UNIVERSITÀ DEGLI STUDI DI PARMA

CORSO DI DOTTORATO DI RICERCA IN INGEGNERIA INDUSTRIALE

XXVII CICLO

# DEVELOPMENT OF A SOFTWARE FOR PANORAMIC VIDEO AND AUDIO RECORDING WITH MICROPHONE ARRAYS

Coordinatore:

Chiar.mo Prof. AGOSTINO GAMBAROTTA

Tutor:

Chiar.mo Prof. ANGELO FARINA

Dottorando:

Dott. Ing. SIMONE CAMPANINI

a *Giovanna Emanuela*

*e*

*ad Ivo ed Elsa*

# Contents

# List of Figures

# Chapter 1

# Introduction

Multichannel recordings are usually performed by means of microphone arrays. In many cases *sparse* microphone arrays are used, where each discrete microphone is employed for capturing one of the channels.
However, also the usage of *dense* microphone arrays has a long history, dating back to the first *MS-matrixed* microphones setups and passing through the whole Ambisonics saga.
A dense microphone array is employed differently from a sparse array, generally speaking each channel is obtained by a combination of the signals coming from all the capsules, by means of different matrixing and filtering approaches.
During last years at the *Industrial Engineering Department* of *Università degli Studi di Parma* (IED) with the collaboration of the spinoff AIDA srl., a new meta-theory was developed that provides a framework capable of representing all the known microphones array, based on the model of *Virtual Microphones.*
The strict collaboration with the *Centro Ricerca ed Innovazione Tecnologica* (CRIT, Research Center and Technology Innovation) of RAI, the national Italian television broadcasting company, quickly individuates some key applications - mainly in live or off-line broadcast contest - of this technology that conduct to the realization of some *ad-hoc* dense arrays and of a software to manage all the acquisition and post-processing system, that is the subject of this PhD thesis work.

## 1.1 Why microphone arrays?

The use of microphone arrays is the basis for making audio recordings (or acoustical measurements) capturing information about the spatial distribution of the wavefronts impinging onto a listener.

In a sparse microphone array the microphones are placed at large distances, with the goal to sample the sound field at points where the sound is significantly different.

On the other hand, in a dense array, the microphones are usually small and close each other, so that the minimum distance between two microphones is significantly smaller than the wavelength for any allowed direction of arrival of the wavefront The intended usage of the signals captured by the two types of microphone arrays is completely different: in a sparse array, the signal coming from each capsule has to be kept well separated from all the others, and is tailored to become the feed for a single loudspeaker in the playback system. The only adjustments possible for the sound engineer is to apply to each signal some amount of gain, some spectral equalization, and perhaps some delay. But each signal is always processed independently from the others. The only exception to this rule is the so-called *mixdown*, which occurs whenever the playback system is equipped with a number of loudspeakers which is smaller than the number of recorded channels: in this case the channel count is reduced by simple mixing rules.

In a dense array, instead, the whole set of signals are treated as an unique entity, and the processing required for computing the output signals (speaker feeds) involves generally complex matrixing operations, so that each output channel includes some amount of information captured by all the microphones. Furthermore, heavy filtering is usually required, instead of the simple gain and delay adjustments employed for sparse arrays.

This means that, inevitably, the dense array technology requires more effort: all the microphones must be reliable and high quality,[1] the processing algorithm requires much more computational power,[2] and the risk to get artifacts is vastly larger.

[1] A noisy capsule affects the behaviour of the whole system

[2] As each output signal requires to apply specific filtering to all the input channels, and each filter is generally computationally very heavy.

Whilst the sparse array approach is perfectly compatible with a minimalistic, *audiophile* philosophy, avoiding almost completely dangerous digital processing effects, the dense array approach cannot work without massive and powerful filtering and matrixing.
These factors explain the comparatively larger success encountered till now by the sparse array technology, and the limited success of dense microphone arrays. However, regarding stereo (2-channels) recordings, the prominent *dense array* approach has been the M-S technique,[3] which conquered favoritism for some applications in the film-video-tv industry; mostly thanks to its excellent *mono compatibility*, obtained simply by discarding the $S$ signal, when not needed, and by the versatility of adjusting the amount of *ambience*, obtained changing the relative gain of the $M$ and $S$ signals. Generally speaking, however, most high-quality stereo recordings make use of sparse arrays, employing typically two spaced microphones, such as in the ORTF method[4] or in the A-B method.[5] Going to *surround* recordings for horizontal sound systems (quad, 5.1, 7.1, etc.), again we have seen a modest success of one particular type of dense microphone arrays, the 1st order Ambisonics method [2]. This makes use of just 3 channels ($W$, $X$, $Y$) for capturing the spatial information in the horizontal plane, with the possibly to scale up to full periphony adding a 4th channel ($Z$). Different types of dense microphone arrays have been developed for surround recording, which can be subdivided in three categories:

- Tetrahedrical probes (Soundfield™ microphone and similar)
- Multiple M-S probes (Schoeps™)
- *native* B-format arrays (Nimbus™)

The latter, in theory, already provides the required $W$, $X$ and $Y$ signals as the outputs of the three capsules (one omni capsule for $W$, and two figure-of-eight capsules for $X$ and $Y$), but in practice some amount of post- processing is always required, for ensuring perfect gain and phase matching among the three channels.

[3]See [6].
[4]Modestly spaced capsules, approximately 170 mm.
[5]Here the capsules can be several meters apart

So all of these microphone arrays require some processing for delivering the canonical B-format signals. For years the processing has been analog, but nowadays it is better performed digitally, and hence these microphone arrays are already embraced by the meta-theory described in chapter 3.
Of course, also for surround recordings, the vast majority are performed with *sparse* microphone arrays. Also these can subdivided in two main categories:

**Distributed arrays** several microphones spread in the orchestra and picking up the voices of the vocalists, followed by amplitude-panning over the surround channels

**Compact arrays** for example Williams', INA- 5, OCT, etc.

In this case the most successful approach was the first, giving full control to the sound engineer to place each soundtrack at will on the surround panorama by panning.
The second approach, indeed, is the preferred one for minimalist/audiophile surround recording, as the signal of each capsule is directly the feed for the corresponding loudspeaker, without the need of any further processing. But this comes at the price of requiring an ideal placement of the microphone array, inside a venue with excellent acoustics, and making it impossible to fix anything after the recording is done. And finally in the last years novel *massive* recording/playback techniques emerged, capable of driving advanced sound systems equipped with dozens or hundredths of loudspeakers, and employing advanced digital sound processing methods.[6] Here the channel count increases dramatically, ranging from 16 (3rd order HOA) up to several hundredths (large WFS systems). In this field, dense arrays are usually preferred, making it possible to pack a large number of capsules in a single, compact and light object, which can be conveniently manipulated and placed.
These dense arrays were built of various shapes, with or without a *solid object* over which the capsules are mounted, and in many different shapes: line, planar, circle, sphere, pseudo-random *cloud*, and recently cylinders. The system usually contains also the digital interface, ensuring that all the signals are sampled with

[6]WFS - Wave Field Synthesis [3], HOA - High Order Ambisonics [5], SPS - Spatial PCM Sampling [8], etc.

perfectly matched gain and phase, and the same digital sampling clock. The massive processing required is performed digitally on a standard PC, which nowadays provides enough computational power even for very large numbers of capsules, thanks to the raw computing capabilities of modern CPUs and the refinement of advanced processing algorithms, such as the FFT[7] and the partitioned convolution.[8]

However, some *sparse* microphone arrays did survive also for these massive multichannel applications, such as the bulky microphone system developed by the Japanese television broadcast company NHK in cooperation with the McGill University for the 22.2 three-dimensional surround system.[9]

At IED a single, unified meta-theory was developed for describing the processing performed in any kind of dense microphone array. The approach is perfectly accurate whenever the processing being performed is perfectly linear and time invariant, as it is the case for many classical approaches.

Whenever a not linear approach is employed, such as the *steering* methods based on analysis of the acoustical three-dimensional scene and consequent continuous change of the filters,[10] the approach adopted is still useful for representing each temporary *state* of the variable processing system. This remains a rigorous approach whenever the processing is performed over chunks of data, provided that an independent evaluation of the system's filtering is performed for each chunk. The approach becomes just approximate, indeed, whenever the processing employs continuously-changing filters: this would require to recompute a complete digital filtering network for each data point of the sampled sound.

[7] A fast, open source implementation is described in [10].

[8] An implementation example in [21].

[9] See [23].

[10] As in the Dirac and Harpex-B algorithms [20], [1].

# Chapter 2

# Microphone arrays: the state of the art

Having already introduced what is a microphone array and why this kind of probes have captured the attention for many years, in the following sections a panoramic, divided by field of application, of the systems currently available on the market by various manufacturers will be given, preceded by a description of the so far universally adopted technique of *beam forming*.
As conclusion, it is reported a short description of the most interesting arrays developed at IED during the years 2011-2014.

## 2.1 The beam forming technique

With set of $M$ real microphone capsules, it is possible to synthesize one or more *virtual microphones*, with a certain orientation and order, by the application of the *beam forming* technique, widely employed in the electromagnetic field, but it can be applied with some modifications in the acoustic field, too.

With reference of figure 2.1, representing a plane wave that invests a linear microphone array, if $\lambda$ is the wavelength of the incident sound, it is possible to state that a certain wave front, taken as reference, will reach the left capsule with a delay $\Delta t$ respect the time instant in which the same front has invested the center capsule.

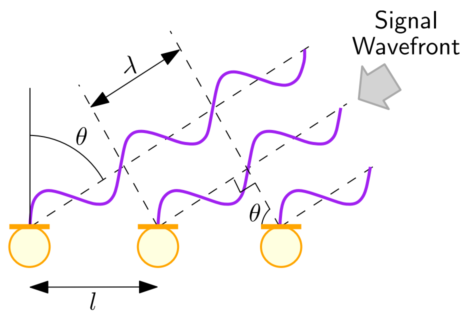


**Figure 2.1:** Schematic representation of a plane wave incident on a linear microphone array.

Let $l$ the distance between nearest capsules and $\theta$ the incident angle respect the normal to the plane on with the capsules lie, this delay can be expressed in the following way:

$$\Delta t(\theta) = T\frac{l}{\lambda}\sin\theta. \tag{2.1}$$

It should be now clear that summing the signals coming from the $M$ capsules with a progressive advance from right to left multiple of $\Delta t(\theta)$, it will be possible to obtain a sort of *virtual orientation* of the synthesized microphone,[1] the so called *steering*. In equations:

$$y_\theta(t) = \frac{1}{M}\sum_{k=0}^{M-1} x_k\left(t + k\Delta t(\theta)\right). \tag{2.2}$$

The desired effect is to obtain the maximum phase cancellation for waves coming from a direction different from $\theta$, and, conversely, to leave the maximum amplitude to waves coming from $\theta$ direction.

This theory is obviously extensible to planar and three-dimensional arrays, but in any case the only modifications introduced to captured signals are on amplitude and phase.

[1]Given by the sum of $M$ signals.

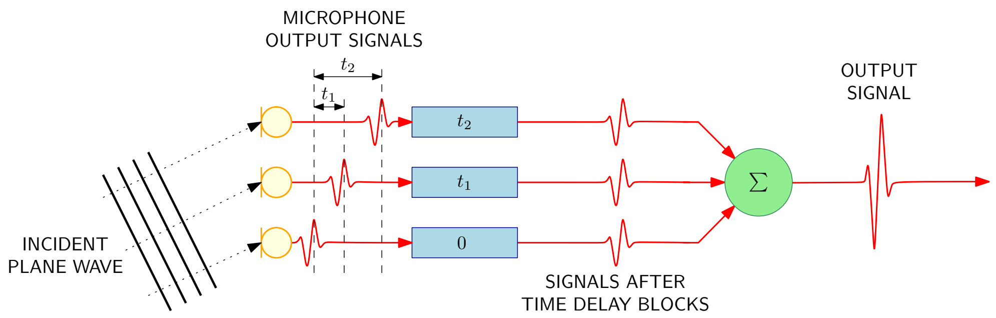


**Figure 2.2:** *Steering* of the synthesized virtual microphone.

### 2.1.1 Limits of the beam forming theory

In the previous paragraph, two important hypothesis were made to simplify the understanding:

1. incident waves don't suffer any kind of attenuation;
2. incident front wave is *planar*, that is the array is placed in *far field* respect the sound source.[2]

Also the *spatial aliasing* phenomenon was ignored, because the sound field is *sampled* in the space: the spatial counterpart of Shannon sampling theorem states that the minimum wavelength the array can sample correctly is given by:

$$\lambda \sin\theta > 2l, \tag{2.3}$$

that becomes $\lambda > 2l$ in the worst case.

Finally, the equation 2.2 was written ignoring the uniqueness or not of the solution found for every $\theta$: let the focus direction $\theta = 0$, in this case the equation 2.2 become

$$y_\theta(t) = \frac{1}{M}\sum_{k=0}^{M-1} x_k(t); \tag{2.4}$$

it can be noted that incoming signals $x_k(t)$ can be written, referencing to picture 2.1, as a function of the wave arriving to the capsule on the right

[2]This implies that the distance between sound source and array is almost equal to the major dimension of the array.

$$\begin{aligned} x_k(t) &= x\left(t - \frac{klT}{\lambda}\sin\theta\right) \\ &= x\left(t - \frac{kl}{c}\sin\theta\right), \end{aligned} \tag{2.5}$$

and, combining the results of equations 2.4 and 2.5,

$$y_\theta(t) = \frac{1}{M}\sum_{k=0}^{M-1}\left(t - \frac{kl}{c}\sin\theta\right). \tag{2.6}$$

Passing to frequency domain and by the application of the sum and time shift properties, it can be easily obtained the transfer function module, expressed in the equation :

$$Y_\theta(f) = \frac{1}{M}X(f)\cdot\sum_{k=0}^{M-1}\mathrm{e}^{j\omega\frac{kl}{c}\sin\theta} \tag{2.7}$$

$$\begin{aligned} |H|_{\mathrm{dB}}(f) &= 20\log\left|\frac{Y_\theta(f)}{X(f)}\right| \\ &= 20\log\left|\sum_{k=0}^{M-1}\mathrm{e}^{j\omega\frac{kl}{c}\sin\theta}\right| \qquad [\mathrm{dB}]. \end{aligned} \tag{2.8}$$

In picture 2.3 is shown the equation 2.8 in normalized form on polar diagram in the specific case $M = 4$, $l = 0.2$ m, $f = 1$ kHz and $c \sim 343$ m/s, varying the microphone capsules number; this picture illustrates that the virtual orientation $\theta = 0$ of the array has not prevented the generation of secondary lobes on remarkable amplitude. Also, it can be noted that, fixed the sampling distance $1/l$, the acoustic focus - hence the array *spatial* resolution - increases together with the employed capsules number, while it decreases with the frequency.[3]

### 2.1.2 Advanced beam forming techniques

As already written, the number of application fields for microphone arrays is very large, but some limits of the traditionally adopted beam forming theory can

[3]Concept are well and concisely expressed in [12], where are also present some C code examples, but for a more deep and scientifically precise analysis, see [4].

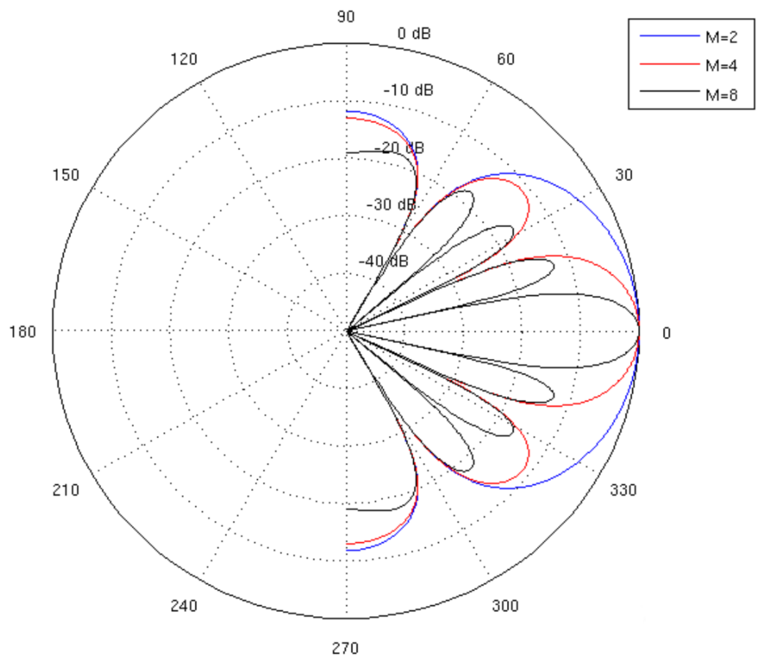


**Figure 2.3:** Normalized beam patterns by varying the microphone capsules number.

discourage the use of this kind of probes in place of usual, well experienced technology; for example, the noise analysis inside vehicles, requires to consider a wide frequency range, because the various kinds of stresses that normally affecting a vehicle, and beam forming technique cannot have a large frequency range in such small environments, as equation 2.3 states.

Some of the research efforts to surpass traditional beam forming theory limits are aimed to develop *combined* techniques that is where another one is employed in the application domains where beam forming is weak. An example for the noise analysis field is the Spatial Transformation of the Sound Field (STSF[4]) that employs Near-field Acoustic Holography (NAH[5]) and its more recent modification, the Statistically Optimized NAH (SONAH[6]), that, being able to do a more accurate analysis of the near field, it has been proved they are good and efficient *partners* of beam forming.[7]

Basically, NAH is a technique that allows the study of spatial and frequency (or

[4]See [13].
[5]See [18].
[6]See [15].
[7]See [14].

time) dependence of acoustic fields. Rather than measure all points in 3D space directly, NAH is used to reconstruct the quantities of acoustic field above and below the measured hologram plane, which means that the data-acquisition process to be considerably shortened.
The theory of NAH is based on the Helmholtz integral equation and the two-dimensional spatial Fourier transform. Complex pressure on a plane parallel to the hologram plane is computed by inverse 2D FFT of product of hologram pressure spectra and modified Green's functions. The acoustic velocity vector is then acquired using Euler's equation, active and reactive intensity vector as real and imaginary parts of product of computed complex pressure and conjugate acoustic velocity.
The STSF was developed for *stationary* sound fields: the technique was then extended to Non-Stationary STSF, becoming able to apply time domain averaging techniques, like, for example, the transient analysis.
To be noted that some of the theories listed, like NAH, were known since 1985, but the CPU computation power needed made them unenforceable at the time of their formulations.
Another source locations technique, this time as an alternative to beam forming, is the Inverse Boundary Elements Method, that connects together the Inverse Method and, as the name said, the Boundary Elements Method; while the latter is a well-known numerical technique for solving general radiation problems, the first is a method to solve the *acoustic source problems* that consist of the determination of source descriptors - e.g. the surface vibration on a virtual surface wrapping the object under investigation - from a knowledge of the radiated acoustic pressure as measured by a microphone array located around the source. The major advantage of IBEM is the possible estimation of the vibration velocity and the sound intensity right on the surface of a totally irregularly shaped source. Therefore, this allows a very precise source location and a very precise noise source ranking.[8]

[8] For a more complete review of spatial array techniques, see [11].

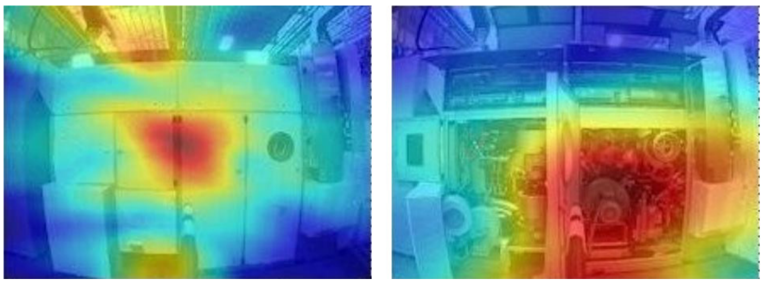


**Figure 2.4:** An example of two noise maps overlapped to the machinery analyzed.

## 2.2 Array for noise maps elaborations

An important field of application for microphone arrays is certainly the environmental acoustic, in particular, these probes are often employed as *acoustic cameras* giving to the user a visual map of the pressure level of the space to which the array is pointed. Usually this map is made partially transparent and overlapped with a photo of the environment analyzed, so it's really straightforward the localization of noise sources, as can be seen in figure 2.4.

Some commercially available microphone arrays are especially made for environmental application and described in the following sections.

### 2.2.1 Brüel & Kjær

Brüel & Kjær produces some arrays dedicated to different environmental applications; after research on microphone dispositions to find optimized random distributions that correspond to *array pattern* minimization, the *Wheel Array* represents a compromise solution between the difficulty setting the freedom degrees of the random configuration and the complexity of the physical structure requested by a too random disposition.
The *Wheel Array* is composed of an odd number of line arrays inclined by a fixed angle respect the wheel spokes, each of them has the same random configuration, optimized with the array pattern. The array mechanical design is entirely modular: every spoke can be removed from the support rings - one in the interior and one in the exterior - and it's possible to reassemble themselves in different ways.

