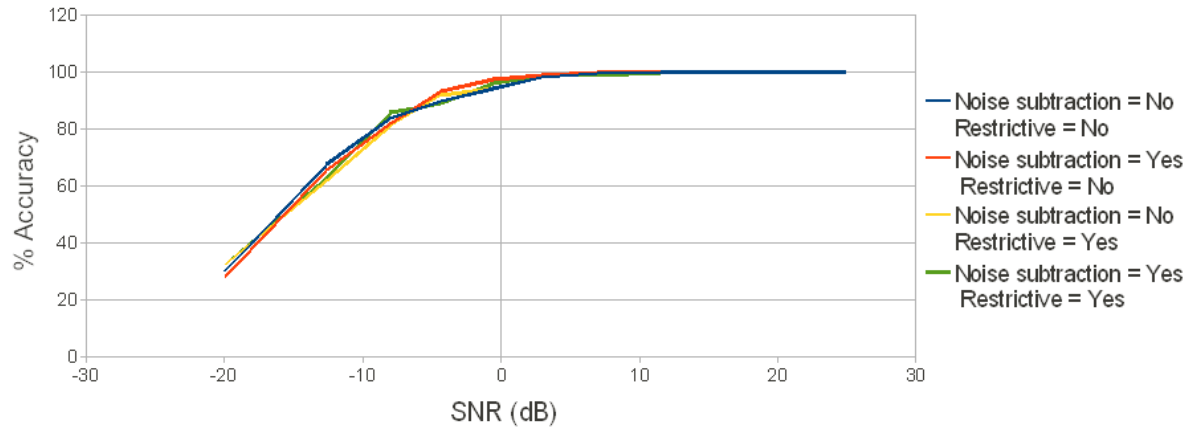

Figure 5.13: *Accuracy of the Broken Glasses class with simple validation.*

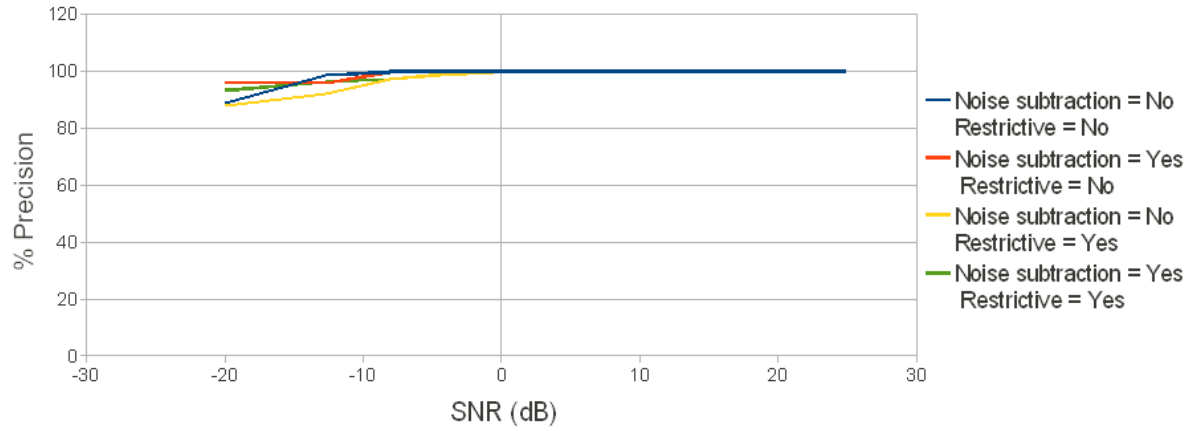

Figure 5.14: *Precision of the Glass class with simple validation.*

Leave-one-out cross-validation

Here the precision (Figure 5.16) is not so constant as in the simple valuation and there is a little interference with the other classes. No difference with the noise exclusion.