In figure 2.5 on the left, the version with 90 microphones can be seen: 15 identical spokes with 6 microphones with a total diameter of 2.43 m; it hasn't an

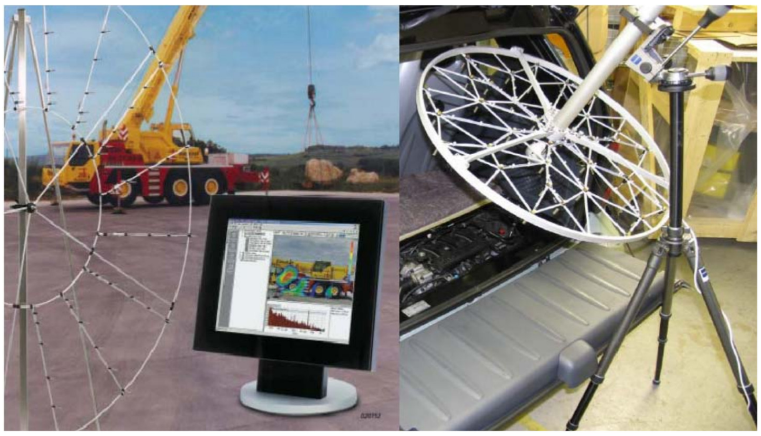


**Figure 2.5:** The Brüel & Kjær *Wheel Array*.

embedded video camera, but the software allow the overlap and the alignment of an optical image, captured separately by the user, with the acoustic one, finding two common points in the two images. In the same picture, on the right, it's shown a smaller array, with 66 microphones, 11 spokes and diameter of, circa, 1 m; it's provided of an embedded camera and it's more suited for acquisitions in the near field: the array diameter determines the main lobe amplitude and, accordingly, the angular resolution.
The package includes the dedicated acquisition hardware, PULSE™, that offers some interesting utilities for multichannel calibration, ensures the synchronization between channels and a high quality A/D conversion.
A software, also named PULSE™, is provided: it can produce both static and dynamic - the *acoustic movie* - maps. It's important to mention that Brüel & Kjær is the only array manufacturer that has released public documentation describing the algorithm employed,[9] that is basically a beam forming with a particular gains set,[10] and it's relevant to point out that, regardless the microphone calibration, the system cannot output absolute pressure levels because it's impossible to quantify the spatial noise in a map point.

An acoustical map produced by Brüel & Kjær's PULSE™ and measured with the *Wheel Array* is reported in figure 2.6: coloured spots appears only in corre-

[9]See [4].

[10]The documentation reports how these gains has been obtained.

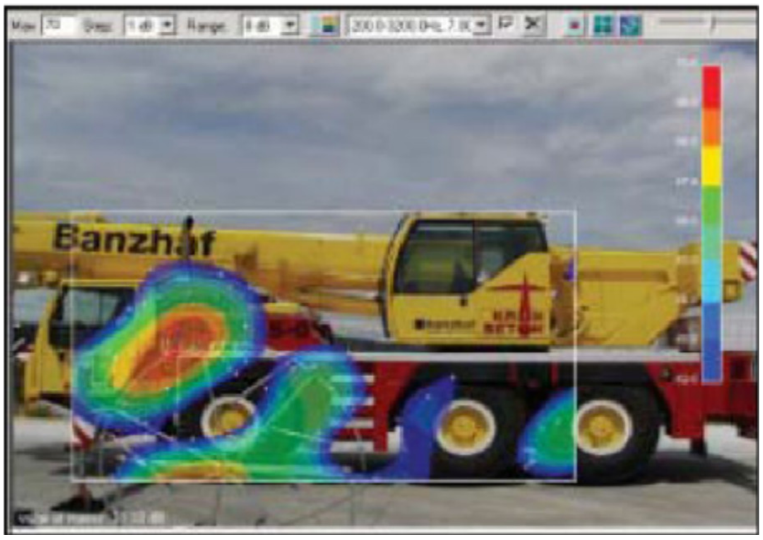


**Figure 2.6:** Example of Brüel & Kjær's PULSE™ acoustic map.

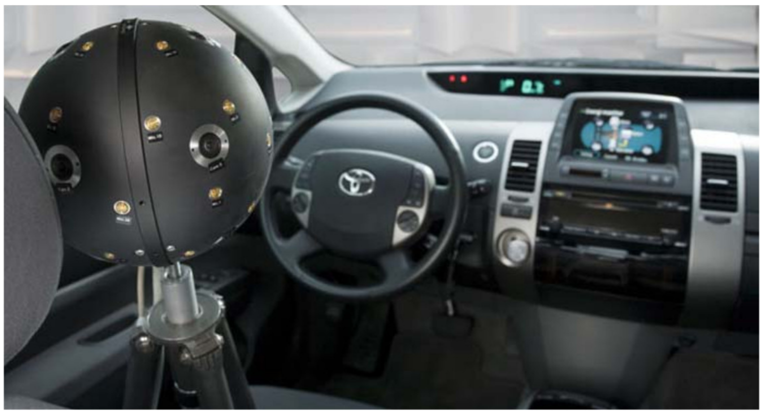


**Figure 2.7:** Brüel & Kjær spherical beam-former.

spondence with the noise sources, this mean that the software shows noise levels only above a certain threshold, but it's unknown if automatic or user-defined. Another interesting model from the danish company is a spherical array (figure 2.7) with 12 video cameras on the surface, available in two versions, with 36 or 50 microphones. The data are acquired placing the sphere in the center of the measuring area; the algorithm employed is the spherical beam forming, a technology that allows an omni-directional mapping independent from the acoustic environment.

Brüel & Kjær is also the developer of the SONAH algorithm, that, combining

NAH with beam forming, tries to conciliate the capabilities of both technologies: calibrated intensity, pressure and velocity maps of the particles near the source from the first,[11] and high performance with irregular disposition from the latter. In particular, SONAH is used in the evaluation of holography and the scaling of the sound intensity outputted by the beam forming algorithm, when collecting measures with an ad-hoc designed array, the *Sector Wheel Array*, that maintains the rotational symmetry of the *Wheel Array*, but is composed by identical sectors containing 12 elements disposed following an irregular pattern to optimize the Maximum Side lobe Level.

### 2.2.2 gfai tech

*Gfai tech* is the German company that developed and produced *Acoustic Camera*, the first microphone array for acoustic visualization appeared on the market. This producer is experienced and engaged in research in many branches of media technology and offers various models of arrays, each equipped with camera.

The first reported model is the *Ring Array*, visible in figure 2.8, it is a measure system designed for applications in acoustics laboratories, suitable for near field acquisitions, it is available in versions with 32, 36, 48 and 72 microphones, with different sizes and characteristics, for example, *Ring 36* has a diameter of 0.35 m and a mapping frequency range between 1.0 to 20 kHz, but the *Ring 48*, with a diameter of 0.75 m, maps frequencies between 400 Hz to 20 kHz; in every model the ring is made in carbon fiber.

In figure 2.9 is shown the *Star Array*: it is composed of three spokes, the maximum opening amplitude is 3.40 m, and is suitable for the far field, that means long distances (from 3 to 300 m) and low frequencies (from 100 Hz to 7 kHz). The system can be folded and moved quickly, so it's easy to carry it. The non planar structure is patented and it guarantees the maximum attenuation for signals arriving from behind that is a requirement for measures in disturbed environments.

[11] This implies a uniform measure grid, with spacing inferior to the half of the wave length, that covers completely the sound source.

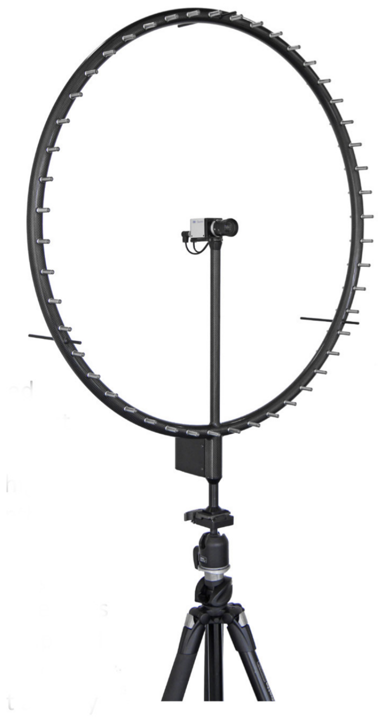


**Figure 2.8:** gfai tech *Ring 48* (size: $\varnothing$ 0.75 m, Measured range: $0.7 \div 5$ m, SPL$_{\max}$: 130 dB, $f_{\text{range}}$: $400 \div 20$k Hz).

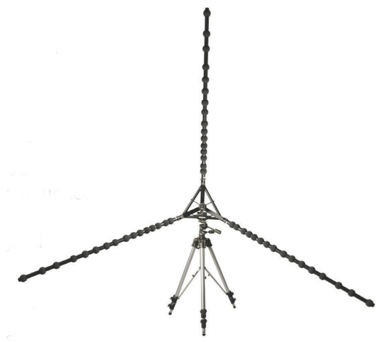


**Figure 2.9:** gfai tech *Star 48*(size: max span 3.4 m, Measured range: $3 \div 300$ m, SPL$_{\max}$: 130 dB, $f_{\text{range}}$: $100 \div 7$k Hz).

For measures of large objects and special applications like wind gallery, gfai tech proposes the *Spiral Array* (figure 2.10) that has excellent acoustic and dynamic ranges.

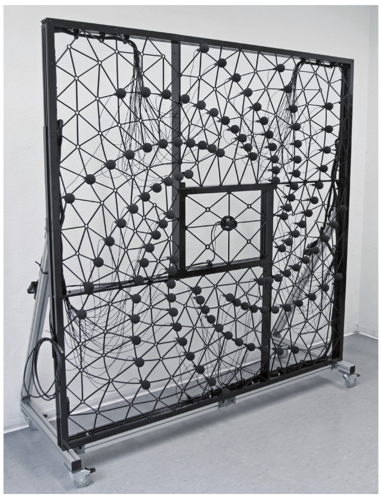


**Figure 2.10:** gfai tech *Spiral 120* (size: $2 \times 2$ m, Measured range: $3 \div 150$ m, SPL$_{\max}$: 130 dB, $f_{\text{range}}$: $200 \div 20\text{k}$ Hz).

For closed environment applications, where sound is typically more or less diffused, so a good resolution on all solid angle is requested, the *Sphere Arrays* (figure 2.11) are suggested, suitable for measures on all the space around, they are built for very small spaces, also vehicle cabins, and high frequencies. As the *Ring* model, the sphere array is available with 48 or 120 microphones, with different sizes and characteristics: the first has a diameter of 0.035 m and a frequency range between 1 kHz to 10 kHz, the latter has a diameter of 0.6 m and measures between 400 Hz to 10 kHz. The structure is in carbon fiber and guarantees the maximum acoustic transparency, even the design minimizes aliasing and reflection effects.

Another options offered from gfai tech is that of microphones freely configurable (figure 2.12) presented as the integration of gfai tech beam forming hardware and NAH software from SenSound LLC (USA).

As can be seen in figure 2.13, like Brüel & Kjær PULSE™, the *Acoustic Camera* software from gfai tech, plots colours only in correspondence of the sources, so when it's capable to reject lateral diffraction and aliasing lobes: it is not known if this is done simply using a noise threshold, or applying more complex processing algorithms because no description of the techniques used is offered. *Acoustic*

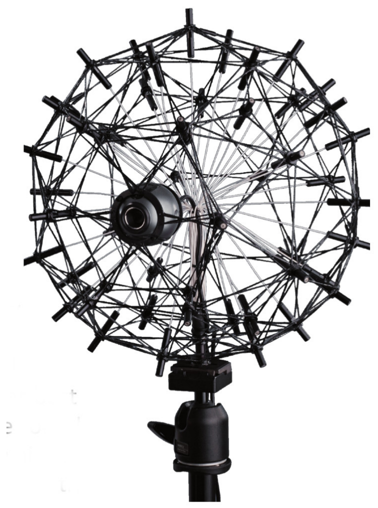


**Figure 2.11:** gfai tech *Sphere 48* (size: $\varnothing$ 0.35 m, Measured range: $0.3 \div 1.5$ m, $SPL_{max}$: 130 dB, $f_{range}$: $1 \div 10$ kHz).

**Figure 2.12:** gfai tech's free configurable 24 channel microphone bundle (10-10k Hz).

*Camera* operates in post-processing, but allows a real-time preview with limited screen resolution that probably uses a simplified algorithm.

### 2.2.3 Nittobo

The Japanese manufacturer Nittobo produces a system called *Noise Vision*$^{TM}$ (figure 2.14) that consists in a spherical transducer provided with 31 microphones and 12 video cameras (weights: 12 kg) and a notebook computer with software; the system allow the generation of an acoustic map in every direction referred to the spherical transducer. The map representation can be both static, to analyze stationary phenomenon, and animated, to analyze phenomena from non-stationary or moving sources. The declared frequency range is $200 \div 5000$ Hz.

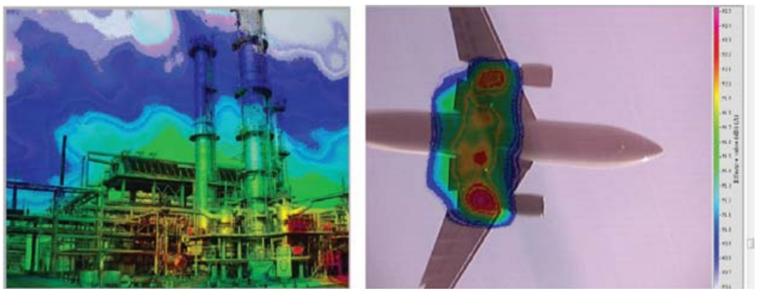


**Figure 2.13:** Examples of acoustic maps produced by gfai tech *Acoustic Camera*.

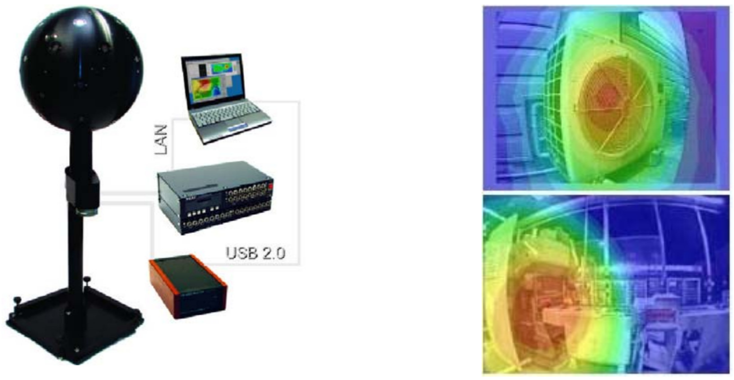


**Figure 2.14:** Nittobo's *Noise Vision*™ system.

The manufacturer doesn't release specific technical documentation, but only very generic informations.

## 2.3 Directive microphoning

Another particularly interesting field of applications for microphones arrays is the so called *directive microphoning* which shares working principles and theory with the system presented in the previous chapter, but for which some manufacturers realized, in some cases very specific products, in some other more generic ones.

### 2.3.1 Acoustic Magic

The *Voice Tracker*™ Array Microphone of Acoustic Magic contains an array if 8 elements and doesn't present moving parts. It has a rugged structure with

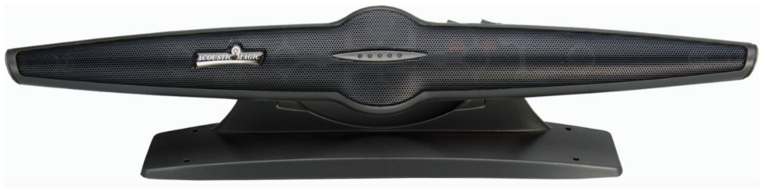


**Figure 2.15:** Acoustic Magic Voice Tracker™ I Array Microphone.

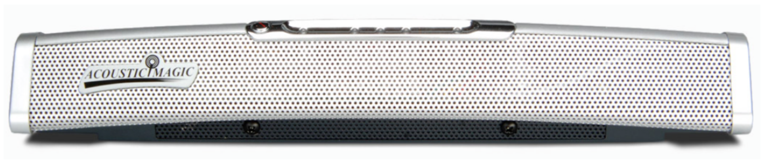


**Figure 2.16:** Acoustic Magic Voice Tracker™ II Array Microphone.

reduced weight and is easy to use: no specific software is requested, it is not very demanding for computer CPU and is compatible with different operating systems. It measures 45.7 cm (width) by 6.3 cm (height) and an image is shown in figure 2.15, where it can be noticed the five optical indicators for chosen speaker position and signal level.

In the product documentation it is asserted that the system can localize automatically the active speaker and that it can orient electronically the *listening beam* around 360°, moving from a speaker to the neighbor in an amount of time in the order of milliseconds. The speaker is also free to turn or move. The declared range is about 9 m for meeting recording and some meters for automatic speaking detecting; frequency response comes from 100 Hz to 11250 Hz.
In addition to focusing on the speaker, the Voice Tracker™ system realize a spatial filtering on noise coming from other directions, but there are no explications on how this is done. The manufacturer mentions proprietary algorithms for noise reduction, capable to filter background noise and environment reverberation, in particular, always without explications, the *Location Dependent Source* (LDS) technology is cited that can desensitize microphones for sounds coming from preselected directions.
The manufacturer has also released the version II of Voice Tracker™, that features 6 microphones instead of 8, an Acoustic Echo cancellation filter, a *quantitative* talker location capability, power through USB interface and more compact di-

mensions.

### 2.3.2 Yamaha

A representative model, suitable for medium-small environments is the PJP-50R *Conference Microphone System*, visible in figure 2.17; the name stands for *ProjectPhone*: it is a videoconference system with 16 microphones 4 loudspeakers, connected to, alternatively, a Local Area Network or, an analog phone line or an audio device with analog line in/out connections.[12] It is suitable for four speakers placed approximately around the device. The ring shaped microphone array can detect automatically the sound source locations, by the analysis of signal retards, but it is possible to specify some directions to exclude from the analysis in advance, like known noise sources (for example, blowers). No details about the employed algorithms on the documentation, but it is mentioned an automatic system for echo cancellation that corrects filter coefficients accordingly with the environment.
The loudspeakers reproduce the voice clearly with a volume sufficiently high per large rooms, in addition there is the possibility (*divide mode*) to reproduce different voices with different speakers, accordingly to the speakers' position, so it can be easy to detect its placement.

A more interesting model from Yamaha was the PJP-100UH (figure 2.18), that featured 32 microphones, 12 loudspeakers and could serve up to eight speakers, four for every side of the table where the device was typically placed,[13] but now this product has been discontinued and substituted with a modular solution, the YVC-1000 (figure 2.19), a system formed by a main unit to which are connected one or more microphone-speaker units, each of them with 3 microphone capsules installed: it seems that Yamaha, instead of developing a more complex technology for a single dense microphone arrays, has preferred to rely to distributed, simpler, devices, minimizing efforts and speeding up results.

[12] It exists also a USB version, but equipped with 8 microphones and 4 loudspeakers.

[13] With the optional camera PJP-CAM1, the system could manage a video signal too: it could choose and bounds the image from the area in which active speaker was located.

**Figure 2.17:** The Yamaha PJP-50R Conference Microphone System.

**Figure 2.18:** The discontinued Yamaha PJP-100UH Conference Microphone System with 16+16 microphones on long sides.

### 2.3.3 Squarehead Technology

The *AudioScope* (figure 2.20) represents on large scale what can be realized with a planar array. It is a stand-alone audio system designed for sport and live entertainment shows substantially composed by three parts: the first is a strongly directive array, realized with a 2 m carbon fiber disc where 315 microphones are arranged in a regular pattern and with a video camera at the center; this is placed above the area to shoot and it can cover a surface of the size of a basketball court. Through a communication device, the array communicates with one or more control centers from which the operator manages the system simply moving the cursor on the image reproduced on the monitor: in particular a trackball let to follow the moving sources and with a knob it is possible to rewind the recording for the *replays*. The system allows the selection of five audio sources at the same

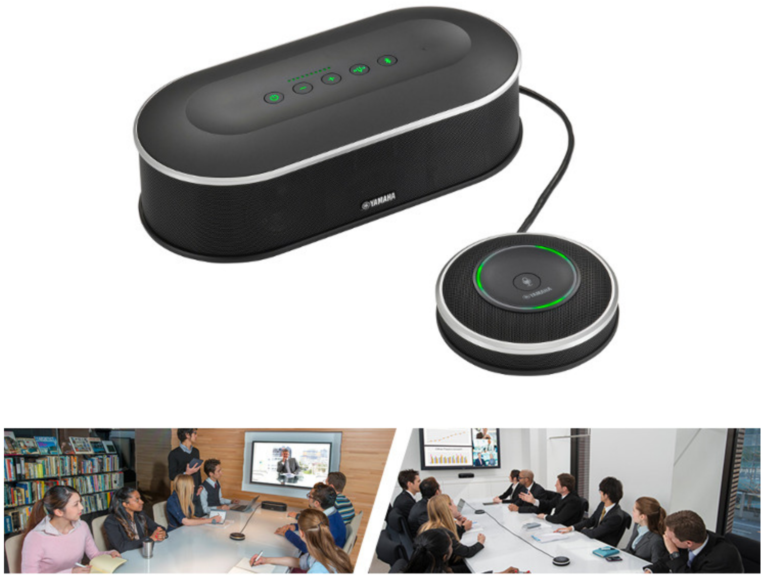


**Figure 2.19:** The Yamaha YVC-1000.

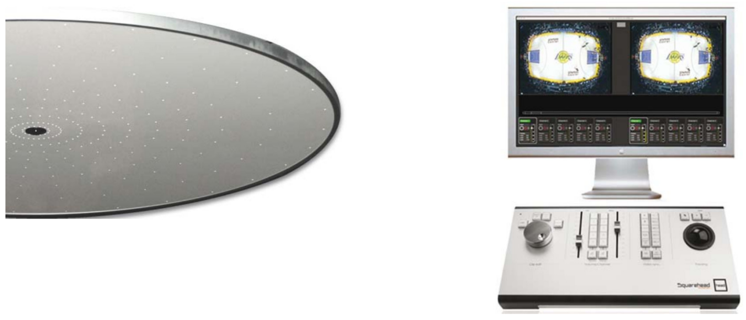


**Figure 2.20:** The Squarehead Technology's *Audio Scope* system: on the left the array, on the right the control station.

time.

### 2.3.4 mh acoustics

The microphone array *Eigenmike®* *EM32* (figure 2.21)of American company *mh acoustic* is since some years at the center of a collaboration between the IED/AIDA and CRIT.

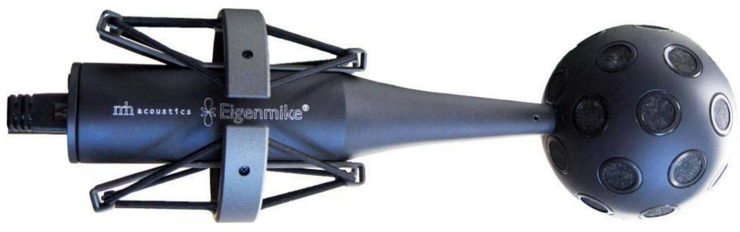


**Figure 2.21:** mh acoustic Eigenmike® EM32.

The microphone consists of an anodized aluminum sphere, with 43 mm radius, where 32 high quality electret microphones capsules are arranged on the surface following a relatively uniform pattern. In the sphere interior it has been placed the entire analog preamplifying circuitry, with software controlled gain, and the analog to digital converters that can sample capsules signals with 24 bit of resolution and 48 kHz of sampling rate. The signals are then transferred to the *Eigenmike® Microphone Interface Box* (EMIB) with a fully digital connection, using a common CAT-5 network cable that guarantees disturb immunity also for long distances between probe and interface (up to 140 m).
The EMIB is connected to a computer through FireWire® interface, behaving like a normal audio interface; drivers are provided for OS X® and Windows®, but it works also on GNU/Linux systems using Ffado drivers.
The manufacturer bundles with the microphone a software, the *EigenStudio*,[14] that employs the beam forming technique for sound sources pointing.
This device, starting from the same principles, is quite different from the previous listed, because it offers at the same time, high audio quality, due to the Sennheiser® capsules, very compact dimensions and a smart connection to the dedicated hardware, using only one small cable: these features have made the Eigenmike® an ideal candidate for live audio broadcast applications, and is thanks to this device that the virtual microphone technique, described in the next chapters, has been developed.

[14]Available only for Microsoft®Windows® platform.

## 2.3.5 Customs

The requests made to the IED by the broadcast company RAI have conducted to the development of other Eigenmike®-based arrays,[15] more suitable for some very specific applications.

### 2.3.5.1 Planar Array

The original Eigenmike® EM32 is a spherical probe and it can resolve sound sources placed above, below and behind its placement, but in some real cases there's no needing to point virtual microphones in these directions, for example when shooting a show on a stage from a frontal point of view: what happens in other directions than the frontal one is a disturb, and more spatial resolution is needed on the front side. A shape that can satisfies these specification is the planar one, with capsules arranged with semi-random distances between them so there aren't privileged wavelength. To get a pointing visual reference, a video camera - completely missing in the Eigenmike® - has been placed in the center of the array. The realization is shown in figure 2.22.

### 2.3.5.2 Cylindrical Array

Similar specifications to which that led to the realization of the IED Planar Array, gave the birth of the Cylindrical Array[16]: this time the scene has to be shot 360° around the probe, but not above or below, so the ideal shape is a cylinder, always with the capsules placed following a semi-random pattern. Since in live recording applications, together with the audio is always needed a video reference for virtual microphone pointing a camera capable of 360° shooting, described in chapter 4, has to be placed on the top of the device, as can be seen in the figure 2.23.

[15]In these realizations, all the electronic of the original Eigenmike® EM32 has been used, but placing the microphone capsules in different patterns.

[16]Thanks to the hard work of Eng. Lorenzo Ebri and Eng. Lorenzo Chiesi.

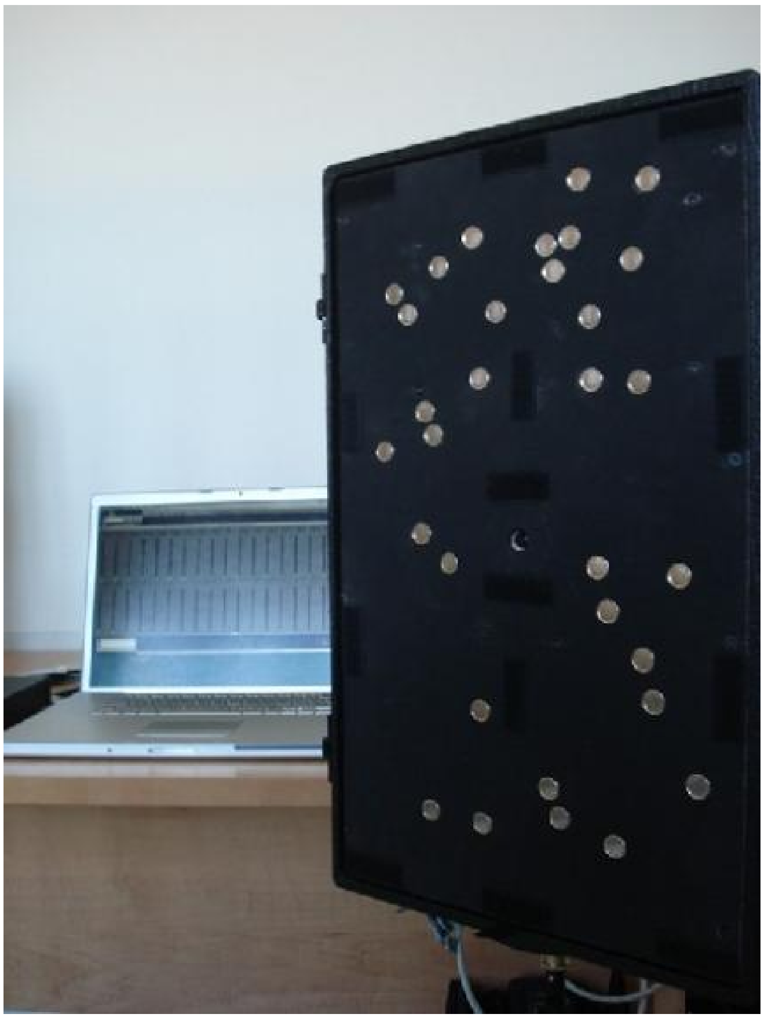


**Figure 2.22:** The IED Planar array with the control laptop.

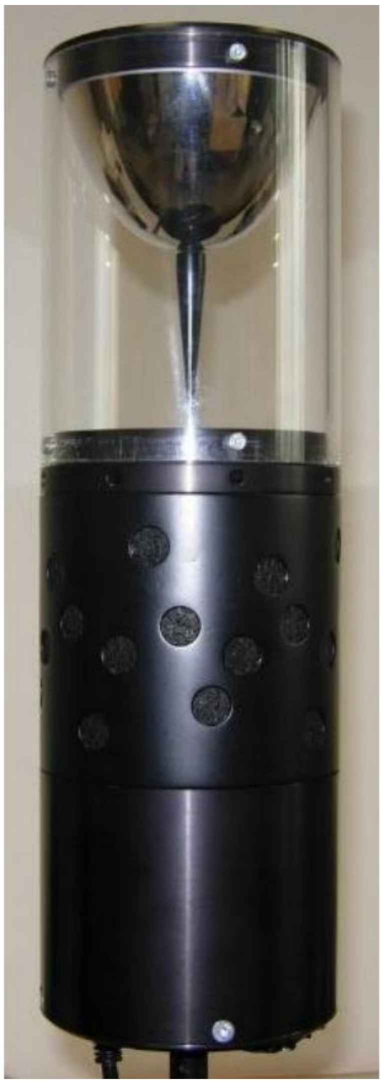


**Figure 2.23:** The IED Cilindrical array.

# Chapter 3

# Virtual microphones

In recent years a lot of research has been produced about technologies for capturing and reproducing the spatial properties of the sound. Most of the proposed approaches employ massive arrays of microphones and loudspeakers, and process the signals by means of very complex mathematical theories, based on various modifications of the classical Huygens principle. These methods include decomposition of the sound field as a superposition of plane waves, working in cartesian coordinates,[1] or as a superposition of spherical harmonics, working in spherical coordinates.[2]

Recently, even more complex methods have been proposed by Nelson and Fazi.[3]

Whatever method is employed, it is possible always think to the whole processing as the synthesis of a number of *virtual microphones*, each of them feeding a loudspeaker in the playback system.

Having realized that, we decided to remove the constraints inherent with any of the previously-known techniques, and to generate directly the desired virtual microphones as a direct transformation of the *raw* recorded signals, without relying on an intermediate layer (or kernel) of *basic* waveforms.

[1]For example *Wave Field Synthesis* [2].

[2]*High Order Ambisonics* [19].

[3]Decomposing the sound field in complex Hankel functions [9].

Albeit this approach can work, in principle, with any geometrical layout of the microphone array, we decided to develop our system around an high-quality 32-capsules spherical microphone array, the Eigenmike® EM32 made by *mh acoustic* fully described in section 2.3.4.

The 32 signals are filtered employing a massive convolution processor, capable of real-time synthesis and steering of up to 7 virtual directive microphones, controlling their aiming and capture angle by means of a mouse/trackpad, and employing a wide-angle panoramic video camera and a graphical *view & point* interface for an easy-to-operate user's interface.

This can be done in real-time and with small latency during a live broadcasting event; alternatively, the raw signals from the 32 capsules can be recorded, together with the panoramic video, for subsequent synthesis of the virtual microphone signals in post-production.

## 3.1 A *theoryless* approach

The *Virtual Microphone System* (VMS) developed at IED is not based on any of the previously known theories, such as *High Order Ambisonics*, *WFS* and similar. There is no intermediate spatial kernel, such as a set of spherical harmonics or Hankel functions.

The virtual microphones being synthesized can be highly directive,[4] and are intrinsically coincident, so the signals can be mixed without concerns of comb filtering; it is possible to continuously move their aiming for following actors or singers on scene, or for giving instantaneous miking to people in the audience. Surround recording of a concert is just one of the possible scenarios for employing this approach, which has also been successfully tested for dramas, sport events, and TV shows in which there is systematic interaction of conductors and guests with in-studio audience.

[4]The polar pattern is, in fact, constant with frequency, and with a beam width much sharper than a *shotgun* microphone.

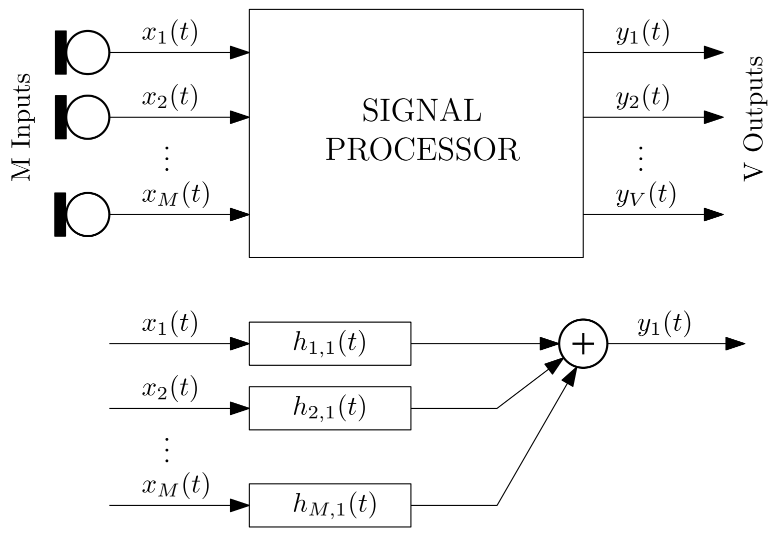


**Figure 3.1:** Scheme of signal processing.

A careful analysis of the performances of the new microphone system did show that frequency response, signal-to-noise ratio and rejection of off-beam sounds are better than those obtainable employing traditional processing algorithms applied to the same input signals, or dedicated top-grade ultra-directive microphones.

### 3.1.1 Computation of the filters

Given an array of transducers, a set of digital filters can be employed for creating the output signals. In our case the $M$ signals coming from the capsules need to be converted in $V$ signals yielding the desired virtual directive microphones: so we need a bank of $M \times V$ filters. For a couple of reasons, FIR filters are preferred.

Assuming $x_m$ as the input signals of $M$ microphones, $y_v$v as the output signals of $V$ virtual microphones and $h_{m,v}$ the matrix of filters, the processed signals can be expressed as:

$$y_v(t) = \sum_{m=1}^{M} x_m(t) * h_{m,v}(t) \tag{3.1}$$

Where $*$ denotes convolution, and hence each virtual microphone signal is obtained summing the results of the convolutions of the $M$ inputs with a set of $M$ proper FIR filters.

One of the most used techniques for deriving filter banks for generating virtual microphones with arbitrary directivity is the *Ambisonics* method: first the $M$ signals are processed deriving an equal or smaller number of spherical harmonics. Later these spherical harmonics signals are added together with proper gains, for synthesizing the desired virtual microphones. This has the advantage of allowing for the derivation of a large number of virtual microphones with a small additional effort, as most of the effort is represented by the computation of the spherical harmonic signals.

Furthermore, it is possible to change dynamically the aiming or the directivity of every virtual microphone simply changing the gains employed when adding the signals together.

In our approach, instead, every desired virtual microphone is derived directly from the original $M$ signals, avoiding the *Ambisonics* encoding and decoding: the outputs of the processing system are directly the result of the digital filtering of the input signals, with a different set of filters for every virtual microphone.
In principle this allows for synthesizing virtual microphones having an arbitrary directivity pattern. In practice we decided, for now, to synthesize frequency-independent high-order cardioid virtual microphones.
The directivity factor $Q$ of a virtual cardioid microphone of order n is described, in spherical coordinates $\theta$, $\phi$ , by the expression:

$$Q_n(\theta,\phi) = [Q_1(\theta,\phi)]^n \tag{3.2}$$

Where $Q_1(\theta,\phi)$ is the directivity factor of a first order cardioid microphone:

$$Q_1(\theta,\phi) = 0.5 + 0.5 \cdot \cos(\theta) \cdot \cos(\phi) \tag{3.3}$$

The processing filters $\mathbf{h}$ are usually computed following one of several complex mathematical theories, based on the solution of the wave equation, often under certain simplifications, assuming the microphones are ideal and identical. In some implementations the signal of each microphone is processed through a digital filter for compensating its deviation, with a heavier computational load.

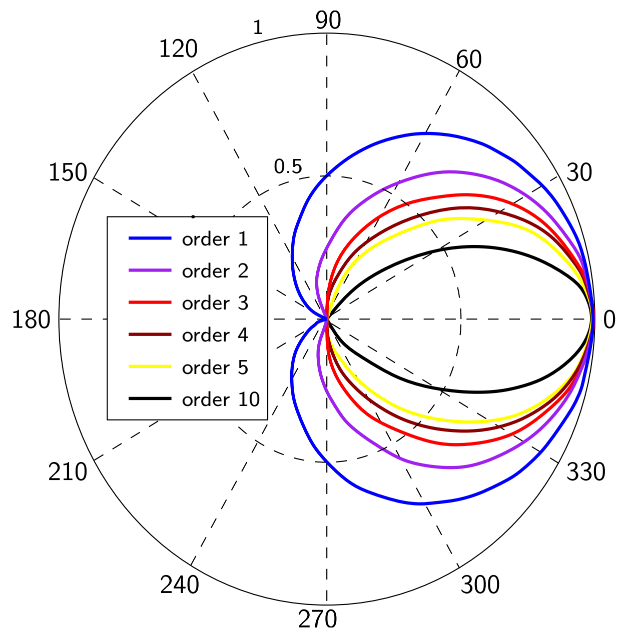


**Figure 3.2:** Polar plots of virtual cardioid microphones of various orders (target patterns $Q_n$).

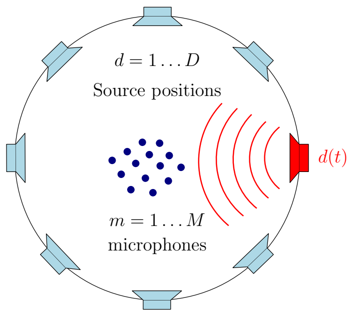
In this novel approach no theory is assumed: the set of filters **h** are derived directly from a set of measurements, made inside an anechoic room. A matrix of measured impulse response coefficients **c** is formed and the matrix has to be numerically inverted[5]; in this way the outputs of the microphone array are maximally close to the ideal responses prescribed. This method also inherently corrects for transducer deviations and acoustical artefacts (shielding, diffraction, reflection, etc.).

The characterization of the array is based on a matrix of measured anechoic impulse responses, obtained with the sound source placed at a large number $D$ of positions all around the probe, as shown in figure 3.3.

The processing filters **h** should transform the measured impulse responses **c** into the prescribed theoretical impulse responses **p**:

$$p_d = \sum_{m=1}^{M} c_{m,d} * h_m \qquad d = 1 \dots D \tag{3.4}$$

[5]usually employing some approximate techniques, such as Least Squares plus regularization



$$\mathbf{C} = \begin{pmatrix} c_{1,1} & c_{1,2} & \dots & c_{1,d} & \dots & c_{1,D} \\ c_{2,1} & c_{2,2} & \dots & c_{2,d} & \dots & c_{2,D} \\ \vdots & \vdots & \dots & \vdots & \dots & \vdots \\ c_{m,1} & c_{m,2} & \dots & c_{m,d} & \dots & c_{m,D} \\ \vdots & \vdots & \dots & \vdots & \dots & \vdots \\ c_{M,1} & c_{M,2} & \dots & c_{M,d} & \dots & c_{M,D} \end{pmatrix}$$

**Figure 3.3:** Impulse response measurements from $D$ source positions to the $M$ microphones.

Please notice that in practice the target impulse responses pd are simply obtained applying a direction-dependent gain $Q$, given by equation 3.2, to a delayed unit-amplitude Dirac's delta function $\delta$: $p_d = Q_d \cdot \delta$.

Computation is easier in frequency domain.[6] Let's call **C**, **H** and **P** the resulting complex spectra. This way, the convolution reduces to simple multiplication between the corresponding spectral lines, performed at every frequency index $k$:

$$P_d = \sum_{m=1}^{M} C_{m,d,k} \cdot Hm, k \qquad \begin{cases} d &= 1 \dots D \\ k &= 0 \dots N/2. \end{cases} \tag{3.5}$$

Now we pack the values of **C**, **H** and **P** in proper matrices, taking into account all the $M$ input microphones, all the measured directions $D$ and all the $V$ outputs to create:

$$[H_k]_{M\times V} = \frac{[P]_{M\times V}}{[C_k]_{D\times M}} \tag{3.6}$$

This over-determined system doesn't admit an exact solution, but it is possible to find an approximated solution with the least squares method, employing a regularization technique for avoiding instabilities and excessive signal boost[7]. The block diagram of the least-squares method is shown in figure 3.4.

In this scheme we observe the delay block $\delta$, required for producing causal filters, and the resulting total modelling error $e$, which is being minimized by the

[6]That is, computing the complex spectra, by applying the FFT algorithm to the $N$-points-long impulse esponses **c**, **h** and **p**

[7]See [16] and [17].

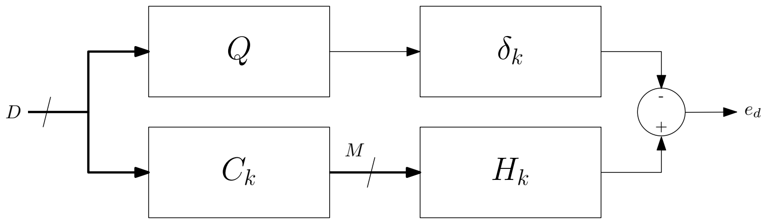


**Figure 3.4:** Scheme of the Least Squared method with a delay in the upper branch.

least-squares approach.

In general, the frequency-domain representation of a Dirac's delta delayed by $n_0$ samples is given by:

$$\delta_k = \mathrm{e}^{-j2\pi k\frac{n_0}{N}} \tag{3.7}$$

Albeit various theories have been proposed for defining the optimal value of the causalisation delay $n_0$, we did take the easy approach, setting $n_0 = N/2$. Choosing $N/2$ samples is a safe choice, which creates inverse filters with their *main peak* close to their centre, and going smoothly to zero at both ends.

Furthermore, a regularization parameter is required in the denominator of the matrix computation formula, to avoid excessive emphasis at frequencies where the signal is very low.

So the solution formula, which was first proposed in [17], becomes:

$$[H_k]_{M\times V} = \frac{[C_k]^*_{M\times D}\cdot[Q]_{D\times V}\cdot \mathrm{e}^{-j\pi k}}{[C_k]^*_{M\times D}\cdot[C_k]_{D\times M} + \beta_k\cdot[I]_{M\times M}} \tag{3.8}$$

As shown in picture 3.5, the regularization parameter $\beta$ should depend on frequency. A common choice for the spectral shape of the regularization parameter is to specify it as a small, constant value inside the frequency range where the probe is designed to work optimally, and as much larger values at very low and very high frequencies, where conditioning problems are prone to cause numerical instability of the solution.

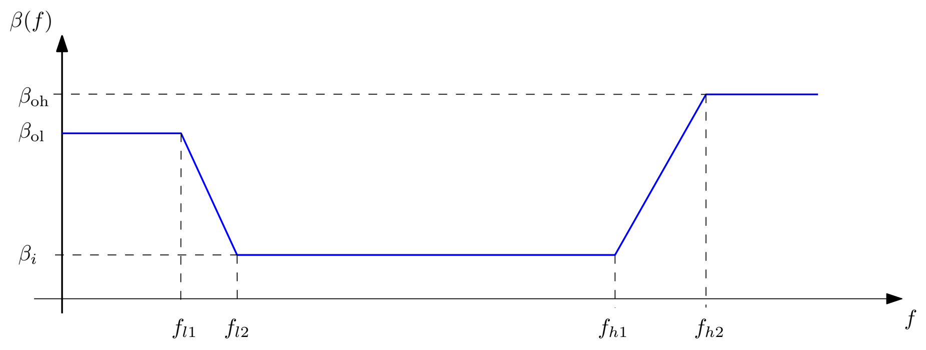


**Figure 3.5:** Regularization parameter in dependence of the frequency.

### 3.1.2 Experimental characterization of the array

Measurements of the microphone array were made employing the Exponential Sine Sweep (ESS) method, in order to obtain 32 Impulse Responses for each direction of arrival of the test signal.

The ESS method was chosen due to its capability of removing unwanted artifacts due to nonlinearities in the loudspeaker, and because it provides significantly better S/N than other methods based on periodic signals, such as MLS or linear sine sweep (TDS), as the researches of A.Farina already proved.[8] This made it possible to get a good S/N ratio employing short test signals, speeding up the measurement procedure, and consequently enhancing the time invariance of the system during the measurement period.
These measurements were made inside an anechoic room, to avoid undesired reflections and to maximize the signal/noise ratio.
The test signal was pre-filtered with a suitable FIR in order to flatten perfectly the frequency spectrum and to linearize the phase response of the full-range dual-concentric loudspeaker employed as sound source.
The array was rotated along azimuth (36 steps) and elevation (18 steps), using a movable fixture for the azimuth rotation and a turntable for the elevation. In this way we obtained $36 \times 18 \times 32$ impulse responses, each 2048 samples long (at 48 kHz).