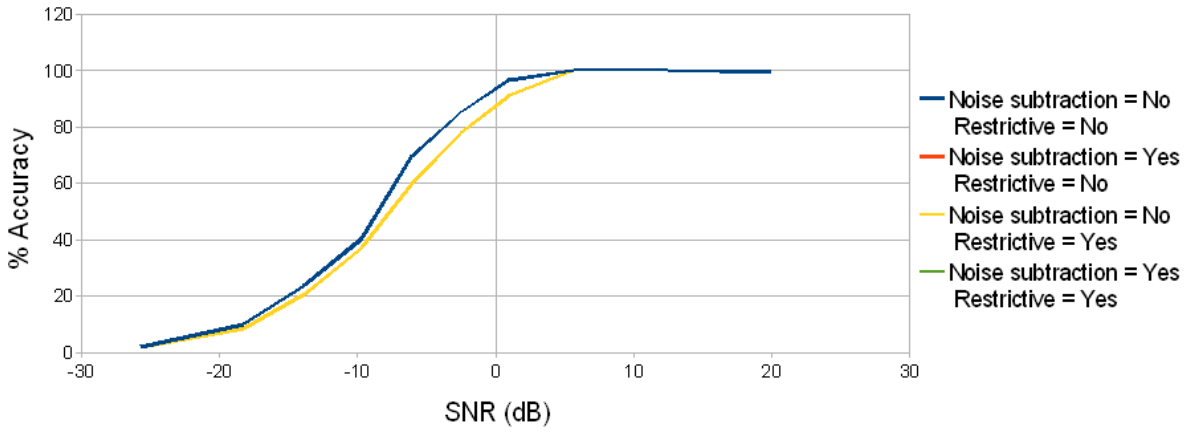

Figure 5.15: *Accuracy of the Glass class with LOOCV.*

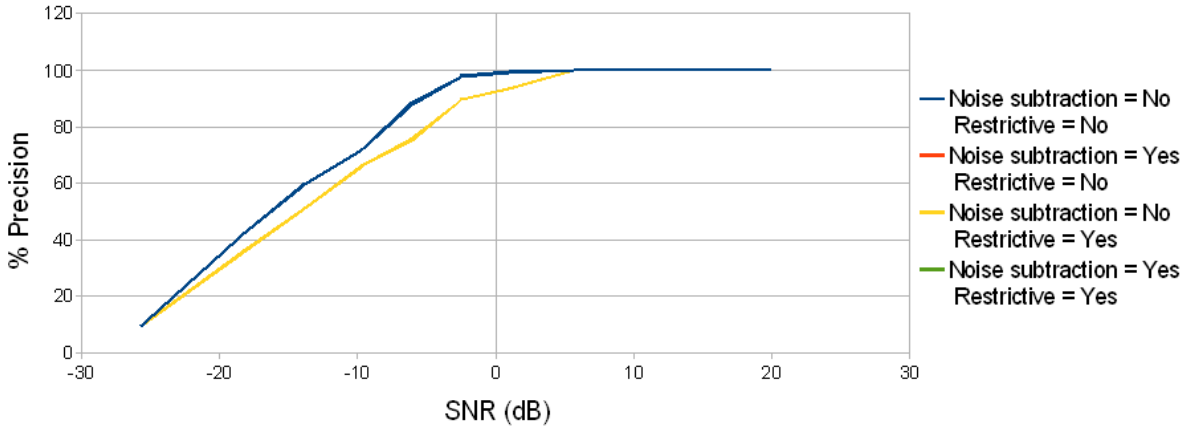

Figure 5.16: *Precision of the Glass class with LOOCV.*

5.1.2.4 Barking dogs

Simple validation

This class performs quite well with medium high SNRs. With low SNRs accuracy and precision go to zero. With negative SNRs, the lines of the two different techniques of evaluation (smooth/sharp) have the same behaviour, suggesting that other classes does not wrongly detect barks as sounds of their class.

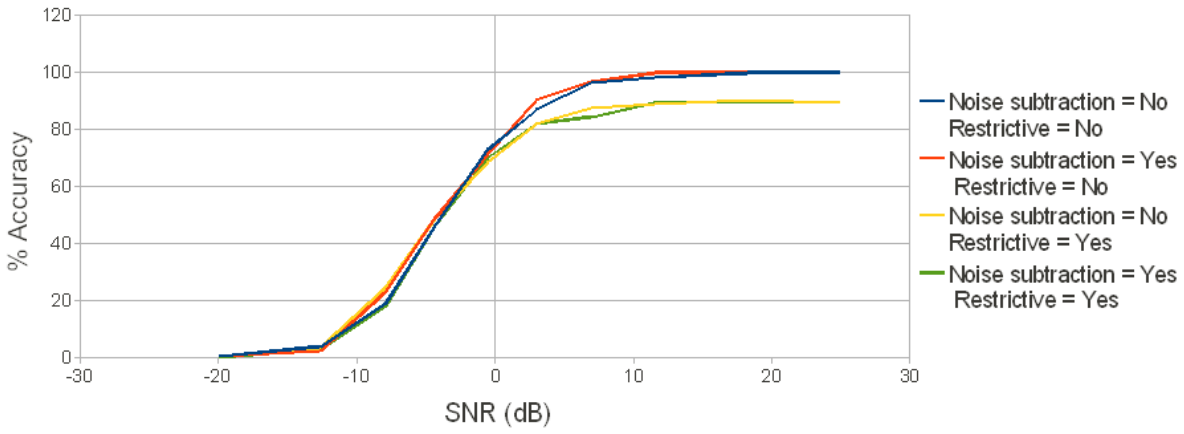

Figure 5.17: *Accuracy of the Barking dogs class with simple validation.*

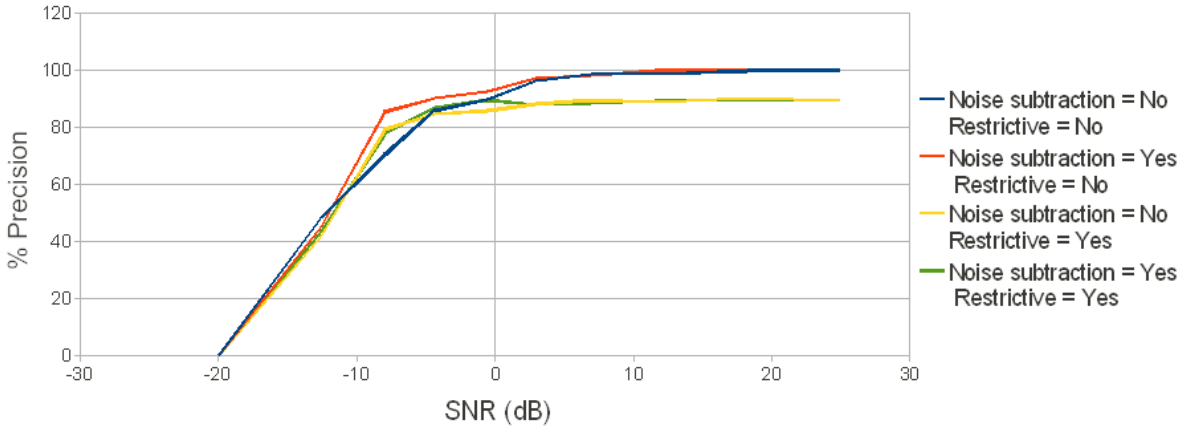

Figure 5.18: *Precision of the Bark class with simple validation.*

Leave-one-out cross-validation

Same notes stated in the simple evaluation paragraph of this class. Still no difference in using the noise exclusion.

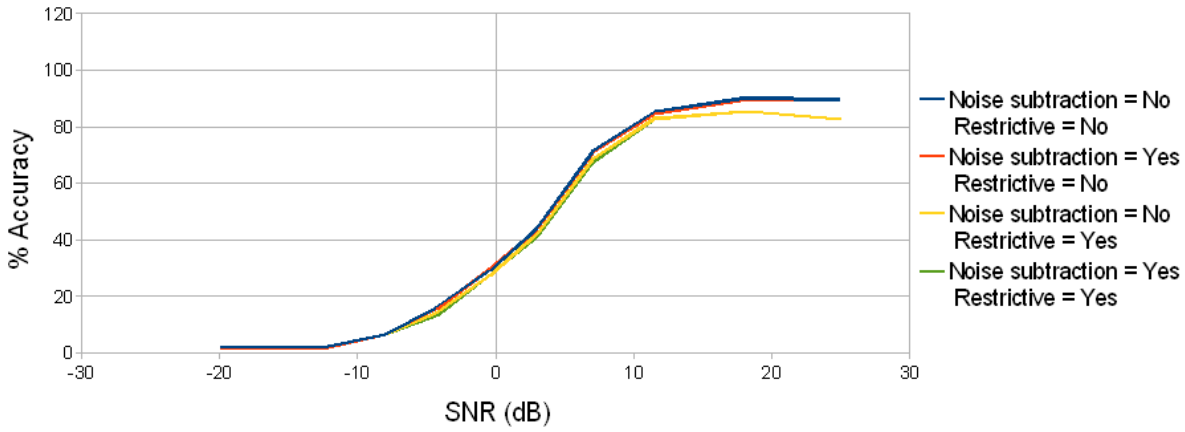

Figure 5.19: *Accuracy of the Bark class with LOOCV.*

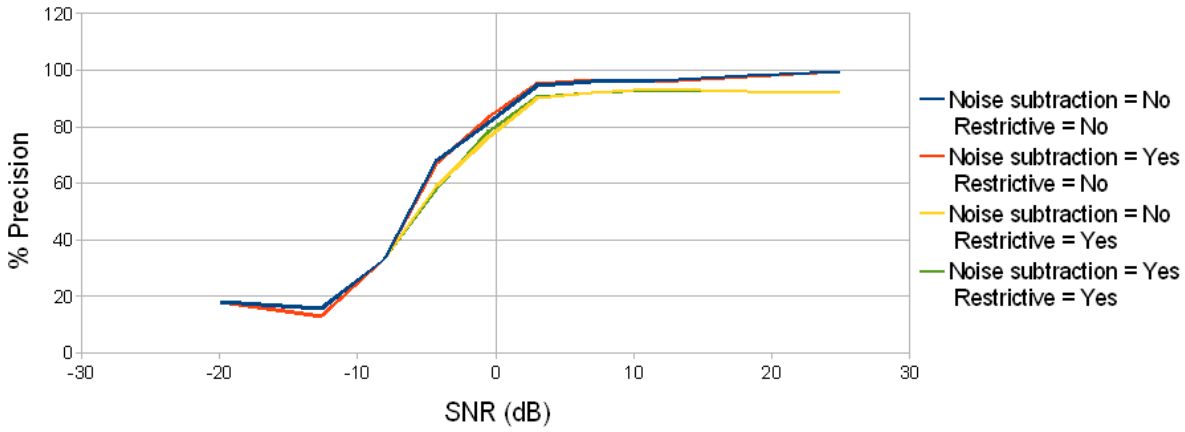

Figure 5.20: *Precision of the Bark class with LOOCV.*

5.2 Testing Real-time

The testing of the system in real-time is made using a sound card, a microphone and a loud speaker (see 4.1.3). For logistical reasons, we lacked spaces with stable conditions of background noise. Tests were made in the Robotics Lab of DEI/O in Padova. The microphone records a quite constant or stationary background noise with constant energy with added audio events with different energies. So the mean SNR can be considered the mean value of the noise. This value characterizes the noise with one value. Unfortunately it is impossible to calculate the accuracy and the precision of the system in real-time for the difficulty of calculating the SNRs. So, in this section, we will give only a qualitative accuracy.