[8]See [7].

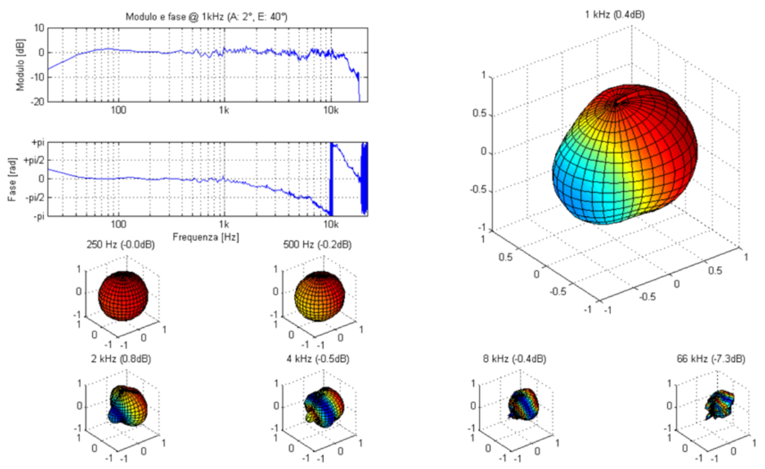


**Figure 3.6:** Frequency response and polar patterns for capsule n.1.

The raw result of the measurement are the response of each capsule of the array to the sound arriving by every direction. In picture 3.6 are shown the results for the capsule n.1 (all the others are very similar):

From the graphs shown in figure 3.6 it is possible to verify that the directivity of the capsules, intrinsically omnidirectional at low frequency, is highly influenced by the microphone structure (the aluminum sphere): above 1 kHz the capsule becomes significantly directive. From the magnitude spectrum calculated in the direction in which the capsule shows the higher sensitivity, it is possible to verify that the transducers have a quite good frequency response from 30 Hz to 13 kHz, with a gentle roll off at higher frequencies.

A small phase error was introduced by this measurement technique in every impulse response, due to the variation of the distance between the microphone array and the sound source during the rotation. In particular the rotating table produced a sinusoidal variation of the distance caused by the little eccentricity of the support, which was accurately measured thanks to the absolute time-of-flight of the sound from the loudspeaker to the microphone, as shown in figure 3.7.

With a mathematical calculation, based on the theoretical distance between capsule and source, implemented in a *Matlab* program, the impulse responses were carefully realigned in time, correcting for these positioning errors. Before

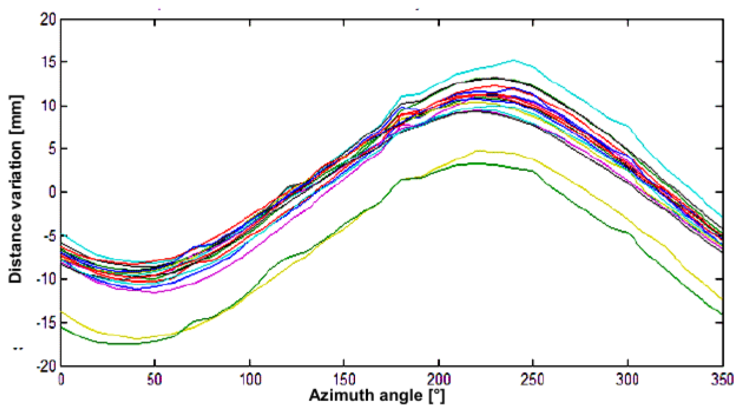


**Figure 3.7:** Variation of the distance during rotation.

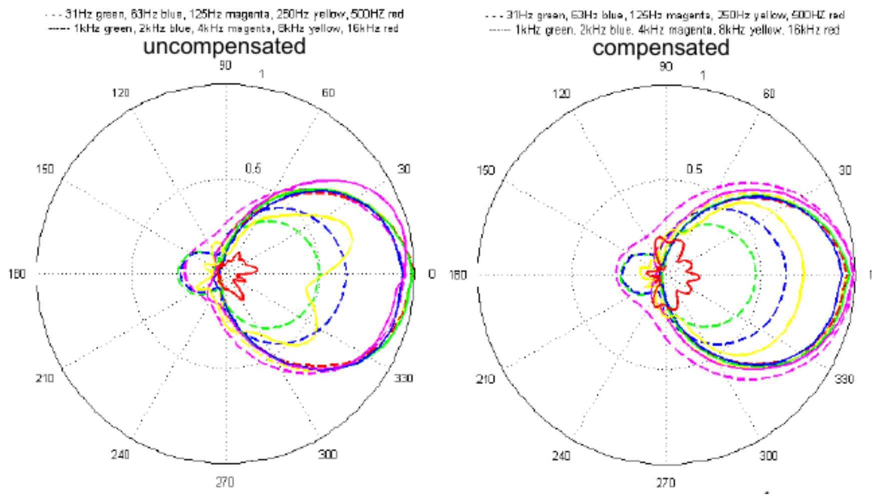


**Figure 3.8:** Improvement of polar patterns of a 3rd order virtual cardioid due to delay compensation.

this operation the worst case of distance error was 30 mm, after it was of 2 mm, a length that corresponds to the wavelength of a 170 kHz signal. For this reason we could assume that the phase coherence of impulse responses is reasonably good in the spectral region of interest. As shown in the figure 3.8, in the case of a 3rd order virtual cardioid, the compensation improves the polar pattern at 4 and 8 kHz.
In the band of 16 kHz the signal is compromised in both cases, and this is probably due to the spatial aliasing.

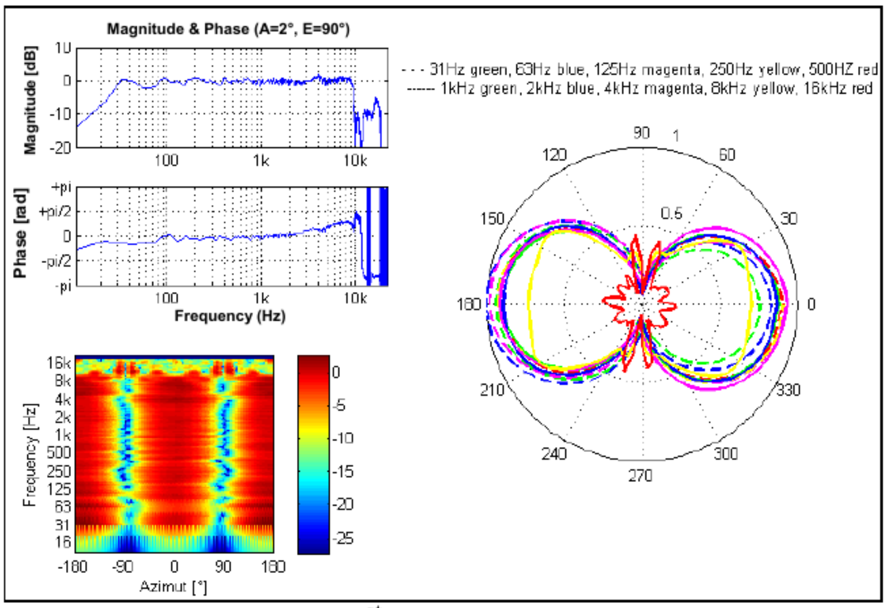


**Figure 3.9:** 1st order figure of eight.

### 3.1.3 Synthesis and test of virtual microphones

In order to derive the matrix of filters, a *Matlab* script was produced. This script employs 2048 samples of each impulse response and it needs as inputs the number of virtual microphones to synthesize, their directivity, their azimuth and elevation. From these inputs, according with the theory and the procedure described in paragraph 3.1.1, it is possible to invert the matrix of impulse responses obtaining the matrix of filters to apply to the capsule signals.

In our specific case the number of incoming sound directions $D$ was 648, the number of microphones $M$ was 32 and the number $V$ of virtual microphones was the one desired.

The convolution of the FIRs matrix with the 32 signals coming from the capsules of the array should give as outputs the signals of virtual microphones with the desired characteristics. In pictures 3.9, 3.10 and 3.11 are shown some experimental results, showing some of the different directivity patterns obtained.

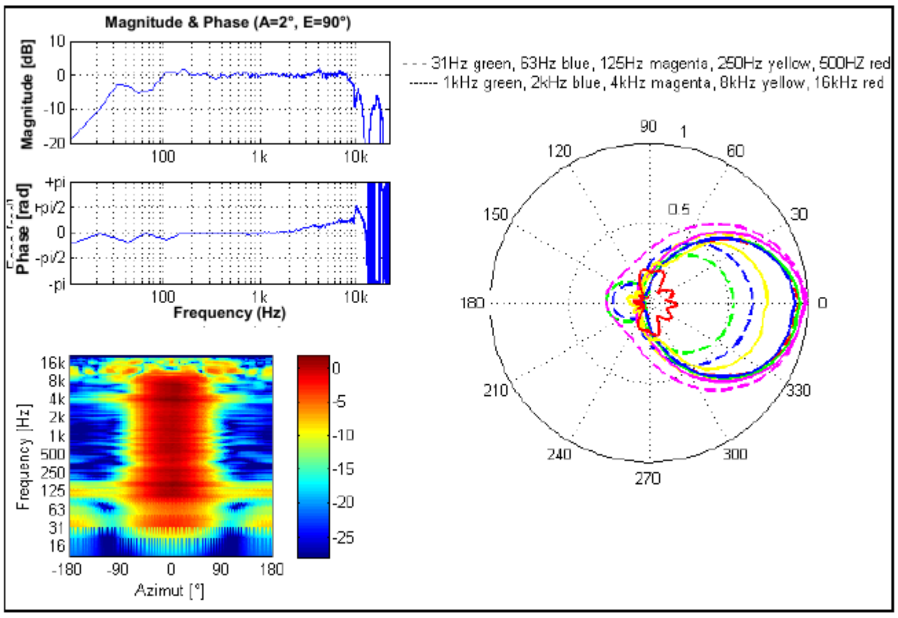


Figure 3.10: 3rd order cardioid.

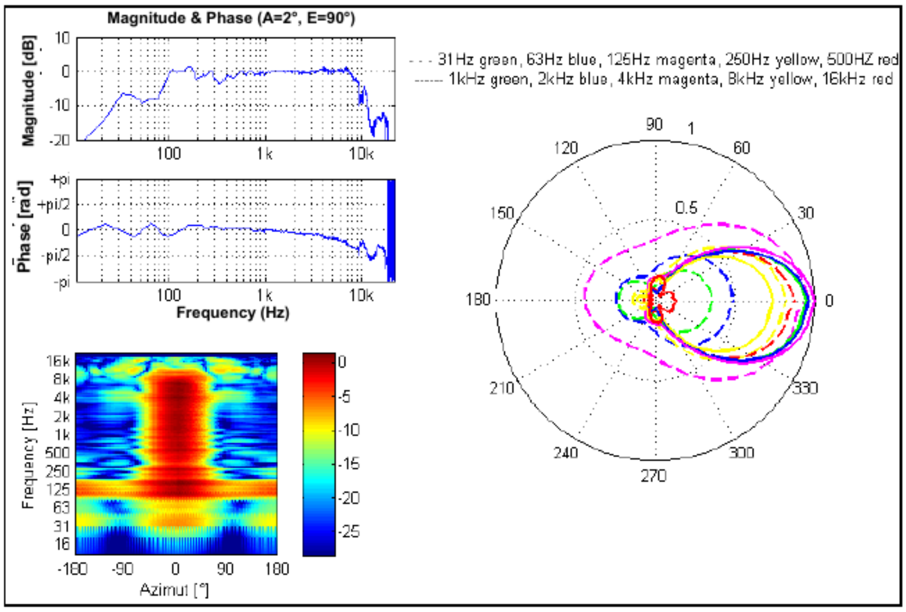


Figure 3.11: 6th order cardioid.

# Chapter 4

# The optical part

The virtual microphone system, like the acoustic camera-like products listed in chapter 2, needs always a visual reference, otherwise it would be impossible to point correctly the virtual microphones to the desired sources. This is requested in both post-processing and real-time applications, so a new problem was introduced during the development of the research project between IED and RAI broadcasting company: the systems, to be suitable for live events, must be provided with a camera and the recording software must be capable to show the frame as background in the virtual microphone pointing space, as fully described in section 5.1.4.

But while a probe like the *Planar Array* requires a "planar" background image, so every video camera can be suitable for this use, a non-planar array needs visual reference that covers as much as possible the spatial range of this kind of probe: for a spherical array, the shooting angles, in terms of width and elevation ranges, are $360°/180°$ and for a cylindrical array they are $360°/\sim 100°$. The solution that has been successfully experimented in the realizations of IED is the coupling of a camera with an hyperbolic mirror that outputs a full panoramic image to be unwrapped with a proper software algorithm, as described in section 4.2.2.

## 4.1 The camera

In the first attempts to embed a camera in the array a low-cost *webcam*, was used, connected to the computer through a USB port, and this kind of device revealed immediately a couple of drawbacks, the most important of them are:

- USB connections are limited to relatively short distances: in a broadcast live set cable connections can be long tenths or hundreds of meters;
- *webcams* have cheap fixed lens and quite low resolutions, not suitable to obtain good quality output frames after the unwrapping process.

To overcome these issues, the search of suitable devices was done between cameras with IP technology (IP-cameras): in these cameras, typically, only a standard network cable with RJ45 plugs is needed to connect them because power also comes from this cable.[1] This is indeed a great benefit for the system robustness and portability because with only two network cables - that, as known, can be long tenths of meters - it is possible to feed, send and receive data from the array and the camera. The quality target was achieved choosing two different cameras, one for the *Planar array* where unwrapping process is not needed and another for the applications where a full 360° image is necessary; the first camera is the model IPELA® SNC-CH210 from Sony® (figure 4.1), the latter is the WS-M8P31-38B model (figure 4.2) from Chinese manufacturer Wision.
In table 4.1 some characteristics of the two cameras are reported.

Both cameras deliver the signal using the *Real Time Streaming Protocol* (RTSP), widely adopted and well supported by the most known video player and recorders.

## 4.2 The panoramic frame

### 4.2.1 Mirror and lens

In order to obtain a good panoramic unwrapped frame it is absolutely necessary to do a very precise optical and mechanical coupling between mirror and camera lens; for the IED *Cylindrical array* was chosen a mirror built by *0-360° Panoramic Optic™* (figure 4.3), that is a glass hyperbolic mirror with a horizontal visual field

[1]The so called *Power over Ethernet* (PoE) technology; obviously a common network adapter or switch cannot power up these devices: a PoE power unit is needed, but this has not to be near the device.

| | Sony IPELA SNC-CH210 | Wision WS-M8P31-38B |
|---|---|---|
| Camera | | |
| Image Sensor | 1/2.8' 3M Pixels CMOS | 1/2.5' 5M Pixels CMOS |
| Minimum illumination | 2.0 lux | Color: 1.0 lux @ F1.2<br>B/W: 0.1 lux @ F1.2 |
| Shutter control | 1 to 1/10000 s | 1/5 to 1/50000 s |
| Lens type | fixed | interchangeable with C/CS mount |
| Horizontal viewing angle | 88° | - |
| Focal length | 3.3 mm | - |
| F-number | 2.8 | - |
| Minimum object distance | 0.5 m | - |
| Image | | |
| Maximum size (H×V) | 2048×1536 | 2592×1920 |
| Codec image size (H×V) | 2048×1536 (JpegOnly),<br>1920×1080,1600×1200,<br>1680×1056,1440×912, 1280×1024,<br>1280×960,1376×768, 1280×800,<br>1024×768,1024×576, 768×576,<br>720×576,704×576, 800×480,<br>720×480,640×480, 640×368,<br>384×288,352×288, 320×240,<br>320×192, 176×144 | 2592×1920, 2048×1536,<br>1920×1080 (1080p),<br>1280×720 (720p) |
| Video compression format | H.264, MPEG-4, M-JPEG | H.264, M-JPEG |
| Maximum frame rates | H.264: 15 fps (1920×1080)/<br>30 fps (1280×720)<br>M-JPEG: 12 fps (2048×1536)/<br>15 fps (1920×1080)/<br>30 fps (1280×720)<br>MPEG-4: 15 fps (1920×1080)/<br>30 fps (1280×720) | 10 fps: 2592×1920<br>20 fps: 2048×1536<br>30 fps: 1920×1080<br>30 fps: 1280×720 |
| Network | | |
| Protocols | IPv4,IPv6,TCP,UDP,ARP,ICMP,<br>IGMP,HTTP,HTTPS,FTP (client),<br>SMTP,DHCP,DNS,NTP,RTP/<br>RTCP,RTSP,SNMP (MIB-2) | HTTP,DHCP,UDP,RTP/RTSP,<br>NTP,FTP |
| ONVIF software | Yes | Yes |
| Interface | | |
| Ethernet | 10BASE-T/100BASE-T (RJ-45) | 10BASE-T/100BASE-T (RJ-45) |
| General | | |
| Weight | 100 g | - |
| Dimensions | ⌀44 × 93 mm | - |
| Power | PoE (IEEE802.3af compliant) | DC 12V/PoE |

**Table 4.1:** Some characteristics of the Sony® and Wision cameras adopted for planar and cylindrical array respectively.

of 360° and a vertical one of 115°,[2] that was, in fact, a project constraint. The hyperbolic shape conducts the reflected rays in the *first* focal point of the hyperboloid - "internal" to the mirror - hence the optical center of the camera lens must be placed in the *second* focal point. A mirror with these characteristics is

[2]Not symmetric: 52.5° above and 62.5° under the horizon.

**Figure 4.1:** The Sony® IPELA® SNC-CH210 IP-camera with black case.

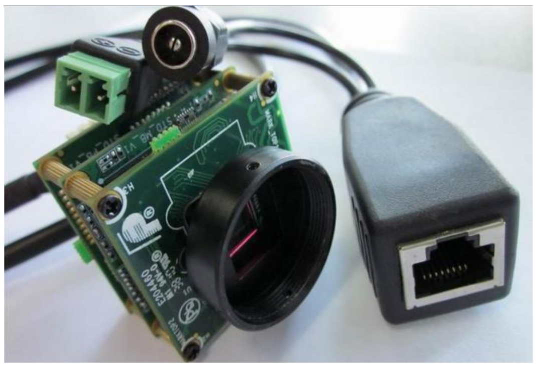


**Figure 4.2:** The Wision WS-M8P31-38B IP-camera in bulk version without lens.

defined as *single point of view*, and allows the correct estimation of distances and moving objects as it were a natural perspective image. That was not true for spherical mirrors, where rays doesn't intersect in a single point, creating a spherical aberration; parabolic mirrors, however, require a telecentric lens in order to obtain single point of view images.

Once the mirror has been chosen, the correct camera lens had to be found; the target was to obtain a complete mirror image with minimum or no aberration at all, minimizing at the same time the distance between mirror and lens, to make the entire system as small as possible.

Different lenses was tested, evaluating the following parameters:

- the minimum distance $D$ that allows the correct field angle on the minor

**Figure 4.3:** The *0-360° Panoramic Optic™* hyperbolic mirror.

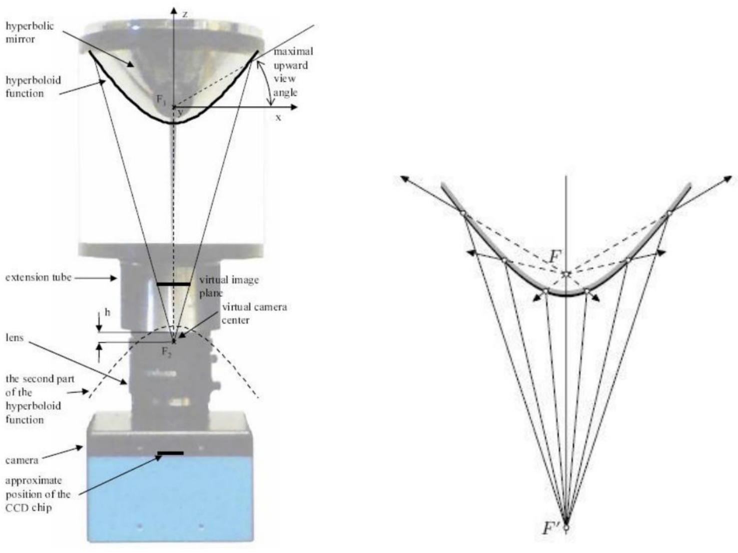


**Figure 4.4:** The optical path in an omnidirectional vision system with hyperbolic mirror.

CCD sensor dimension (usually, the vertical one);

- the minimum focus distance on vertical pattern $W_r$ ($D$ on figure 4.5);
- the *field depth*, or the correct focus within a defined range;
- the qualitative optical distortion;
- the lens physical size.

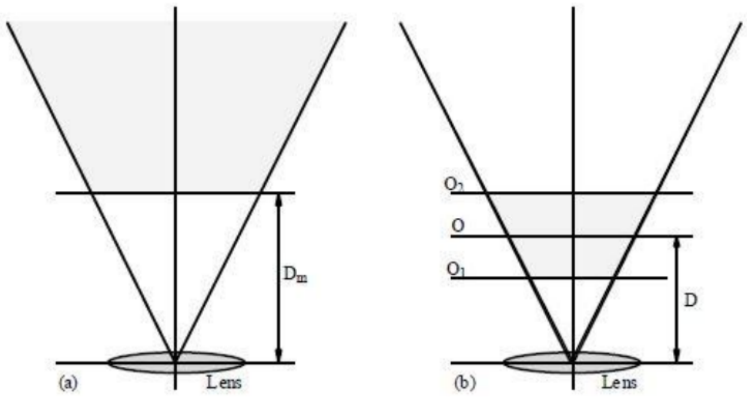


**Figure 4.5:** Scheme of the field depth.

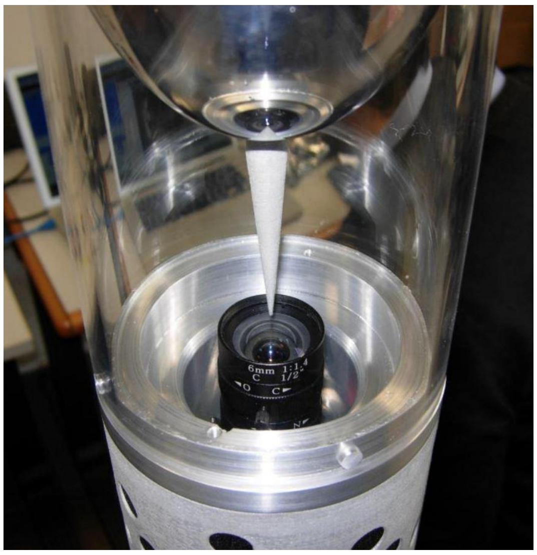


**Figure 4.6:** The mirror and the camera in test mount; the cover of the space around the lens is missing and the body is not yet painted in the final black color.