5.2.1 Classification performance

Test were divided depending on the type of the source:

1. loud speaker - that plays the audio tests (The microphone is directed towards the speaker);
2. voice and firecrackers - that generate screams and simulate gunshots.

For the first type of source, the quality of the system remains with good performance having some volume problems because the features are a bit dependent on the frequency and volume. In fact, directing the microphone towards the tweeter of the speaker, the broken glasses class achieved better performance; while a low volume sound, played by the speaker, was not detected.

For the second type of source the classifier performed worse, due mainly to the fact that the database is small.

Furthermore it should be studied the effect of the reverberation in the room and the consequences of it on the system because it caused some problems during the classification step.

5.2.2 Impulse detection performance

The impulse detector accuracy depends directly on the gain and volume of the microphone and consequently to the threshold used to detect the impulse. When installing the microphone, this parameter has to be set to a value that allows the system to detect low volume impulse sounds making some tests in the environment. Once set the “volume” threshold to a value, most of the impulsive sounds that overcome the threshold will be detected because we apply only a simple threshold condition followed by another tight temporal condition that prevents some sporadic isolated peaks of the signal.

Chapter 6

Conclusions and further work

This project achieves a good off-line classification of the environmental sound taken in consideration (gunshots, screams, broken glasses and barking dogs) reaching still 70% of accuracy and 90% of precision at 0 dB SNR, starting from 100% of both accuracy and precision with clean sounds at 20 dB SNR.

During the classification the noise exclusion step can be omitted because it does not improve the performance (e.g. see Figure 5.11). This means that the overhead can be optimized and that our system does not require a training step for training of the environmental noise. This can be a winning feature for a future industrial product that does not require a particular set-up that depends from the environmental background noise.

The detection of an impulsive sound is made with only one simple feature and has a very high precision and accuracy according obviously to the volume threshold used.

Now we suggest some other works and hints that we have not developed for lack of time but we think that they can improve the system:

- to enlarge the database. To achieve a more generic classifier for every audio class there is the need to have a very large database;
- to investigate how much to cut (a formula or a handmade cut ?) the beginning and the tail of the files of the database in order to keep the true "essence" of the audio files of each class;
- feature selection. To improve the classification step, the features that best describe each class should be identified and selected through a more rigorous feature selection stage, using e.g. principal component analysis (PCA) or other statistical methods;
- to find the best local optimum of the GMMs with repeated iterations of the GMM algorithm - both more iterations of the EM step (now they are 300) and more restarts of the algorithm to avoid falling into a local optimum. This also will improve the classification step;
- to classify more sound classes. Adding a new class is very simple (it requires only the construction of the GMM of the class) and does not affect in any way the classifier (it requires only an insertion of an additional case in the possible outputs of the classifier);

- to create a confusion matrix to know how much the models are separated;
- to improve the classification's validation step using k-folds with a larger database;
- to implement an adaptive threshold for the impulse detector as suggested in [12];
- localization of the sound with arrays of microphones; this will provide more info to the surveillance system;
- study of the environmental effects (e.g. reverberation) and possible improvements (e.g. dereverberation).

APPENDIX

Appendix A

Libraries for the installation of NMM

- *a52dec* - AC3 audio decoder;
- *faad* - Decoder for AAC;
- *ffmpeg* - Various audio and video codecs;
- *l1394* - Management of IEEE1394 devices,
- *libmp3lame* - MPEG layer3 audio encoder;
- *libraw1394* - Low level IEEE1394 support;
- *libmad* - MPEG layer3 audio decoder;
- *libdvdnav* - DVD reading and navigation;
- *libdvdread* - DVD reading;
- *libogg* - OGG file parser;
- *libvorbis* - OGG/Vorbis audio decoder;
- *libshout* - Injection of audio streams into a ShoutCast server;
- *fftw* - Fast Fourier Transformation, used for audio visualization;
- *libliveMedia* - RTP support;
- *mpeg2dec* - MPEG version 1/2 video decoder;
- *cdparanoia* - CDDA Support for Linux.
- *libpng* - PNG file reader;
- *asoundlib* - ALSA headers and library;

- *Xlib* - X11 headers and libraries;
- *libjpeg* - JPEG Support;
- *ImageMagick* - Imaging Support;
- *mplayer* - Parts of the implementation of DirectXDisplayNode is based on MPlayer;
- *vlc* - Parts of the implementation of DirectXDisplayNode is based on vlc;
- *transcode* - Parts of the C implementation of deinterlacing algorithm in YUVDeInterlaceNode;
- *ogmtools* - Parts of the implementation of OGMDemuxNode is taken from the ogmtools package;
- *libxmlpp* - Some optional plug-ins and features of NMM use a modified version of libxml++, which is provided along with NMM;
- *libx264* - H264 encoding support. Used by FFMpegEncodeNode;
- *DVB API 5.1* - DVBS2ReadNode depends requires DVB API 5.1, which is part of the Linux kernel headers;
- *ulxmlrpcpp* - XML-RPC library used by example client of TVCaster using XML-RPC API;
- *openssl* - Dependency of ulxmlrpcpp;
- *expat* - Dependency of ulxmlrpcpp.

A.1 Informations on external libraries

Here is the tables' legend:

Name	The original package name of the library
Description	A brief description of the library
Version	The version of the library which works with NMM. NMM may or may not be be compatible with newer versions of the library. If a custom version of the library is required, then this is indicated here.
Homepage	The official project homepage of the external library
Source	Name of the package which contains the full source code of the external library as well as build files required to build the library on all supported platforms. The package is either an original source package of the library or a customized source package which is based on an original source package.
Binary	List of platforms for which precompiled binaries of the library are provided.
Licence	The type of licence under which the external library is available. For custom licences (i.e. licences other than the well-known GNU licences), the full licence text is provided in this documentation.
Build Instructions	Instructions to build the library from provided sources and build files on all the platforms

A.1.1 a52dec

Name	a52dec
Description	AC3 audio decoder
Version	0.7.4
Homepage	http://liba52.sourceforge.net/
Source	tar.gz
Binary	Linux, PS3, Mac OS X, Windows
Licence	GNU GENERAL PUBLIC LICENCE Version 2

Build Instructions:

```
tar xzf a52dec-0.7.4.tar.gz
cd a52dec-0.7.4
./configure --enable-shared --disable-static
make
make install
```

A.1.2 faad

Name	faad2
Description	AAC audio decoder
Version	2.7
Homepage	http://www.audiocoding.com
Source	tar.gz
Binary	Linux
Licence	GNU GENERAL PUBLIC LICENCE Version 2 (Commercial non-GPL licensing possible)

Build Instructions:

```
tar -xvjf faad2-2.7.tar.bz2
cd faad2-2.7
./configure --with-mp4v2
make
mkdir faad_installed
make install DESTDIR=<ABSOLUTE_PATH>/faad_installed
```

A.1.3 ffmpeg

Name	ffmpeg
Description	Various audio and video codecs
Version	0.5
Homepage	http://www.ffmpeg.org
Source	tar.gz
Binary	Linux
Licence	GNU GENERAL PUBLIC LICENSE Version 2, June 1991

Build Instructions:

We assume that libmp3lame is installed in /home/bob. This is required to enable support for MP3 encoding using the ffmpeg library. If you do not need this, you can omit the –enable-libmp3lame configure switch and build ffmpeg without libmp3lame. NMM encoder plug-ins which use ffmpeg will not be able to encode audio in MP3 format. We assume that libx264 is installed in /home/bob. This is required to enable support for H264 encoding using the ffmpeg library.

```
Unpack the source file and set environment variables so that
dependencies are found:

tar xjf ffmpeg-0.5.tar.bz2
export LD_LIBRARY_PATH=/home/bob/lib
cd ffmpeg-0.5

Replace the following line in libavformat/avformat.h

#define MAX_STREAMS 20

by

#define MAX_STREAMS 99

If you need AMR support, build FFMPEG as follows:

cd libavcodec
mkdir amr_float
cd amr_float
unzip ../../../26104-510.zip
unzip 26104-510_ANSI_C_source_code.zip
cd ../../../

./configure --enable-shared --disable-static \
--enable-libmp3lame \
--enable-amr_nb --enable-amr_wb \
--extra-cflags=-I/home/bob/include \
--extra-ldflags=-L/home/bob/lib \
--enable-libx264 --enable-gpl
make
make install

If you do not need AMR support, build FFMPEG as follows:

./configure --enable-shared --disable-static \
--enable-libmp3lame \
--extra-cflags=-I/home/bob/include \
--extra-ldflags=-L/home/bob/lib \
--enable-libx264 --enable-gpl
make
make install
```

A.1.4 I1394

Name	I1394
Description	Management of IEEE1394 devices
Version	0.2.7
Homepage	http://sourceforge.net/projects/l1394/
Source	tar.gz
Binary	Linux
Licence	GNU GENERAL PUBLIC LICENCE Version 2

Build Instructions:

We assume that libraw1394 is installed, and header files are available in the standard include paths. If libraw1394 is installed in a non-standard location, you can use the –with-raw1394 switch when running configure.

```
tar xzf l1394-0.2.7.tar.gz
cd l1394-0.2.7
./configure --enable-shared --disable-static
make -k
make -k install
```

A.1.5 libmp3lame

Name	libmp3lame
Description	MPEG layer3 audio encoder
Version	3.96.1
Homepage	http://lame.sourceforge.net
Source	tar.gz
Binary	Linux, Windows, PS3, Mac OS X
Licence	GNU GENERAL PUBLIC LICENCE Version 2

Build Instructions:

```
tar xzf lame-3.96.1.tar.gz
cd lame-3.96.1
./configure --enable-shared --disable-static
make
make install
```

A.1.6 libraw1394

Name	libraw1394
Description	Low level IEEE1394 support
Version	as provided by SuSE 11.0
Homepage	http://sourceforge.net/projects/libraw1394
Binary	version abailable no
Licence	GNU LESSER GENERAL PUBLIC LICENCE Version 2.1

A.1.7 libmad

Name	libmad
Description	MPEG layer3 audio decoder
Version	0.15.1b
Homepage	http://www.underbit.com/products/mad/
Source	tar.gz
Binary	Linux, Windows, PS3, Mac OS X
Licence	GNU GENERAL PUBLIC LICENCE Version 2