Finally, it was chosen a 6 mm lens, with iris range from F1.2 to F16
The Camera and mirror has been mounted, perfectly coaxial, inside a transparent plexiglass container, with a black needle that removes internal optical reflections. The finished object is visible in figure 4.6.

Since, once the system is complete the camera is not anymore easily reachable, before screwing the structure a lens setup has been done, finding the regulations that allow the largest field of applications: lens focus was regulated to find a

compromise focus for near and far subjects in the mirror reflected image. The field depth is, in this case, the really important parameter in order to focus subjects placed at different distances. The chosen setup is slightly out from the near field.
Then the iris diaphragm opening has to be regulated, determining the amount of light that enters in the lens and heavily influencing the field depth obtainable. Closed iris means long field depth, conversely is the iris is open, the field depth will result very short. Obviously the desired behavior was to have an infinite field depth, but when the environment is poor illuminated, the camera built-in automatic image lightness control system intervenes, opening the iris to get more light and reducing the field depth. So the iris opening was regulated in order to obtain the maximum field depth and the minimum disturb allowed in poor illuminated environments, about 2/3 of the fully opening.[3]

### 4.2.2 The unwrapping technique

There are a lot of commercial and free software to obtain the projection on a rectangular surface of the 360° mirror reflected image - the *unwrapping* -, but all of them does single images processing; in this case the camera video stream has to be unwrapped in real-time, frame by frame, thus a software was developed for this purpose, in the form of GStreamer (see section 5.1.4) plugin, because the images processed are large (5 Mpixel) and the performance requested is high.

The core of the geometric transformation is the passage from the cartesian coordinates system of the starting frame (rectangular, with the image of the mirror, as can be seen in figure 4.7) to a polar coordinates system centered on the center of the image of the mirror, then returning to a new cartesian coordinates system with only the pixel contained in the mirror image suitably translated so as to form a rectangular image.

The processing algorithm to work properly needs some parameters that de-

[3]The same light regulation system corrects the entire image lightness preferring an intermediate value between minimum and maximum: this mean that environments with concentrated light sources and dark zones results, respectively, over and under-exposed.

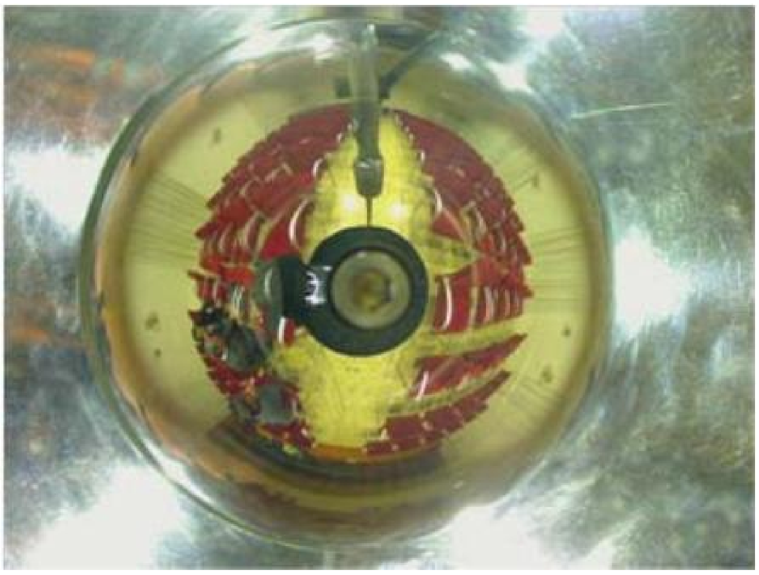


**Figure 4.7:** The image of the mirror seen by the camera.

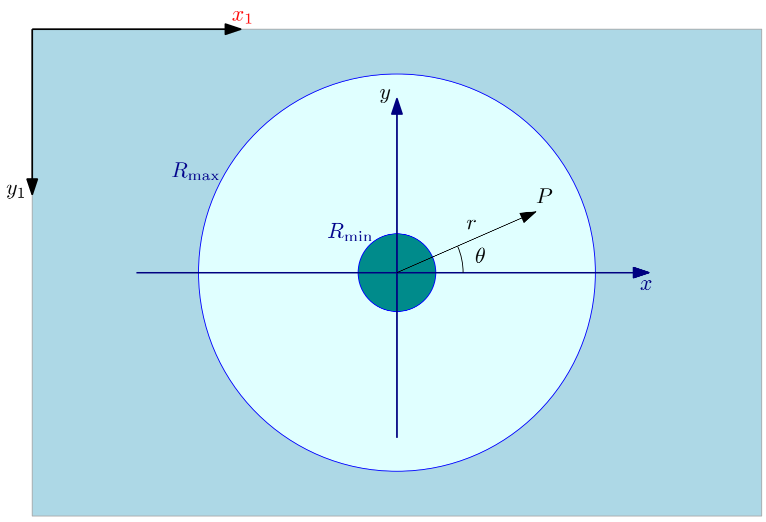


**Figure 4.8:** The polar coordinates system.

pends by the particular physical shape of the employed mirror and by the optical coupling between mirror and camera; in figure 4.8 are highlighted the radius $R_{\max}$ and $R_{\min}$ that can be determined with a calibration procedure, using a static test frame, as can be seen also in figure 4.9.

The output reference system is shown in picture 4.10, where the scale factor of both axis is user defined and it depends by the desired size of the output frame.

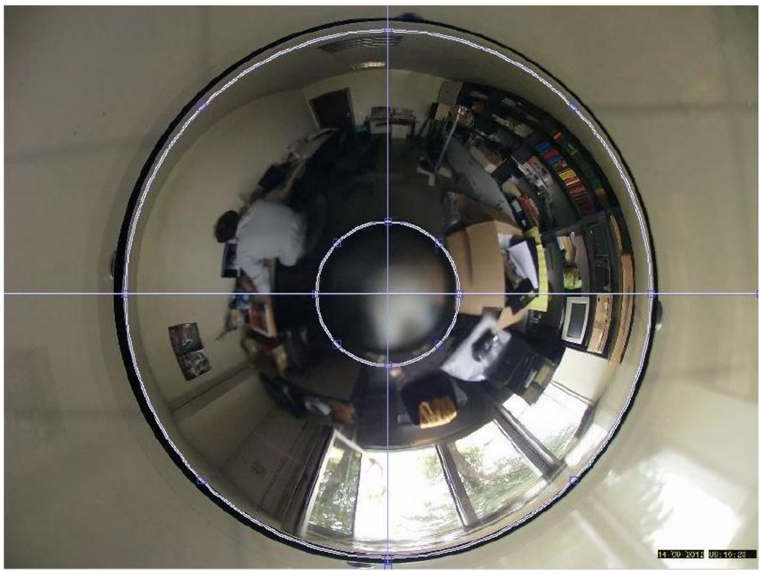


**Figure 4.9:** The optical calibration procedure.

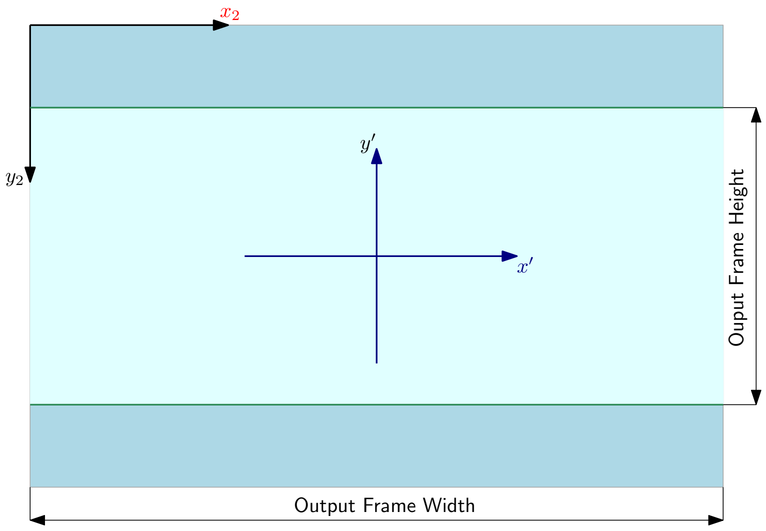


**Figure 4.10:** The output (cartesian) coordinates system.

It is also necessary to know the curvature of the mirror shape, in this case expressed by the experimentally obtained[4] equation

$$y' = -0.4r^2 + 1.4r, \tag{4.1}$$

that correlates a pixel final $y'$ coordinate (figure 4.10) to the radius $r$ of the

[4]No informations about it were given from the producer.

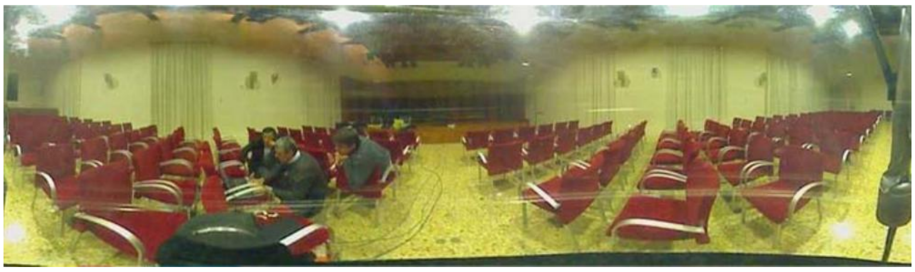


**Figure 4.11:** The *unwrapped* frame.

starting $(\theta, r)$ reference system (figure 4.8).

Now it is possible to obtain a law, based on the polar to cartesian conversion equations

$$\begin{cases} x = r\cos\theta \\ y = r\sin\theta, \end{cases} \tag{4.2}$$

that accomplish the desired *unwrapping*, as can be seen in figure 4.11. This algorithm was firstly written as Matlab script, then implemented modifying an existing GStreamer module[5] and finally a brand new module, named VMSUnwrap, has been written and inserted in the main 3D VMS source code.

[5]Work done mostly by Eng. L.Chiesi.

# Chapter 5

# The software

After the first successful experiments with the virtual microphones technique, mainly executed with *Matlab* scripts, it was quickly clear that to obtain a product stable, reliable and easy installable in a live broadcast or post-production setup, as requested by the partnership with RAI, a new custom software had to be developed, and, through progressive and continuous enhancements and improvements, this software, called *3D Virtual Microphones System* (3D VMS), is currently always used as part of the standard RAI live broadcast equipment. To reach this target, a number of issues had to be faced:

- provide high directivity in order to exclude the undesired noises that usually affect audio recordings in broadcast and film production;
- provide the capability of changing orientation and directivity in real-time by using a controller (joystick/mouse/trackpad);
- the capability of using this system in post-production, varying the virtual microphones on recorded audio;
- the capability of seeing the subject on which the virtual microphone is focusing;
- low latency;
- high audio quality;
- the control unit must run on Apple® OS X®.

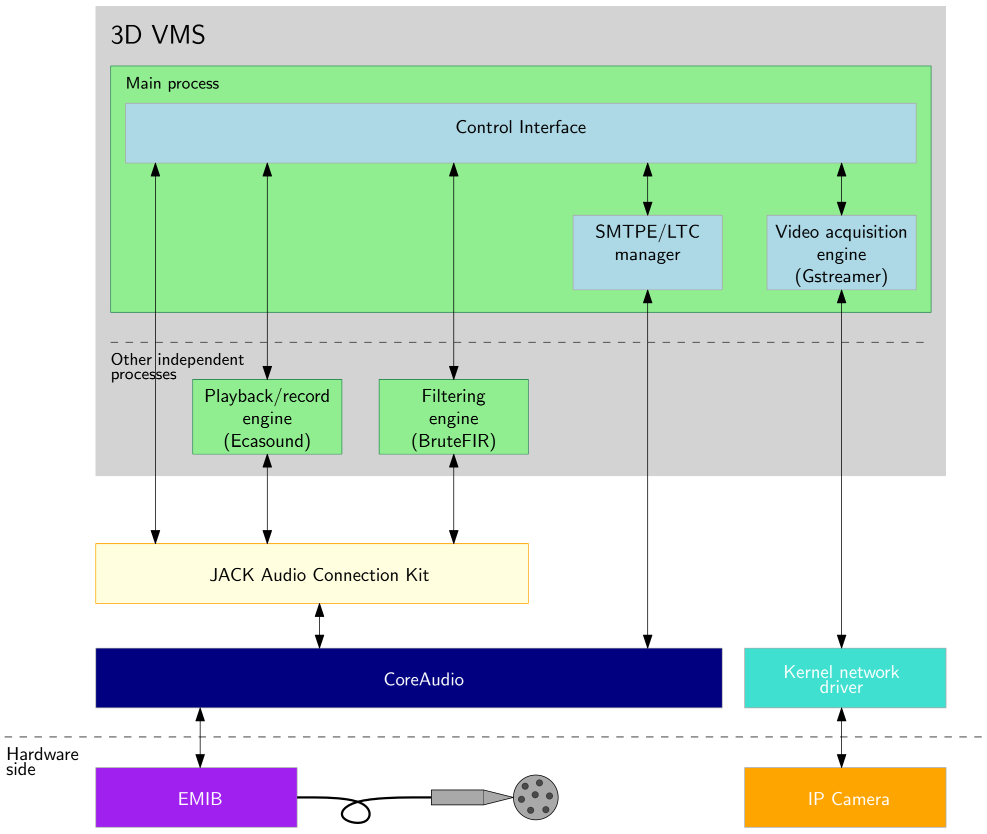

3D VMS
Main process
Control Interface
SMTPE/LTC manager
Video acquisition engine (Gstreamer)
Other independent processes
Playback/record engine (Ecasound)
Filtering engine (BruteFIR)
JACK Audio Connection Kit
CoreAudio
Kernel network driver
Hardware side
EMIB
IP Camera

**Figure 5.1:** The structure of 3D VMS.

## 5.1 3D Virtual Microphones System

How can be seen in figure 5.1, the 3D VMS is composed of different parts, in some ways independents between them: roughly it is possible to individuate three main blocks:

- the filtering engine
- the recording/playback engine
- the control interface.

In the really first implementation of the 3D VMS these blocks resides on different machines because of an attempt to make the entire system rack-mounted and remote controlled, so the audio components were enclosed in a *black box*, controlled by a laptop, where the graphical interface was installed: this layout influenced deeply the development of 3D VMS that, like sometimes happens in big projects, in fact never leave this architecture; actually the system runs flawlessly on a single, well boosted, laptop,[1] but the software components still acts as they are on different machines, so they are completely separated processes.

These blocks can communicate in bi-directional way between them to grant full control on what is happening and all the audio is routed through *Jack Audio Connection Kit*[2] (JACK).

As previously mentioned, a very important specification, needed in particular for live processing, is the latency that has to be very low, typically inferior to 40 ms.

### 5.1.1 From Pyhton to OS X® App

Actually the 3D VMS is written completely in C/C++, but at the beginning of the project it was not so: while BruteFIR and Ecasound were - and still they are - stand alone programs, to obtain quickly a working prototype of the system, Python programming language was employed to write the GUI, at the time indistinguishable from the control interface, built with wxWidgets[3] libraries that guarantee a visual family-feeling with other OS X® applications.

The very first versions of the software were written with a prototyping paradigm in mind, so very little true object-oriented programming (OOP) was used, but this, when the complexity begun to grow, soon had shown its limits: in 2011 a first complete rewrote of the code brought to a full OOP Python code, much more readable and simpler to maintain.

Python is a wonderful tool that allows to write programs in a small amount of time, forcing the author to use some "style", but it is and remains a very high-

[1]When the first prototype was built, in 2009, the CPU computational power requested by the filtering engine was heavy, and a single portable computer cannot run the entire system without glitches in the audio output. Nowadays, the audio engine is still demanding, but laptops are much more powerful and for live events less boxes are needed, the better is.

[2]http://www.jackaudio.org

[3]http://www.wxwidgets.org

level language that needs an interpret and a lot of installed libraries, plus, in this case, the wxPyhon framework; one of the first big problem encountered was the incompatibility between the default OS X® Python version[4] and the binary wx-Python distribution: this was resolved easily freezing the Python version to 2.6, but it comes immediately clear the risk of a software hang after, for example, an OS update that made unavailable the chosen version of Python.[5] Again, Python is a non-compiled language, this means that every kind of error comes out at runtime, and it is possible to detect them only running the program from the console, if this doesn't happen, the application hangs and the user doesn't know why.

Finally the 3D VMS has to be simply installed and eventually uninstalled, like a common OS X® application: the Python version of 3D VMS had really a lot of dependencies, not easy to bundle in a reliable way.

For these reason, in the fall 2013, the 3D VMS was entirely converted to C++ and all its components bundled in a OS X® App; the only external dependency remained is JACK that still have to be installed separately.

The code conversion was quite straightforward because most part of the Python source had been written using wx-based classes[6] and since wxWidgets C++ namespace is very similar to its Python implementation, the editor search/replace function was one of the most used.

As side-effect, the C++ version has significantly better performances than the previous Python one, and this became really crucial when GStreamer entered in 3D VMS source mainstream (see paragraph 5.1.4).

[4]In OS X® 10.5 and 10.6 the default Python version was 2.6; in OS X® 10.7, 2.6 was still installed, but the default was 2.7.

[5]Another considered solution was to embed in the 3D VMS installation package the entire working Python 2.6 + wxPython distributions, but writer fully disagree with this kind of approaches.

[6]wxWidgets library offers not only GUI related classes, but a very complete set of helpers and accessories to cover all the needing of a generic application in a cross-platform way.

### 5.1.2 The filtering engine

One of the most critical parts of the system is obviously the filtering process, that involves not only a non-negligible number of channels, but also it must have the capability of real-time filter switch, when the user moves a virtual microphone in the space.
To reach a good reliability as soon as possible, instead of writing a filtering engine from scratch, a ready-made solution was searched, finding an open source tool of proved stability and reliability, BruteFIR[7] written by Anders Torger[8], that matches system requests.

BruteFIR permits the implementation of up to 256 filters and considering that for one single microphone the system needs 32 filters - one for each input channel - we can get 8 virtual microphones. The choice of synthesizing only 7 virtual microphones is due to the need of having one *free slot* (32 filters) dedicated to the *double buffering*, the dynamic switch from the old and the new virtual microphone every time the user needs to modify the characteristics of one of the virtual microphones (figure 5.2). With BruteFIR, it is also possible to have a cross-fading during the change of the coefficients with the benefit of a glitch-free transition between two different positions of the virtual microphone.

BruteFIR, born for GNU/Linux, was never ported to OS X® systems, but its intrinsic POSIX nature and the full support of JACK audio interface had permitted a relatively straightforward recompilation on OS X® environment, with only some minor changes.

To keep the 3D VMS control interface informed about program status, an error communication channel was added, implemented with a POSIX socket to which is possible for another application to connect and receive status messages from BruteFIR.

[7]http://www.ludd.luth.se/ torger/brutefir.html
[8]See [21].

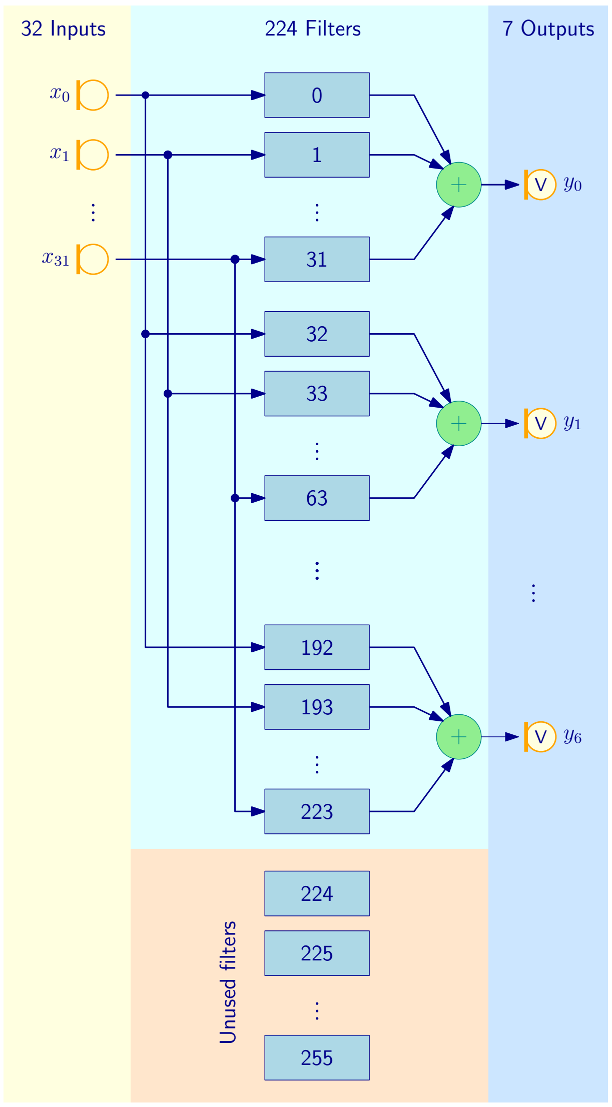


**Figure 5.2:** BruteFIR filtering scheme.

#### 5.1.2.1 The logic module

Obviously BruteFIR doesn't come with the implementation of the FIR's synthesis process described in the paragraph 3.1.3, but it's fully modular and its open source nature permits to write new modules[9] starting from existing ones, so it had to be possible to realize a 3D VMS own module implementing direction dependent filter synthesis, and permitting the control of 7 independent virtual microphones. The virtual microphone's parameters can be modified in real-time via POSIX

[9]In BruteFIR world a module is a plugin, loaded in runtime on request.

socket interface[10] through a proper protocol. Every time the parameters are modified, the system automatically recalculates the FIR filters and substitutes them in the memory of the convolver through a *double buffering* mechanism.
The filter computation subroutine was rewritten in highly-optimized C, making it possible to recalculate all the filters in a few milliseconds, so that the virtual microphones can be smoothly moved around without artifacts.