Build Instructions:

```
tar xzf libmad-0.15.1b.tar.gz
cd libmad-0.15.1b/
./configure --enable-shared --disable-static
make
make install
```

A.1.8 libdvdnav

Name	libdvdnav
Description	DVD reading and navigation
Version	0.10.1
Homepage	http://dvd.sourceforge.net/
Source	tar.gz
Binary	Linux, Windows, PS3, Mac OS X
Licence	GNU GENERAL PUBLIC LICENCE Version 2

Build Instructions:

```
tar xzf libdvdnav-0.1.10.tar.gz
cd libdvdnav-0.1.10
./configure --enable-shared --disable-static
make
make install
```

A.1.9 libdvdread

Name	libdvdread
Description	DVD reading
Version	0.9.4
Homepage	http://www.dtek.chalmers.se/groups/dvd/
Source	tar.gz
Binary	Linux
Licence	GNU GENERAL PUBLIC LICENCE Version 2

Build Instructions:

```
tar xzf libdvdread-0.9.4.tar.gz
cd libdvdread-0.9.4/
./configure --enable-shared --disable-static
make
make install
```

A.1.10 libogg

Name	libogg
Description	OGG file parser
Version	1.1
Homepage	http://xiph.org/ogg/
Source	tar.gz
Binary	Linux
Licence	GNU GENERAL PUBLIC LICENCE Version 2

Build Instructions:

```
tar xzf libogg-1.1.tar.gz
cd libogg-1.1
```

```
./configure --enable-shared --disable-static
make
make install
```

A.1.11 libvorbis

Name	libvorbis
Description	OGG/Vorbis audio decoder
Version	1.0.1
Homepage	http://xiph.org/vorbis/
Source	tar.gz
Binary	Linux
Licence	GNU GENERAL PUBLIC LICENCE Version 2

Build Instructions:

```
tar xzf libvorbis-1.0.1.tar.gz
cd libvorbis-1.0.1
./configure --enable-shared --disable-static --with-ogg=/home/bob
make -k
make -k install
```

A.1.12 libshout

Name	libshout
Description	Injection of audio streams into a ShoutCast server
Version	2.0
Homepage	http://www.icecast.org/
Source	tar.gz
Binary	Linux
Licence	GNU GENERAL PUBLIC LICENCE Version 2

Build Instructions:

```
tar xzf libshout-2.0.tar.gz
cd libshout-2.0
./configure --enable-shared --disable-static --with-ogg=/home/bob
--with-vorbis=/home/bob
make
make install
```

A.1.13 fftw

Name	fftw
Description	Fast Fourier Transformation, used for audio visualization
Version	2.1.5
Homepage	http://www.fftw.org/
Source	tar.gz
Binary	Linux
Licence	GNU GENERAL PUBLIC LICENCE Version 2

Build Instructions:

```
tar xzf fftw-2.1.5.tar.gz
cd fftw-2.1.5
./configure --enable-shared --disable-static
make
make install
```

A.1.14 libliveMedia

Name	libliveMedia
Description	RTP support
Version	2009.06.02 with modifications by Motama
Homepage	http://live555.com/liveMedia/
Source	tar.gz
Binary	Linux, Windows
Licence	GNU LESSER GENERAL PUBLIC LICENCE Version 2.1

Build Instructions:

```
# Unpack
tar xzf live.2009.06.02.tar.gz
cd live

# Add -DUSE_SYSTEM_RANDOM to COMPILE_OPTS in config.linux (to avoid segfault)

# Build static libraries
./genMakefiles linux
make

# Build shared library
gcc -shared -o libliveMedia.so.2009.06.02 */*.a
ln -s libliveMedia.so.2009.06.02 libliveMedia.so
```

```
# install to \$INSTALLDIR
find -name \*.hh -exec cp "{}" \$INSTALLDIR/include/live ";"
find -name \*.h -exec cp "{}" \$INSTALLDIR/include/live ";"
cp libliveMedia.so.2009.06.02 libliveMedia.so \$INSTALLDIR/lib

Build Instructions (Windows):

Open Microsoft Visual Studio 2005.

Unpack the file live.2005.07.13.tar.gz.

Open the solution file liveMedia.sln from the source package.

In the combo box for selecting the build configuration, pick Release.

Choose "Build Solution" from the "Build" menu

Copy the DLL file from the Release directory to a directory that is
listed in your PATH environment variable.
This is required for running NMM-based applications that use this library.

Copy the LIB file from the Release directory to a directory that is listed as
additional library directory in your Visual Studio project files that use this
external library.
(This is only required for development of applications that use this
library directly.
You may not need to do this at all.)
```

A.1.15 mpeg2dec

Name	mpeg2dec
Description	MPEG version 1/2 video decoder
Version	0.4.0b
Homepage	http://libmpeg2.sourceforge.net/
Source	tar.gz
Binary	Linux, Windows, Mac OsX
Licence	GNU GENERAL PUBLIC LICENSE Version 2

Build Instructions:

```
tar xzf mpeg2dec-0.4.0b.tar.gz
cd mpeg2dec-0.4.0/
./configure --enable-shared --disable-static
make
make install
```

A.1.16 cdparanoia

Name	cdparanoia
Description	CDDA Support for Linux
Version	IIIalpha9.8-548
Homepage	hhttp://xiph.org/paranoia/
Source	tar.gz
Binary	Linux, Windows, PS3

A.1.17 libpng

Name	libpng
Description	PNG file reader
Version	1.0.13
Homepage	http://www.libpng.org/
Source	http://xiph.org/paranoia/down.html
Binary	no
Licence	GNU GENERAL PUBLIC LICENCE Version 2

Build Instructions:

```
# building
tar xzf libpng-1.2.19.tar.gz
cd libpng-1.2.19
./configure
make

# installing to $DESTDIR
cp libpng12.so* $DESTDIR/lib
cp libpng.so* $DESTDIR/lib
cp png.h $DESTDIR/include
```

A.1.18 asoundlib

Name	asoundlib
Description	ALSA headers and library
Version	1.0.9a
Homepage	http://www.alsa-project.org
Binary	no
Licence	GNU Lesser General Public License Version 2.1

A.1.19 Xlib

Name	Xlib
Description	X11 headers and libraries
Binary	no

A.1.20 libjpeg

Name	libjpeg
Description	JPEG support
Version	6.2.0-738
Homepage	http://www.ijg.org/
Source	http://www.ijg.org/files/
Binary	PS3

A.1.21 ImageMagick

Name	ImageMagick
Description	Imaging Support
Version	6.4.0.4 (as provided with OpenSuSE 11.0)
Homepage	http://www.imagemagick.org
Source	tar.bz2
Binary	Linux

Build Instructions:

```
tar xvfz ImageMagick-6.3.5-5.tar.gz
cd ImageMagick-6.3.5/
./configure
make
make install
```

A.1.22 ImageMagick for Windows

Name	ImageMagick for Windows
Description	Imaging Support
Version	6.4.6
Homepage	http://www.imagemagick.org
Source	ImageMagick-windows.zip
Binary	Windows
Licence	http://www.imagemagick.org/script/license.php

Build Instructions:

```
1) Run Visual Studio 2005
2) Open Solution ImageMagick-6.4.7/VisualMagick/configure/configure.sln
3) Click "Finish" in Conversion Wizard
4) Build Solution
5) Run Solution (CTRL+F5)
6) Press Next...Next...Finish in VisualMagick configuration tool
7) Open solution VisualMagick/VisualDynamicMT.sln
8) Click "Finish" in Conversion Wizard
9) Build Solution
```

A.1.23 mplayer

Name	mplayer
Description	Parts of the implementation of DirectXDisplayNode is based on MPlayer
Version	unkown
Homepage	http://www.mplayerhq.hu/
Source	Included in nmm/plugins/display/DirectX/DirectXDisplayNode.hpp and .cpp
Binary	Will be built from nmm/plugins/display/DirectX/DirectXDisplayNode.hpp and .cpp
Licence	GPL

A.1.24 vlc

Name	vlc
Description	Parts of the implementation of DirectXDisplayNode is based on vlc
Version	unkown
Homepage	http://www.videolan.org/
Source	Included in nmm/plugins/display/DirectX/DirectXDisplayNode.hpp and .cpp
Binary	Will be built from nmm/plugins/display/DirectX/DirectXDisplayNode.hpp and .cpp
Licence	GPL

A.1.25 transcode

Name	transcode
Description	Parts of the C implementation of deinterlacing algorithm in YUVDeInterlaceNode
Version	unkown
Homepage	http://www.transcoding.org
Source	Included in nmm/plugins/video/filter/yv12_deinterlace.h and .c
Binary	Will be built from nmm/plugins/video/filter/yv12_deinterlace.h and .c
Licence	GPL

A.1.26 ogmtools

Name	ogmtools
Description	Parts of the implementation of OGMDemuxNode is taken from the ogmtools package
Version	unkown
Homepage	http://www.bunkus.org/videotools/ogmtools/
Source	nmm/plugins/av/ogm/ogmstreams.h
Binary	Will be built from sources in nmm/plugins/av/ogm
Licence	GPL

A.1.27 libxml++

Name	libxml++
Description	Some optional plug-ins and features of NMM use a modified version of libxml++, which is provided along with NMM.
Version	unkown
Homepage	http://libxmlplusplus.sourceforge.net/
Source	nmm/utils/xml
Binary	Will be built from sources in nmm/utils/xml
Licence	LGPL

A.1.28 libx264

Name	libx264
Description	H264 video encoder
Version	x264-snapshot-20090401-2245
Homepage	http://www.videolan.org/developers/x264.html
Source	tar.bz2
Binary	Linux
Licence	GPL

Build Instructions:

```
NOTE: Use tcshell (use of bash causes problems with
./configure and make of libx264)

tar xjvf x264-snapshot-20090401-2245.tar.bz2
cd x264-snapshot-20090401-2245
./configure --enable-shared --disable-asm
make
make install
```