### 5.1.3 The recording/playback engine

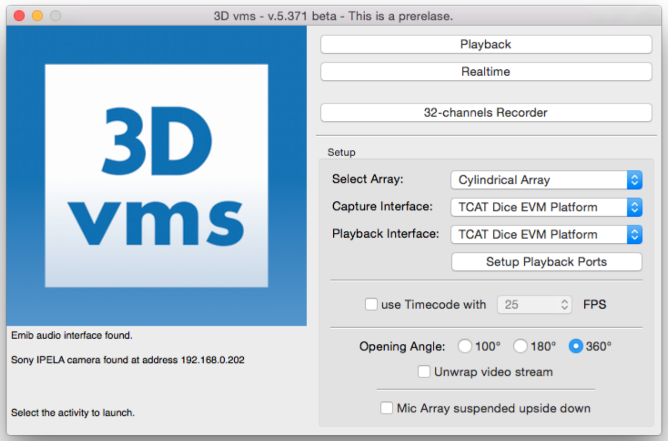
The recording of 32 channels at the same time also, is a task for which there are plenty of affordable and reliable good solutions, so another time an open source program fits perfectly the requests: Ecasound[11] by Kai Vehmanen.
To be honest, this software is much more than a multichannel player and recorder, having also advanced mixing and filtering capabilities, not needed for this project, but the interactive mode and the ready-made Python and C++ interfaces come immediately very useful when Ecasound has to be controlled from an external process, and this is the case of 3D VMS: thanks to these features it is possible at every instant to send commands and to have a feedback from the audio transport without latency at all.

Unlike BruteFIR, Ecasound is distributed for various platforms, including OSX, but the version included in the 3D VMS package is in many ways customized to support BWF file format, needed to store correctly timecode data, as explained in section 5.1.5.

### 5.1.4 The multimedia engine

It was completely described in chapter 4 the optical system needed to have a coherent visual reference of what is shot by the camera, and also it was described that video frames have to be processed to give out the correct unwrapped image, but the first 3D VMS software version mentioned in section 5.1.1, in fact, aren't

[10]BruteFIR support various communication interfaces: when filtering and control were on two separated machines, TCP/IP was used.
[11]http://nosignal.fi/ecasound/



3D vms - v.5.371 beta - This is a prerelase.

Setup

Select Array: Cylindrical Array

Capture Interface: TCAT Dice EVM Platform

Playback Interface: TCAT Dice EVM Platform

☐ use Timecode with 25 FPS

Opening Angle: ○ 100° ○ 180° [✓ 360°]

☐ Unwrap video stream

☐ Mic Array suspended upside down

Emib audio interface found.

Sony IPELA camera found at address 192.168.0.202

Select the activity to launch.

**Figure 5.3:** The 3D VMS main dialog window.

able to do the elaboration in real-time: this was done only in post-processing, because the wxPython wrap of the framework used to display the captured video doesn't allow the addition of code for frame processing.

Since the correct visualization of the video reference during real-time filtering had to be an essential software feature, an extensible and embeddable tool had been searched, and GStreamer[12] was found.

GStreamer is indeed a really complex piece of software: it is open source and what follows is only a subset of all its features, taken from the website:

- graph-based structure allows arbitrary pipeline construction;
- multi-threaded pipelines are trivial and transparent to construct;
- clean, simple and stable API for both plugin and application developers;
- extremely lightweight data passing means very high performance/low latency;
- dynamically loaded plugins provide elements and media types, demand-loaded via a registry cache;

[12]http://gstreamer.freedesktop.org

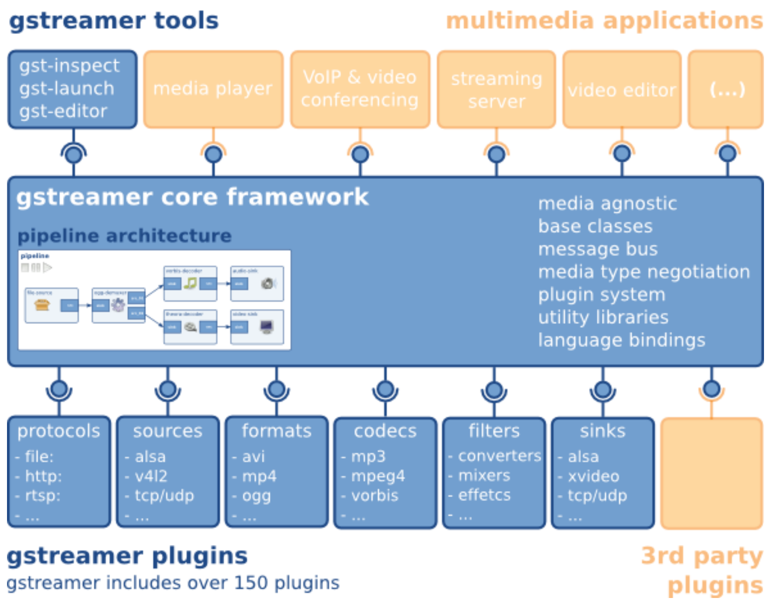


**Figure 5.4:** GStreamer architecture.

- GStreamer capabilities can be extended through new plugins.

- Some features available using the GStreamer own plugins, not counting any 3rd party offerings:

  **container formats** : asf, avi, 3gp/mp4/mov, flv, mpeg-ps/ts, mkv/webm, mxf, ogg;

  **streaming** : http, mms, rtsp;

  **codecs** : FFmpeg, various codec libraries, 3rd party codec packs;

  **metadata** : native container formats with a common mapping between them;

- *gst-launch* command-line tool for quick prototyping and testing, similar to Ecasound;

- a lot of documentation, including partially completed manual and plugin writer's guide;

- access to GStreamer API with various programming languages.

A scheme of how GStreamer works is reported in picture 5.4, where is pretty clear its full modular structure, much similar to a studio with a lot of rack mounted devices routed together to obtain the desired effects on the incoming signal that can be video or audio or both at the same time. From the user point of view, the modules are connected using the principle of the *pipeline*, and the GStreamer distribution provide a command line tool named gst-launch to which it is possible passing the desired pipeline as a string parameter with some syntax conventions. Every module can have one or more inputs named *sinks* and/or one or more outputs named *sources*; modules without sinks, normally start the pipeline, because they themselves are a source, conversely modules without sources, are placed at the end of the pipeline. It is possible to split the pipeline in two or more branches using, for example, a *tee*, a module with one sink and two or more sources and, with the GStreamer API, it is possible to dynamically change the pipeline.
A source and a sink, when connected together must be compatible,[13] if it's not the case, an appropriate conversion module must be placed between them.
A module behavior can be controlled by parameters that can be varied by the user exactly like pushing buttons or moving knobs on a rack device. For example

```
gst-launch videotestsrc ! ximagesink
```

this really simple pipeline use the standard test video signal generator module, named videotestsrc piped - the '!' symbol - to the X window server monitor, so in a X based environment a window will display the piped video signal. Another more complex example

```
gst-launch videotestsrc ! \
       video/x-raw-yuv, framerate=20/1, width=640, height=480 ! \
       x264enc bitrate=512 ! mpegsmux ! filesink location=test.ts
```

Here the same video test signal is piped to a x264 encoder, with bitrate set to 512, through *caps filtered*[14] source/sink which set the image format and the

[13]The tool *gst-inspect* called with a module name as parameter, gives to the user a full information page about its parameters, sources and sinks.

[14]That is the contraction of *capabilities filter*: it is not a module but it is a limitation imposed to the source/sink of connected modules.

framerate; then the signal is multiplexed into a MPEG container and sent to a file whose name is passed as location parameter to the filesink module.

Thanks to the develop and debug tools made available, the unwrapping plugin prototype was quickly developed, then the challenge was to integrate GStreamer with the existing 3D VMS mainstream that is wxWidgets based, so it hasn't nothing to do with the *glib*[15] based GStreamer, that, like BruteFIR and Ecasound, works as an independent process with its own threads, a messages' bus, etc. The junction point is the capability of GStreamer to send a video stream on various destinations, one of them is the so-called appsink from which, defining a callback function, it is possible to access the frame buffer and, for example, to copy it to a window canvas: this is exactly what happens inside 3D VMS, and it becomes clear how performance is important because the amount of data that has to be copied from GStreamer buffer to the window canvas can be huge in the case of 5 Mpixel RGB images.

The scheme of the pipeline used in the *Realtime* activity of 3D VMS is fully reported in picture 5.5: the RTSP signal coming from the IP-camera is collected by the rtspsrc module that begin the pipeline, pass through some conversions, then, if it is the case of a panoramic image, it is unwrapped, cropped and *boxed*.[16] If the array is placed upside-down, the (unwrapped) frame is vertically flipped and, after a proper scaling,[17] the signal route is split: one branch goes to appsink module, making the current frame available for the virtual microphones pointing canvas, the other branch, after some additional conversions, writes the stream to a file. The *recording* branch is fully connected only when the user press the *record* button on the application.

[15] *glib* is one of the core libraries of all GTK applications, containing the definitions of the base classes from which the GTK objects are derived. The interesting - or crazy, it depends by the point of view - thing is that all this was written in C plain, implementing from scratch a complete object oriented superstructure.

[16] The *boxing* process is the addition of black bands above and below to fit the desired frame size.

[17] This scaling can be requested by the *zoom* function, as explained in section 5.2.6.

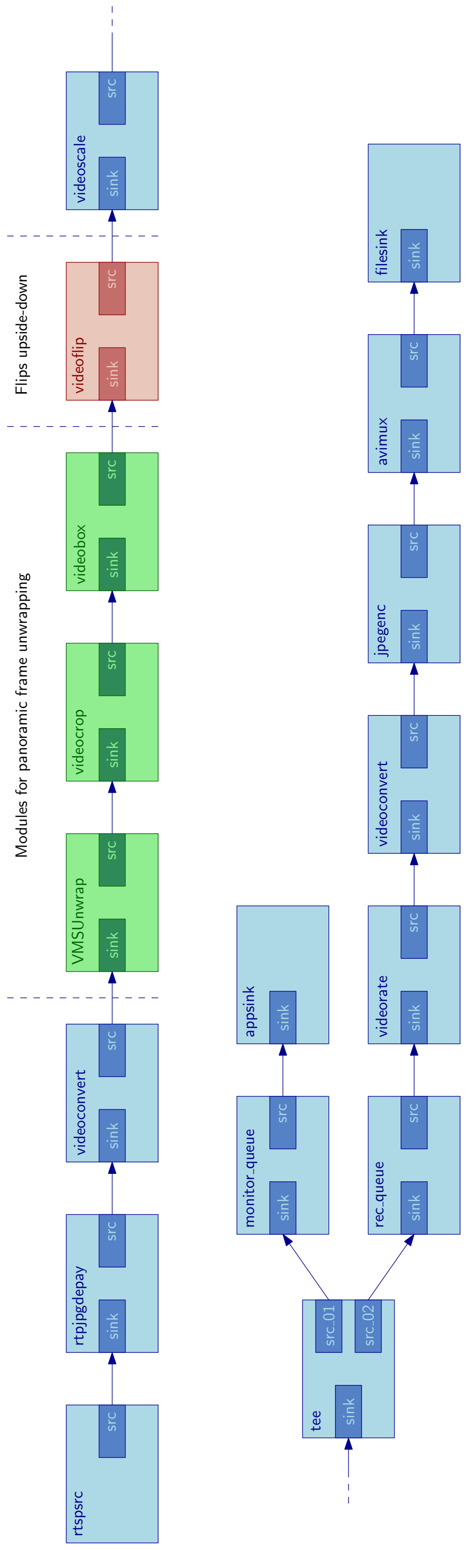


**Figure 5.5:** The GStreamer pipeline implemented in the *Recording* activity of 3D VMS. Green modules are connected only when a panoramic camera is used, and pink module is connected when the camera is placed upside-down.

### 5.1.5 The timecode manager

When shooting broadcast events, it's normal to have a lot of audio and video signal coming from a certain number of cameras and microphones; so, in post-production the amount of tracks to edit is very large and it is absolutely necessary that all tracks are synchronized together. The *ciak* sound is the best known trick to find sync between audio and video, but nowadays the LTC/SMPTE is the standard signal to keep synced all the electronic equipment used in the set. No exception for 3D VMS.

The system had to be able to:

- decode the LTC/SMPTE signal from an external source, maintain the reference during the recording session and store it in the output audio file;
- encode an LTC/SMPTE signal when playback an audio file with timecode reference;

this became possible thanks to *libltcsmpte*[18] by Robin Gareus, now deprecated, but working perfectly as decoder/encoder for such kinds of signals.

Solving the *decoding* problem was not immediate because the EMIB interface, that is normally connected to the laptop during a shooting session, has as the only input the Eigenmike® and there aren't other connectors: the only audio input available was the *Line-in* connector of the internal laptop audio card, but the JACK server wasn't able to manage more than one audio interface.

The solution found was to write the timecode codec as direct CoreAudio thread, leaving JACK to manage the virtual microphones related audio connections, as shown in figure 5.1.

Since it is not possible to maintain physically connected the LTC/SMPTE signal to whatever acquisition device during the entire recording session, the 3D VMS timecode subsystem, has to be autonomous after the synchronization

[18]http://ltcsmpte.sourceforge.net/

procedure and this was achieved by storing the timecode reference as offset between the laptop internal clock that has far than sufficient precision and stability.

So, the procedure to synchronize the 3D VMS with an external timecode source is the following:

1. connect a 3.5 inches male jack terminated cable with LTC/SMPTE signal on the mono/left channel to the *Line-in* connector of the laptop[19];
2. press the *Synchronize* button on the *Realtime* or *Recorder* application's GUI and wait 5 seconds;
3. disconnect the cable. The system is now synchronized.

The LTC/SMPTE is then stored as audio channel on the output audio file that is thus formed by 32+1 channels.[20]

In the *Playback* application, the timecode codec can be synchronized with BWF file informations, but this is not a very used options because commonly W64 with 32+1 channels are used, thus LTC/SMPTE signal is fully available and directly routed to output port, together with virtual microphones signals.

### 5.1.6 The control interface

This part of the software has the task of managing the operations of the entire application: it starts and stops processes and continuously checks if there are errors.
On the top of the control interface there is a Graphical User Interface (GUI) that

[19]With the latest MacBook Pro® machines, a new problem came out: the *Line-in* connector disappeared, and all audio connections to the internal audio card pass through a single 3.5 inches combo jack connector for both input and output; the input is not a line-in but a mic and is turned on after driver sensing, so a little harware interface has to be developed to connect an external line signal.

[20]An attempt was done to use the BWF (Broadcast Wave Format) audio file format to store timecode data, but since the timecode management is not natively supported by audio player/recorder Ecasound, a lot of modifications would be needed to add full support, especially for playback activity.

is what the final user see and what he interacts with; the GUI sends commands to the control interface, and receives information from it, giving full feedback to the user; figure 5.1 describes well the interactions between 3D VMS components and, as can be seen, the control interface resides in the main process and communicates bidirectionally with everything else, to maintain the system in a known state and, in case of errors, to inform instantly the user on what happened.

### 5.1.7 The actual structure

Currently the 3D VMS application is composed by three main activities:

- *Playback*
- *Realtime*
- *Recording*

that will be described in the next sections; they share some common modules, modeled as objects, corresponding to the various resources and tasks used by the application during its workflow. The resources include:

**Configuration manager** , used to store the global 3D VMS configuration: this is a *singleton* always available to all instantiated objects;

**Matrix manager** , that made available to the 3D VMS the matrices installed on the system; on application startup it always does a scan in the standard directories looking for existing setups, and make a list that is available through the *Configuration manager*;

**IP-camera manager** , that checks IP-camera connection and, if present, tries to configure it with *Configuration manager* parameters.

The tasks, however, include:

**JACK connections manager** : a system that query the running JACK server for existing connections, and, accordingly with other tasks setup or user requests, make routing between running clients;

**BruteFIR manager** , to start, stop and communicate with BruteFIR;

**Ecasound manager** , the same as the previous but with Ecasound;

**Virtual Microphones manager** : the system that collects user requests on whatever concerns virtual microphones through its GUI and communicate them to BruteFIR manager;

**Timecode manager** , fully described in section 5.1.5;

**VU-meter display** , window that shows real-time microphone capsules levels.

Once launched the application follows these steps:

- check for system shared memory configuration[21]
- check for connected audio cards and if these are suitable for recordings with 32 channels: if this is the case, the more 3D VMS compliant interface will be selected then *Recording* and *Realtime* activities will be enabled;
- if a recording audio card was found, search for IP-camera and, if present, configure it;
- do software parameter initialization, accordingly with previous checks, searching for existing matrices/setups;
- start JACK server for the selected audio interface;
- show main dialog, waiting for user actions.

Now the user can start one of the available activities, then, after the session is terminated, it will return to this dialog.

[21]This is requested by BruteFIR, that, being a multiprocess program, it needs a correctly configured shared memory to work properly.

## 5.2 How to use the 3D VMS

In order to give a more complete perspective of the software, the following paragraphs may be considered a sort of *user manual*, with detailed instruction about software and hardware installation, and the description of all the current functionalities plus the more common operating procedures.

### 5.2.1 System requirements

3D VMS works on 64-bit OS X® platforms $\geq$ 10.7 with at least 4 GB of RAM memory and a lot of space on disk for recordings, because they are really *huge*.[22]
A fast CPU is recommended[23] to get a smooth processing of virtual microphones, especially when video processing is also requested.
As previously said, the software needs JACK, that can be downloaded from its website, but the ready-made JackOSX binary is usually the best choice for the Mac® environments. Since 3D VMS is JACK-version agnostic there isn't a version recommended: choose the latest stable, but the compatibility with the installed OS X® release has *always* to be verified.

### 5.2.2 Installation

The 3D VMS software package is composed by an application bundle named 3D vms.app[24] whose icon is shown in picture 5.6 plus a bunch of matrix files that may be organized in subdirectories.

The application bundle can be launched from everywhere, but the best place to copy it is, as always, the Application directory; for all the 3D VMS related resources, a directory named vms must be created in the current user home directory, then the matrix files must be copied in the subdirectory $HOME/vms/matrices[25]; now, in order that 3D VMS can find correctly the matrix files, it may contain:

[22]10 GB for a 30 minutes recording is a normal size.
[23]Intel® Core™ i7 with four cores at the time of writing, late 2014.
[24]In a normal OS X® installation, the extension .app should be invisible.
[25]In all operating systems, of the Unix family, $HOME is an environment variable that points to the current user home directory; for example, if there is a user named joe on a OS X® system,

**Figure 5.6:** The 3D VMS application icon.

- single files with extension .dat, the very first version of matrix files, in which are stored 684 IRs[26];
- file pairs .dat/.dir, the first holds the impulse responses, the latter the azimuth and elevation angles in which they are measured;
- subdirectories containing a pair .dat/.dir plus a file named metadata.xml that holds the setup parameters.

The latter is the **recommended** method to store 3D VMS setups,[27] because from the parameter stored in the metadata file, a complete and consistent state for the system can be initialized.

Last thing to do to complete the installation is the configuration of the shared memory. This is due to the multiprocess nature of BruteFIR, since processes in a UNIX® system normally have separated memory addressing spaces and can share datas using Inter-Process Communication (pipes, sockets, etc.); for a realtime application this is a largely inefficient method, thus a shared memory addressing space has to be configured to allow fast data sharing between BruteFIR filtering processes. The shared memory configuration consist of parameters that has to be passed to the operating system kernel and stored in the file /etc/sysctl.conf: it is a simply text[28] file and if it doesn't exist, it must be created. Here the lines to add:

```
# brutefir settings: 1GB of shared memory
```

then $HOME will point to /Users/joe.

[26]This is exactly the total number of steps of the array characterization described in section 3.1.2.

[27]The file formats are fully described in appendix A.

[28]A plain text editor must be used to create or modify this file: don't use a word processor!

```
kern.sysv.shmall=262144 #shmmax / 4096
kern.sysv.shmseg=8192   #shm segments per user
kern.sysv.shmmni=8192   #shm id availables
kern.sysv.shmmin=1      #min size of shm segment
kern.sysv.shmmax=1073741824 #max size of shm segment
```

the symbol # precedes comments. After editing, the kernel must be reinitialized, therefore a computer restart is needed.

In a future, all these operations will be executed by installer scripts.

### 5.2.3 Things to do before launching the application

As described in section 5.1.7, once launched, the program checks for connected hardware and never do it when running, thus every device must be connected before (see pictures 5.7 or 5.8), in particular: the audio card and, if it is intended to do a real-time or recording session, the IP-camera.
Keep in mind that if there aren't connected audio cards with at least 32 inputs, *Realtime* and *Recording* activities will never be available.
For what concerns the IP-camera, currently only the two models described in section 4.1 are supported: in both cases they have to be configured with the fixed IP address 192.168.0.202,[29] without changing the default access credentials, all the other parameters can be set directly by 3D VMS.[30] At this point[31] a new connection has to be set up in the OS X® network preferences:

1. open *Network* from *System Preferences* window;
2. from *Locations* menu, choose *Edit locations...*;
3. press the small '+' button and give a name to the new location (ex. *IP-camera*);

[29]This can be done with the camera bundled software.

[30]There is an exception: the Sony® IPELA® camera frame ratio setting must to be 4:3, and this actually can be done only using its web interface. It has been experienced that if this parameter is accidentally set to 16:9, 3D VMS cannot detect correctly the device.

[31]It is assumed that the IP-camera is already connected to the laptop with the network cable and through the PoE power supply.