A.1.29 DVB API 5.1

Name	DVB API 5.1
Description	Linux DVB API
Version	5.1
Homepage	http://www.linuxtv.org/
Headers	Not provided (part of the Linux kernel headers)
Binary	not provided (part of the Linux kernel)
Licence	GPL

A.1.30 ulxmlrpcpp

Name	ulxmlrpcpp
Version	1.7.4
Homepage	http://ulxmlrpcpp.sourceforge.net
Binary	Linux
Licence	GNU LIBRARY GENERAL PUBLIC LICENSE Version 2

```
Note: You have to use --prefix option to specify install location.
Option $DESTDIR does not work.

tar xfj ulxmlrpcpp-1.7.4-src.tar.bz2
cd ulxmlrpcpp-1.7.4
./configure --with-openssl --prefix=$HOME/ulxmlrpcpp_install
make install
```

A.1.31 openssl

Name	openssl
Version	as provided with SuSE 11.0
Homepage	http://www.openssl.org
Binary	provided with SuSE 11.0

A.1.32 expat

Name	expat
Description	The Expat XML Parser
Version	as provided with SuSE 11.0
Homepage	http://expat.sourceforge.net/
Binary	provided with SuSE 11.0

Bibliography

[1] G. Valenzise, L. Gerosa, M. Tagliasacchi, F. Antonacci, and A. Sarti. Scream and gunshot detection and localization for audio-surveillance systems. In *Advanced Video and Signal Based Surveillance, 2007. AVSS 2007. IEEE Conference on*, pages 21 –26, sep. 2007.

[2] G. Valenzise, L. Gerosa, M. Tagliasacchi, F. Antonacci, and A. Sarti. Scream and gunshot detection in noisy environments. Dipartimento di Elettronica e Informazione - Politecnico di Milano.

[3] Marco Cristani, Manuele Bicego, and Vittorio Murino. Audio-visual event recognition in surveillance video sequences. In *Multimedia, IEEE Transactions on*, volume 9, pages 257 –267, feb. 2007.

[4] J.-L. Rouas, J. Louradour, and S. Ambellouis. Audio events detection in public transport vehicle. In *Intelligent Transportation Systems Conference, 2006. ITSC '06. IEEE*, pages 733 –738, sep. 2006.

[5] C. Clavel, T. Ehrette, and G. Richard. Events detection for an audio-based surveillance system. In *Multimedia and Expo, 2005. ICME 2005. IEEE International Conference on*, pages 1306 –1309, jul. 2005.

[6] A. Dufaux, L. Besacier, M. Ansorge, and F. Pellandini. Automatic sound detection and recognition for noisy environment. In *European Signal Processing Conference (EUSIPCO), Tampere, Finlande*, sep. 2000.

[7] V. Peltonen, J. Tuomi, A. Klapuri, J. Huopaniemi, and T. Sorsa. Computational auditory scene recognition. In *Acoustics, Speech, and Signal Processing, 2002. Proceedings. (ICASSP '02). IEEE International Conference on*, volume 2, pages 1941 –1944, 2002.

[8] G. Tzanetakis and P. Cook. Musical genre classification of audio signals. In *Speech and Audio Processing, IEEE Transactions on*, volume 10, pages 293 – 302, jul. 2002.

[9] Rui Cai, Lie Lu, Hong-Jiang Zhang, and Lian-Hong Cai. Highlight sound effects detection in audio stream. In *Multimedia and Expo, 2003. ICME '03. Proceedings. 2003 International Conference on*, volume 3, pages III – 37–40 vol.3, jul. 2003.

[10] Hoiem D, Yan Ke, and R. Sukthankar. Solar: sound object localization and retrieval in complex audio environments. In *Acoustics, Speech, and Signal Processing, 2005. Proceedings. (ICASSP '05). IEEE International Conference on*, volume 5, pages v/429 – v/432 Vol. 5, mar. 2005.

[11] P.K. Atrey, N.C. Maddage, and M.S. Kankanhalli. Audio based event detection for multimedia surveillance. In *Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on*, volume 5, pages V –V, may. 2006.

[12] James F. Kaiser. On a simple algorithm to calculate the 'energy' of a signal. In *Proceedings ICASSP90*, pages 381–384, 1990.

[13] Geoffroy Peeters. A large set of audio features for sound description (similarity and classification) in the cuidado project. April 2004.

[14] Stanley Smith Stevens, John Volkman, and Edwin Newman. A scale for the measurement of the psychological magnitude of pitch. In *Journal of the Acoustical Society of America*, volume 8 (3), page 185190, 1937.

[15] Douglas O'Shaughnessy. *Speech communication: human and machine*. Addison-Wesley p. 150, 1987.

[16] Jacob Benesty, M. M. Sondhi, and Yiteng Huang. *Springer Handbook of Speech Processing*. Springer, 2008.

[17] Lie Lu, Hong-Jiang Zhang, and Hao Jiang. Content analysis for audio classification and segmentation. In *Speech and Audio Processing, IEEE Transactions on*, volume 10, pages 504 – 516, oct. 2002.

[18] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. Wiley India Pvt. Ltd., second edition, 2007.

[19] The Freesound Project. http://www.freesound.org/.

[20] ChartDirector. http://www.advsofteng.com/.

[21] Motama. Nmm site - http://www.motama.com/nmm.html.