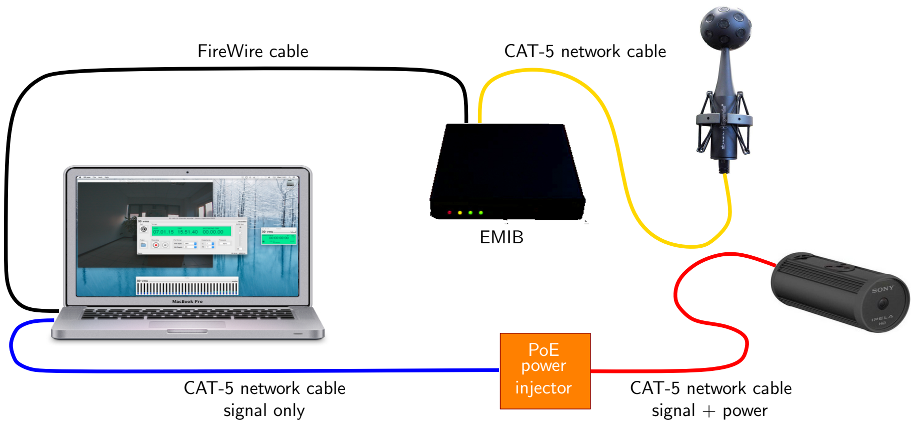

FireWire cable
CAT-5 network cable
EMIB
PoE power injector
CAT-5 network cable signal only
CAT-5 network cable signal + power

**Figure 5.7:** Hardware connections layout: Eigenmike® + external IP-camera.

4. from left panel, chose *Ethernet* device and in the right panel select *Manually* in the *Configure IPv4* menu, give a consistent fixed IP address, for example 192.168.0.200, and the subnet mask 255.255.255.0

5. press *Apply* button at the lower right;

since now, every time is needed the connection with the IP-camera, simply select the corresponding voice from *Locations* menu.

If the 3D VMS doesn't detect the camera, check the connection by issuing the *ping* command from a terminal:

```
ping 192.168.0.202
```

if the answer is *no route to host*, the connection has to be checked, otherwise, if the answer is positive, update camera firmware to the latest version.[32] If still 3D VMS doesn't recognize the camera, last attempt is to set manually - with cameras' web interface - the parameters as follows:

**Sony® IPELA®:** Single stream JPEG, frame rate 4:3, resolution 1024x768;

**Wision:** Single stream MJPEG, resolution 5 MPixel.

[32]This is especially recommended for Sony® IPELA® camera.

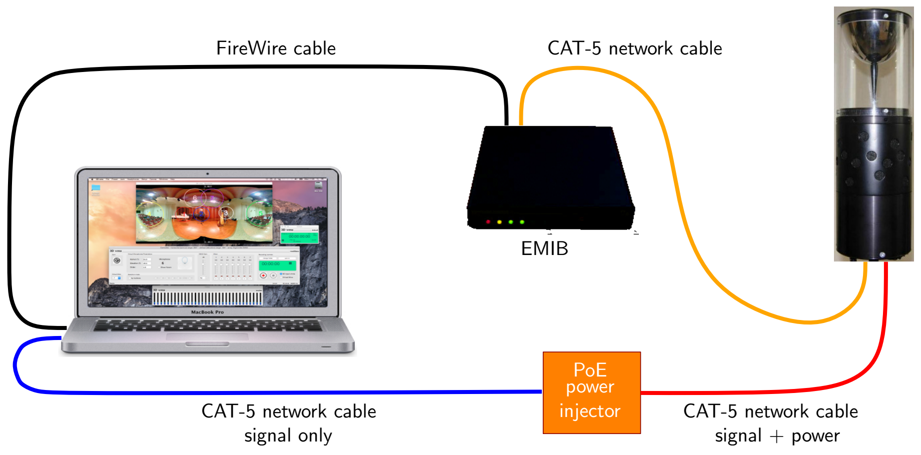

FireWire cable
CAT-5 network cable
EMIB
PoE power injector
CAT-5 network cable signal only
CAT-5 network cable signal + power

**Figure 5.8:** Hardware connections layout: IED Cylindrical Array (with embedded IP-camera).

### 5.2.4 The main dialog

When double-clicking on the application icon, after some waiting for matrix setups scanning and hardware checking, the main dialog (figure 5.3) should appear. In the right panel, below the logo picture, a status log can be found: here is reported if the EMIB, preferred audio interface, was found and the same for the IP camera; the controls are all placed on the left of which follows description, from top to bottom:

**Activity buttons** start the corresponding activity, *Playback*, *Realtime*, or *Recorder*; to be noted that last two can be deactivated if proper audio interface is not present;

**Array selection** menu from which the user can select the connected array in case of *Realtime* or *Recorder* activity or the array used for the recording that user wants to *Playback*[33]; the entries of this menu are the various matrix files/setups the system find in the default matrix directories;

**Interfaces selection** the system can use different interfaces for playing and recording activities: in these menus the more suitable interfaces should be listed after initial scan;

[33]At the time of writing, the system cannot detect which kind of array was used for a certain recording, thus if the wrong setup is chosen, then an incorrect filtering will have place during *Playback* activity and virtual microphones will not work as expected.

**Setup playback ports** this button opens a dialog that can be used to setup the default JACK routing for audio software components. Actually there is some work in progress on it and it will appear as shown in figure 5.9, allowing the user to set the routing for both playing and recording activities;

**Timecode** by checking the *Use timecode* flag, the timecode subsystem will be activated and the timecode display (figure 5.17) will be shown in *Realtime* and *Recorder* activities. Here it is possible to select the FPS used by the external synchronization system that will be connected to 3D VMS;

**Opening angle** using this radiobutton it is possible to force the acoustic horizontal range to a different, predefined value: usually this value is detected from the selected matrix/setup and as consequence of this selection some values can be disabled[34];

**Unwrap video stream** if checked, force the video frame unwrap. This option is checked and unmodifiable if a cylindrical array setup is selected;

**Upside-down placement** when checked, horizontal and vertical angles of virtual microphones will be inverted: as the name say, this option must used when the array is placed upside-down.

### 5.2.5 The Playback activity

The *Playback* activity is intended for post-processing of recorded files; as shown in figure 5.10 it's divided in two windows, one is the *control panel*, the other is the multimedia canvas.

Basically the user has to:

1. load the audio file and, optionally (but recommended, a static or dynamic (movie) background

2. press play;

[34]For example: a planar array cannot resolve a full 360° azimuth range.

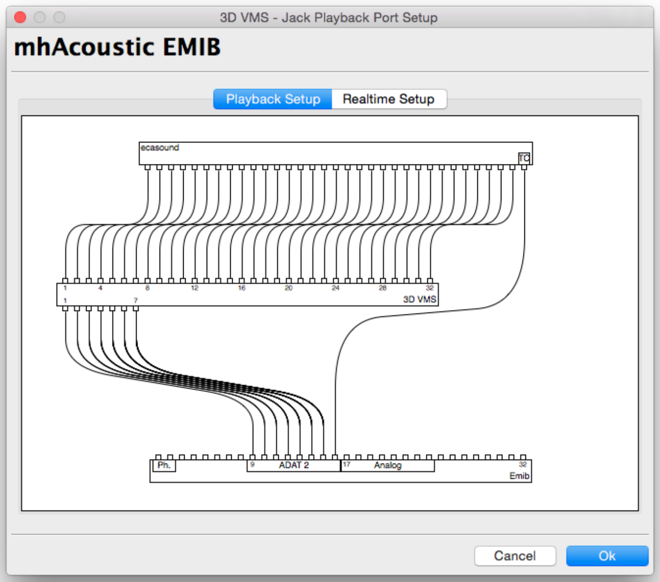


**Figure 5.9:** The 3D VMS default JACK routing dialog.

3. enjoy virtual microphones

Files can be loaded in some way:

- using the menus File/Load Audio... and/or File/Load Media...;
- using the menus Load Audio and/or Load Media buttons placed on the right of the control panel;
- dragging file icons on the media canvas or on the transport display;

in every case if audio and media file have the same name - and obviously different extension! - loading the media file, movie or photo, will load automatically the audio file.
Supported media files for static images are ISO Portable Network Graphics (.png) and ISO JPEG format (.jpg,.jpeg); for movies supported containers are

- Apple® QuickTime® File Format (.mov,.qt).
- Microsoft® Advanced Streaming Format (.asf, .wmv),

- Microsoft® Audio Video Interleave (.avi),
- ISO MPEG-PS (.mpg, .mpeg),
- ISO MPEG-TS (.ts, .tsv),
- ISO MPEG-4 (.mp4, .m4v)

with some of the most diffused codec (MPEG-2, M-JPEG and all codecs supported by the embedded *libav*[35]).

Audio file supported are W64, the default output for 3D VMS recordings, CAF, the Apple® CoreAudio format, and WAV but this is not very useful for its limits on size (see paragraph 5.2.7.1).

Once the files are loaded successfully, the green display on the right will be consequently updated with file names and play button will be enabled: now the user can start the audio (and movie) reproduction pressing play button and other transport controls will work as usual. Main transport commands are also available on menu Transport.

It is important to point out that the movies playback is implemented with GStreamer engine and since there aren't synchronization "hooks" to be used in conjunction with Ecasound, audio and video playback *are not* strictly synchronized, and the play/pause/stop commands are simply issued to the two subsystem one after the other. This is acceptable because the video part is only a reference for virtual microphones pointing and the delay between audio and video - if present - is really negligible for this kind of application.

During the playback, the multimedia canvas shows the loaded image or movie of the recording environment and overlaps the virtual microphones, represented by circles, to it: the user can drag the virtual microphones over the image, and can modify the microphone order using the mouse wheel or the corresponding gesture on the touchpad. It is also possible to modify with more accuracy the

[35]http://www.libav.org

position and order of virtual microphones acting on text controls of the control panel: write the desired angle, then press *Enter* on laptop keyboard to issue the command.
The gains of virtual microphones can be modified using the mixer controls on the control panel.
When there are all virtual microphones on the canvas and it get quite confusing, it is possible to activate the *Select by buttons* function by checking the square on control panel and use the radio-button at the side to select the corresponding virtual microphone: now every click on the canvas will correspond to a position change.

It is possible to save and load the virtual microphone spatial configuration to a text file[36] using the menus Preset/Save microphones preset and Preset/Load microphones preset respectively. It if also possible to load a microphone preset simply dragging its file icon on the media canvas.

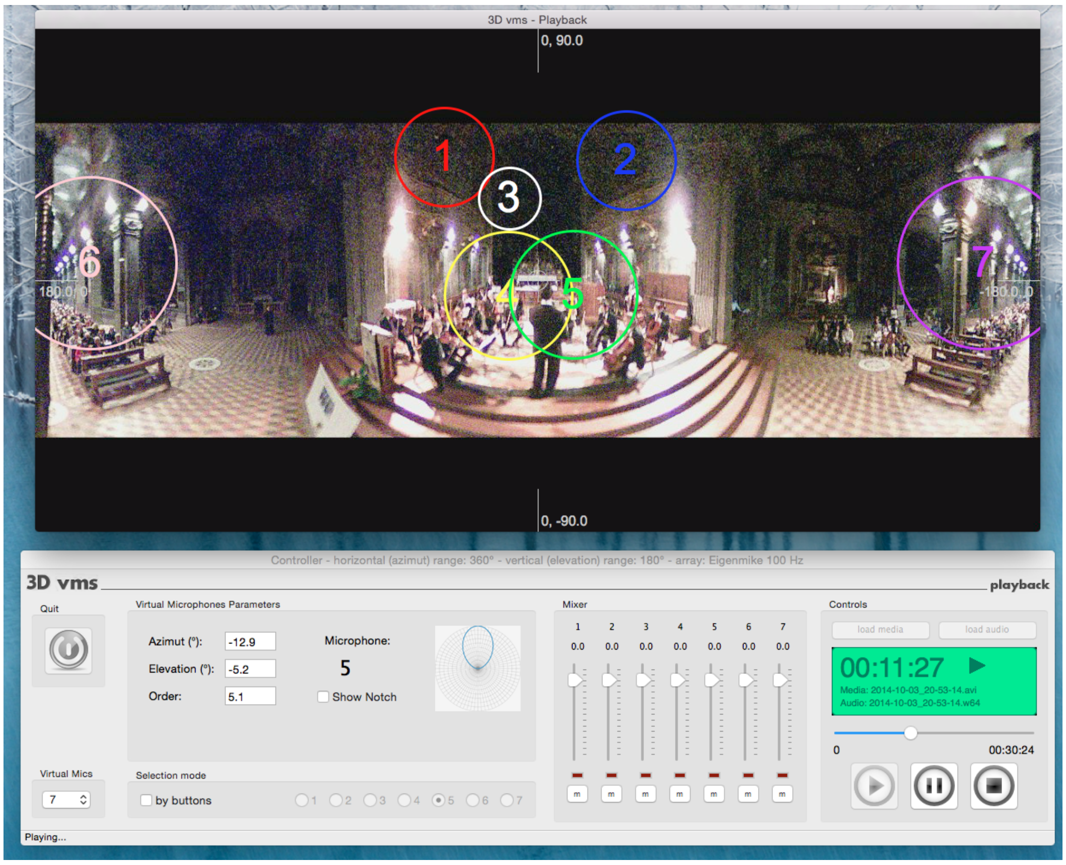
At every time, the user can select the number of virtual microphones using the control at the left of the control panel.

#### 5.2.5.1 JACK routing

Even without the help of the handy *JackPilot* application from *JackOSX* in every activity of 3D VMS it is always possible to modify the routing using the JACK connections dialog, available from menu JACK and shown in picture 5.11. Like the most known JACK routing applications, from this dialog it is possible to check current connections and to connect or disconnect ports by selecting both ends and pressing the button corresponding to desired action.

Every activity also is provided with a *shortcut*, accessible from menu Headphones, useful to quickly connect headphone ports to an array capsule or to a virtual microphone.

[36]Description in appendix A.2



Controller - horizontal (azimut) range: 360° - vertical (elevation) range: 180° - array: Eigenmike 100 Hz

Azimut (°): -12.9
Elevation (°): -5.2
Order: 5.1
Microphone: 5
Show Notch

Virtual Mics: 7
Selection mode: by buttons — 1 2 3 4 [✓ 5] 6 7

Mixer: 1 0.0 · 2 0.0 · 3 0.0 · 4 0.0 · 5 0.0 · 6 0.0 · 7 0.0

00:11:27
Media: 2014-10-03_20-53-14.avi
Audio: 2014-10-03_20-53-14.w64
0 00:30:24

Playing...

Figure 5.10: The 3D VMS *Playback* activity.

### 5.2.6 The Realtime activity

The main windows layout of the *Realtime* activity is similar to that of *Playback*: a control panel, plus a multimedia canvas; but here the input doesn't come from a file: audio and video - if a camera is present - are real-time captured by the connected hardware devices and processed by the filtering engine, so it is given to the user the feeling of moving a *real* microphone in the pointing space having immediately the feedback of what happens. When shooting a show, for example, it is possible to follow an actor moving on the stage, like a spot-light, without losing the acoustic focus on his voice and without the use of radio microphones or similar.[37]

[37]For this kind of application the latency time is absolutely critical and it has to be the smallest possible.

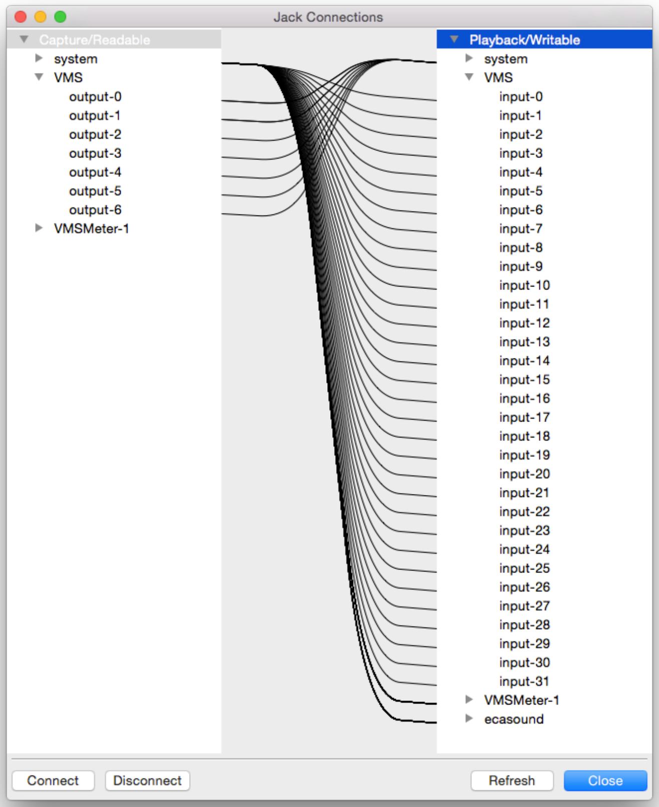


**Figure 5.11:** The JACK connections dialog. Here are exploded the 3D VMS filtering engine input ports (on the *playback* side) and the outputs, also known as *virtual microphones*, on the *capture* side.

The control panel gives to the user roughly the same functions of the *Playback* activity with the substitution of the play/pause buttons with a *record* button; the user can also decide whether to record the 32 array channels row, or the 3D VMS filtered signals, the so-called *virtual microphones*.

Being an *acquisition* activity, a slider to control the gain of Eigenmike® "family" of array is provided; to be noted that the gain updating of these devices is a slow process because it is implemented with a MIDI protocol, thus a delay can be noticed from slider movement to effective gain change.

If a camera is connected, pressing *Record* will record not only the audio, but

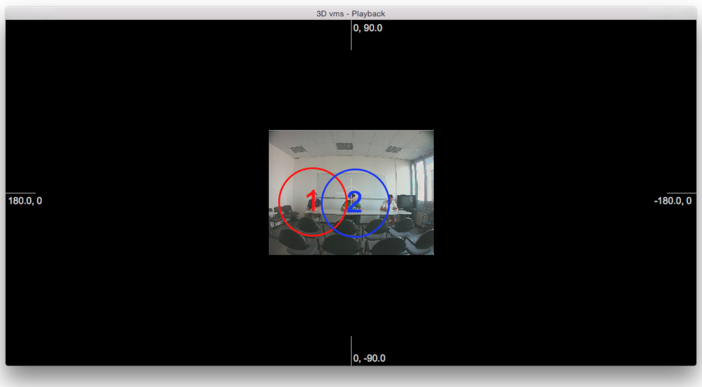


**Figure 5.12:** Using a spherical array with a non panoramic camera system there is a big difference between *acoustic* and optical angles: the visual reference is very small and is difficult to point virtual microphones on the speakers.

the video stream too; if not differently specified using the *change folder* button, all files will be stored in the default directory $HOME/EigenmikeRecordings, soon deprecated in favor of $HOME/vms/recordings.

A *zooming* function for the multimedia canvas had also been recently implemented: this was needed because often it happens that the acoustical angular ranges of the microphone array are different from the optical ones of the connected camera; to leave to the user the full acoustical ranges for virtual microphone operations and since these are often much larger than optical aperture, the movie frame appears small to the user and surrounded by a black area (figure 5.12). With the *zoom* feature, the user can enlarge the movie frame, reducing the visible acoustic range, but making easier to point virtual microphones, as can be seen in figure 5.13 or 5.14.

As accessory window, to check the array capsules level a *Level Meter* frame is also present (figure 5.15): it is a JACK client connected directly to the capture audio interface ports.

If chosen from the main application dialog, it is possible to do a timecode-synchronized recording session: a small dialog window that shows (figure 5.17)

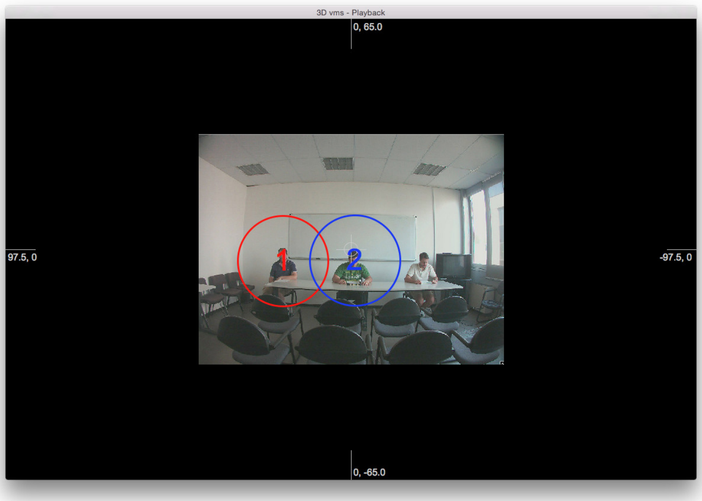


**Figure 5.13:** Increasing the zoom level make easier to point virtual microphone, but the *acoustic space* available for visual pointing has been reduced, as can be seen from the angular bounds reported on the sides of the canvas.

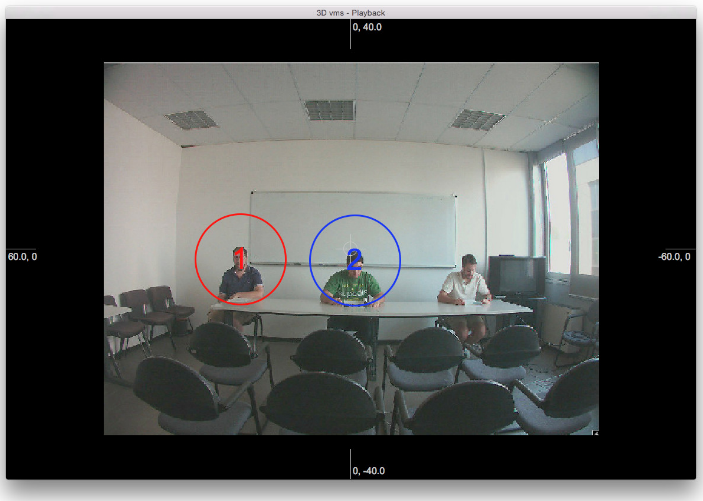


**Figure 5.14:** Zooming further, the virtual microphones circles don't overlap and the canvas appears less confusing, but the *acoustic space* is far smaller than the full angular range of a spherical microphone.

timecode status will appear when starting the activity, and the *Sync* button of the control panel will be enabled. The timecode synchronization has to be done every time the user opens a *Realtime* activity session from the main application

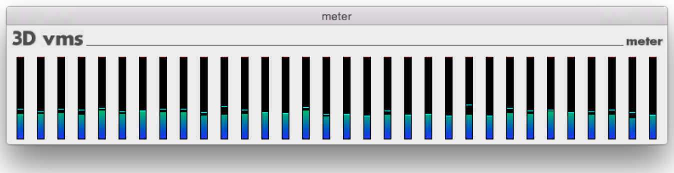


**Figure 5.15:** The 3D VMS meter monitoring array capsules output levels.

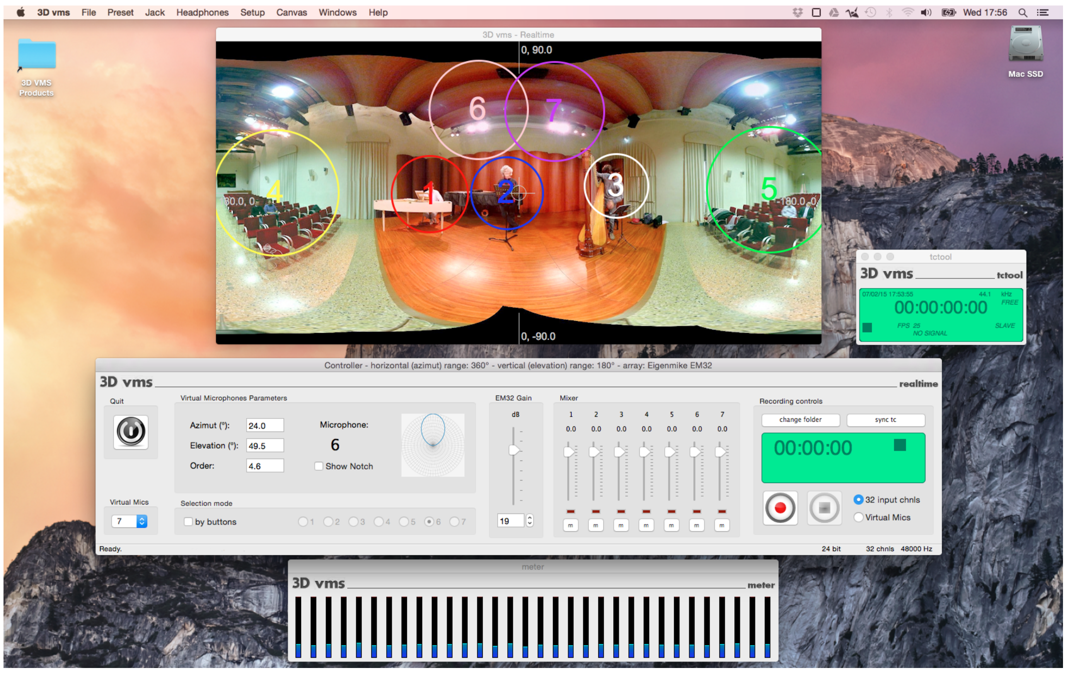


**Figure 5.16:** A 3D VMS *Realtime* session with VU-meter and timecode display.

dialog following the procedure described at the paragraph 5.1.5.

### 5.2.7 The Recorder activity

In professional application, when is preferred to do the virtual microphones steering in post-processing, a simple ad robust application that can record the raw 32 channels of the microphone array with no filtering at all, can be a better choice than the complex *Realtime* activity; the *Recorder* activity was written with this kind of task in mind, a *rec-button* only application that records 32 tracks (figure 5.19).

By users choice, the application can also record the video incoming from the ar-

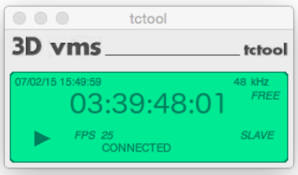
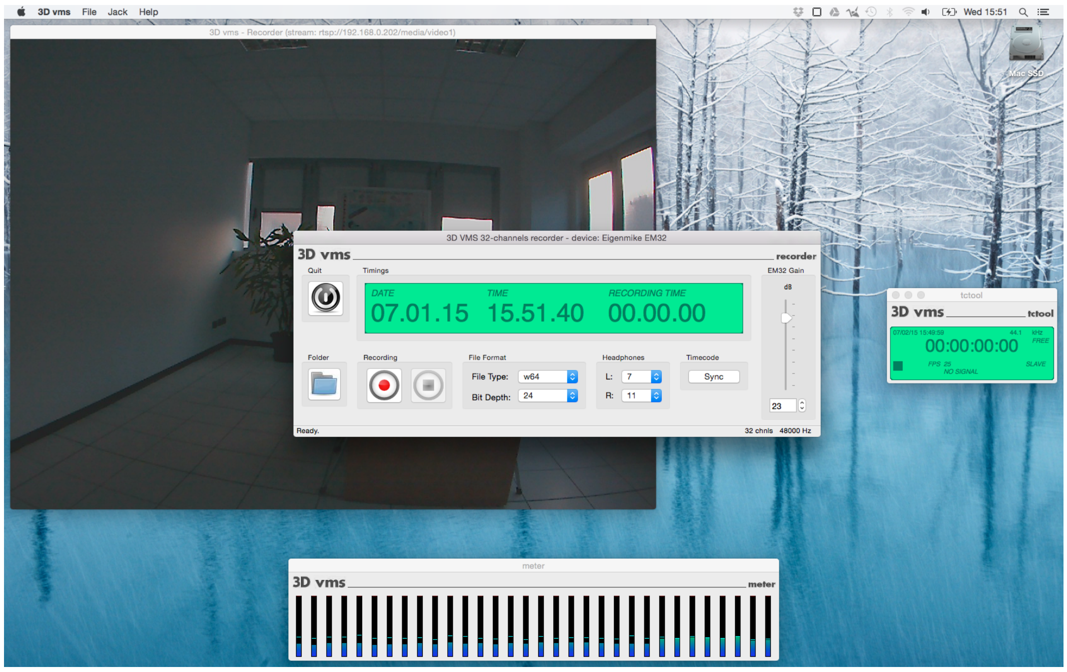


**Figure 5.17:** The timecode display.



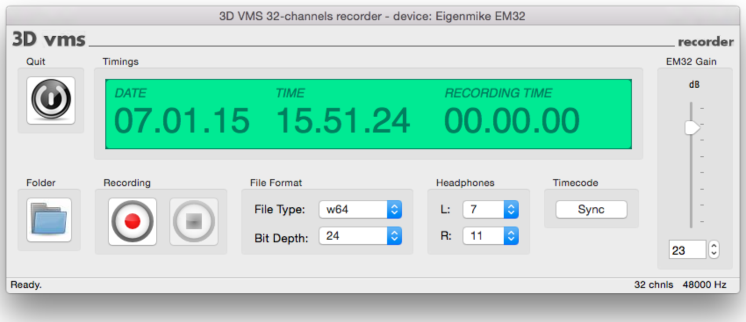
File Type: w64
Bit Depth: 24
L: 7
R: 11
23

**Figure 5.18:** A 3D VMS *Recorder* session with camera monitor, VU-meter and timecode display.

ray camera; in this case an additional window is shown that works as camera monitor.
Like the *Realtime* activity, it is possible to do a timecode-synchronized recording with the same modalities.

#### 5.2.7.1 Output file formats

The default output file format for audio recording is the *Sony*® *Sound Forge*™ *Wave 64*, with extension W64: it is an extension of the *Microsoft*® *WAV* format that overcomes its intrinsic 4 GB file length limit, having a full 64 bit header.



DATE 07.01.15 TIME 15.51.24 RECORDING TIME 00.00.00

File Type: w64
Bit Depth: 24
L: 7
R: 11
EM32 Gain: 23

Ready.
32 chnls 48000 Hz

**Figure 5.19:** The 3D VMS *Recorder* activity.

The system can also output WAV files, but for a non-timecoded recording, the length will be limited to about 10 minutes.
Another output options, without any kind of limitation, is the Apple® *CoreAudio Format*, with extension CAF.

# Chapter 6

# Endings

It is since 2009 that RAI broadcasting company employs a 3D VMS system in show production, and if the first times the task was, in fact, an experimental application of a research project, now the system has gained in stability and reliability, becoming a sort of acoustic *swiss knife* that really can save some tens of meters of cables and the placement of other real microphones in non movable positions and with static polar shapes. Now, when, for example, the audio editing team has available the 32 raw recorded signals from array capsules, they can place the virtual microphones and decide the better polar pattern in post-production, and if something doesn't sound good, they simply have to modify virtual microphone position and order.

Or also it is possible to take full advantage of the realtime features for a live broadcast production, using 3D VMS virtual microphone in place of 7 physical microphones: the first RAI production that employed this technology was the opera *Lucia di Lammermoor* broadcast live by Rai-Radio3 from Theater Regio of Turin the 21th of March 2011.

To achieve these results, the software part requested an accurate and long work, as described in chapter 5 because if it's certainly true that BruteFIR, Ecasound, GStreamer are excellent and intrinsically robust programs, to manage the concurrent execution of them and to maintain the entire system in a consistent state in case of failure of one of the *sub*processes - that are *always* independent

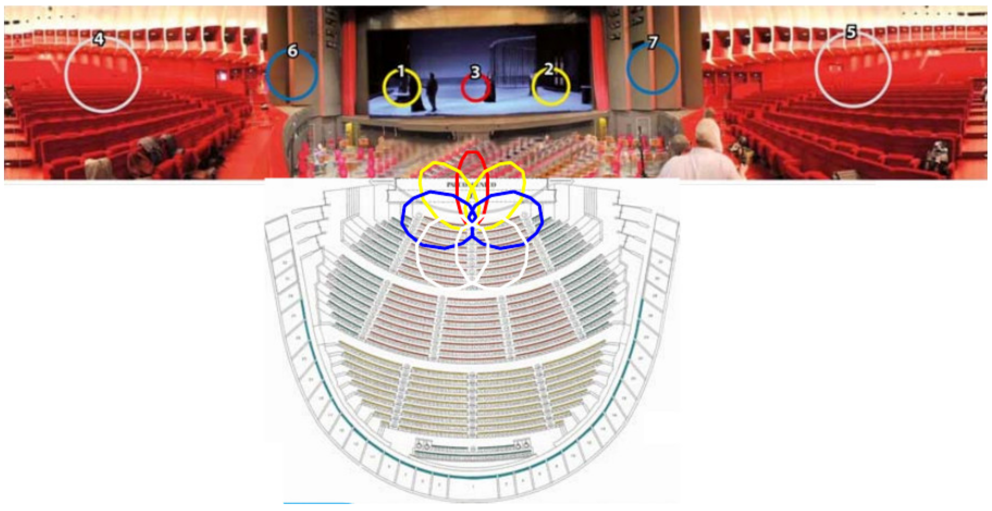


**Figure 6.1:** Example of post-processing with 7 fixed microphones (*La Bohème - Theater Regio, Turin, 20 May 2010*).

- it is absolutely not easy: still today some issues are not completely resolved, mainly in cases of incompatible startup setup.[1] Currently the system watches all running processes and when an error status is detected, the activity is paused or stopped and the user is warned, then a rearm is tried, but there are cases in which the behavior of a process in error status is not completely predictable and the safer way to restore a consistent state is to restart the main process.

In general terms, the program restart as an error recover procedure is a bad solution, plus 3D VMS can have a long initialization phase on startup: what has been done to avoid these uncomfortable status is a careful software architecture design aimed to minimize runtime error status and numerous controls on Graphical User Interface options to prevent status with unpredictable behavior.

The feedback from RAI technicians are always very precious in this sense,[2] showing that the road taken was right, thus now it is possible to consider 3D VMS maybe still *beta*, but *reliable*, also in very chaotic situations like a mobile live broadcast set.

[1]For example: the user tries to start a cylindrical array setup when a camera with different resolution than expected is connected to the laptop. Actually the system warns the user about the incompatibility, and if one activity is launched, not always the video frame adaptation is successful.

[2]Especially when some time is given to work on bugs…

## 6.1 Future developments

In the next future, there are a lot of feature requests for 3D VMS, here a partial list:

- addition of other polar shapes for virtual microphones, for example the figure-of-eight;
- addition of recording system related information in the output files as metadata;
- increase the number of available virtual microphones;
- VST version of the system;
- version for iOS®/Android™ based portable devices of the playback application.

Whilst the first and the second are already intrinsically available in the filtering engine and the audio file manager library respectively,[3] the others require a much more radical action: as described in section 5.1.2, the limit of 7 virtual microphones is due to a limit of the software BruteFIR, therefore increase this number implies the complete replacement of the filtering engine. The extension of BruteFIR engine is not taken in account because its multiprocess based architecture is to be considered obsolete, too much memory consuming and slow to setup[4] compared to modern multithreaded applications.
An attempt was done in 2012 with the software Jconvolver[5] by Fons Adriaensen, a JACK based multithreaded library that features up to 64 inputs and 64 outputs, quite easily embeddable in the 3D VMS architecture, but the needing of a more portable solution and the needing, at the time, to strengthen the mainstream system rather than writing new modules stopped the tests.
An apparently good alternative is represented by the filtering engine of XVolver[6]

[3]For W64 file format the library used is *libsndfile* by Eric de Castro Lopo (http://www.mega-nerd.com/libsndfile).

[4]The time needed to start a new process is effectively longer than the amount of time needed to start a new thread.

[5]http://kokkinizita.linuxaudio.org

[6]http://www.desknotes.it/siten/Software.html

software by Carlo Chiari, already written with portability in mind and for this reason can be a valuable base for a VST version of the software. The cons is the absence of a dynamic filter change feature that has to be implemented.
With portable devices, the computational power available and storage space are limited, for some kinds of devices *very* limited: in this case the challenge is to individuate a good trade-off between resources available on-device and online, for example *pre-filtered* tracks, balancing streaming and CPU loads in order to obtain a responsive feedback for the user.

# Appendix A

# File format descriptions

In the following sections a complete descriptions of file formats used by 3D VMS software.

## A.1 Matrix files

Since the very first versions of the software the measured impulse responses of the array were stored in a dat binary format used by Matlab® to store its data. First array[1] measurements have a fixed, high, number of positions or pairs azimuth/elevation, 684, as described in section 3.1.2. Now this number is not anymore fixed, but the dat has been paired (= having the same name) with a dir in which are stored the measured positions.
So, actually if the system find a not-paired dat file, it assumes that the file contains all the 684 "standard" positions.
An additional extension was done in order to store not only the raw impulse responses, but other array related metadata, contained in an xml file that mus be placed in the same directory where dat and dir files reside. This file must have the name metadata.xml and it can contain a complete 3D VMS configuration.

[1]Essentially the Eigenmike®, a work done by Eng. L.Chiesi.

### A.1.1 The file metadata.xml

All 3D VMS related tags must be enclosed in a ThreeDvmsConfigurationMetadata tag; inside the following tags are recognized:

**Label** (string) the configuration label: this is the name that will appear as main dialog array selection menu entry;

**MatrixFileName** (string) dat file name without extension[2]; it must be placed in the same directory where metadata.xml is;

**Device** (string) the array type; recognized strings are: Eigenmike, Planar Array and Cylindrical Array. It is mainly used to guess the correct camera setup;

**AngularWidthDegrees** (float) the horizontal (azimuth) angular range in degrees;

**AngularHeightDegrees** (float) the vertical (elevation) angular range in degrees;

**CanvasWidthPixel** (int) the media canvas width in pixels;

**CanvasHeightPixel** (int) the media canvas height in pixels;

**MaxOrder** (float) the maximum virtual microphones order;

**CameraStream** (string) the full address of rtsp camera stream;

**UseDirections** (string: yes/no) use or not the directions file. If 'yes' a file vith the same name of that specified in the MatrixFileName field, but with extension dir must be present in the same directory of metadata.xml;

**UnwrapFrame** (string: yes/no) unwrap or not the frames coming from camera. If 'yes' the unwrap parameters of GStreamer module can be specified as tag attributes (referencing to picture 4.8:

- outWidth (int, $0 \div 65535$) output image width
- outHeight (int, $0 \div 65535$) output image height

[2]Spaces are not allowed

- t0 (float $-6.28 \div 6.28$) wrapped image center angle $\theta_0$ in radians
- rMin (float, $0 \div 1$) minimum wrapped image normalized radius $R_{\text{min}}$
- rMax (float, $0 \div 1$) maximum wrapped image normalized radius $R_{\text{max}}$
- a (float, $-10 \div 10$) 2nd order vertical axis unwrapping function coefficient
- b (float $-10 \div 10$) 1st order vertical axis unwrapping function coefficient
- c (float $-10 \div 10$) 0th order vertical axis unwrapping function coefficient
- xc (float, $0 \div 1$) wrapped image center $x$ normalized coordinate
- yc (float, $0 \div 1$) wrapped image center $y$ normalized coordinate

Example:

```
<ThreeDvmsConfigurationMetadata>
    <Label>Cylindrical Array</Label>
    <MatrixFileName>Cyl_R_70HZ_2048</MatrixFileName>
    <Device>Cylindrical Array</Device>
    <AngularWidthDegrees>360.0</AngularWidthDegrees>
    <AngularHeightDegrees>100.0</AngularHeightDegrees>
    <UseDirections>yes</UseDirections>
    <UnwrapFrame outWidth="1280" outHeight="400" t0="-2.3">yes\
         </UnwrapFrame>
</ThreeDvmsConfigurationMetadata>
```

## A.2 Microphones presets

Microphones presets can have the extension txt or prs, but in any case they are plain text files containing the configuration of all current virtual microphones as tab-separated values table.
Every line is referred to a single virtual microphones and contains the following parameters:

- *azimuth* angle in degrees

- *elevation* angle in degrees
- *order*
- *gain* in deciBels.

To be noted that the file will have as many lines as the number of virtual microphones currently available in the application.
If the user loads a preset with a smaller number of lines than virtual microphones, the parameters of microphones *in excess* will not be updated.

# Appendix B

# LTC/SMPTE

Generally speaking, *SMPTE timecode* is a set of cooperating standards to label individual frames of video or film with a time code defined by the *Society of Motion Picture and Television Engineers* in the SMPTE 12M specification.

Timecodes are added to film, video or audio material, and have also been adapted to synchronize music. They provide a time reference for editing, synchronization and identification and, practically, is a form of media metadata. The invention of timecode made modern videotape editing possible, and led eventually to the creation of non-linear editing system.[1]

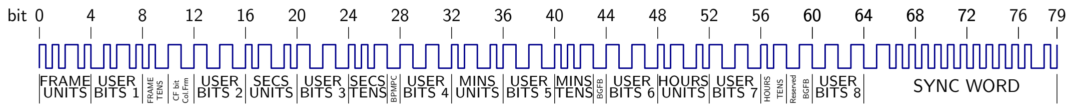


**Figure B.1:** An example of SMPTE frame.

SMPTE timecodes contain binary coded decimal hour:minute:second:frame identification and 32 bits for use by users; an example of the signal is shown in figure B.1 but, in this form, is not used for timecode distribution and storage. Instead the *Linear (or Longitudinal) Time Code (LTC)* encoding is used, that, transforming the binary SMPTE signal in an audio signal, can be stored without distortion in the old, fully analog, VTRs.

[1]See [22].

For backward compatibility and for reliability reasons, the analog LTC/SMPTE signal is still today the standard for timecode distribution in every broadcast set.

# Bibliography

[1] S. Berge and N. Barrett, *High Angular Resolution Planewave Expansion* , Proc. of the 2nd International Symposium on Ambisonics and Spherical Acoustics , (2010).

[2] A. J. Berkhout, D. de Vries, and P. Vogel, *Acoustic control by wave field synthesis* , J. Acoust. Soc. Am., 93(5) (1993), pp. 2764–2778.

[3] M. M. Boone, E. N. G. Verheijen, and P. F. van Tol, *Spatial Sound-Field Reproduction by Wave-Field Synthesis* , J. Audio Eng. Soc., 43 (1995), pp. 1003–1012.

[4] J. J. Christensen and J. Hald, *Beamforming* , Bruel & Kjær Technical Review, 1 (2004).

[5] J. Daniel, S. Moreau, and R. Nicol, *Further Investigations of High-Order Ambisonics and Wavefield Synthesis for Holophonic Sound Imaging* , in 114th AES Convention , mar. 2003.

[6] W. L. Dooley and R. D. Streicher, *M-S Stereo: a Powerful Technique for Working in Stereo* , in 69th AES Convention , may 1981.

[7] A. Farina, *Simultaneous measurement of impulse response and distortion with a swept-sine technique* , in 110th AES Convention , Paris, feb. 2000.

[8] A. Farina, A. Amendola, L. Chiesi, A. Capra, and S. Campanini, *Spatial PCM Sampling: a new method for sound recording and playback* , in 52th AES Conference, Guildford, sep. 2013.

[9] F. M. Fazi and P. A. Nelson, *The ill-conditioning problem in Sound Field Reconstruction* , in 123th AES Conference , New York, oct. 2007.

[10] M. Frigo and S. G. Johnson, *FFTW: An Adaptive Software Architecture for the FFT* , Proceedings of the 1998 ICASSP conference , 3 (1998), pp. 1381–1384.

[11] B. Ginn, J. J. Christensen, J. Hald, J. M. rkholt, A. Schuhmacher, and C. Blaabjerg, *A review of array techniques for noise source location* , in ICSV 10 , Stockholm, jul. 2003.

[12] A. Greensted, *Delay sum beamforming — the lab books pages*, 2012. http://www.labbookpages.co.uk/audio/beamforming/delaySum.html, [Online; accessed 13-January-2015].

[13] J. Hald, *STSF - a unique technique for scan-based Near-field Acoustic Holography without restrictions on coherence* , in Bruel & Kjær Technical Review, vol. 1, 1989.

[14] J. Hald, *Combined NAH and Beamforming Using the Same Array* , Bruel & Kjær Technical Review, 1 (2005), pp. 15–43.

[15] J. Hald, *Patch Near-field Acoustical Holography Using a New Statistically Optimal Method* , Bruel & Kjær Technical Review, 1 (2005), pp. 44–54.

[16] O. Kirkeby and P. A. Nelson, *Digital Filter Design for Inversion Problems in Sound Reproduction* , J. Audio Eng. Soc., 47 (1999).

[17] O. Kirkeby, P. Rubak, and A. Farina, *Analysis of ill-conditioning of multi-channel deconvolution problems* , in 106th AES Convention , Munich, may 1999.

[18] J. Maynard, E. Williams, and Y. Lee, *Nearfield acoustic holography: I. Theory of generalized holography and the development of NAH* , J. Acoust. Soc. Am., 78 (1985), pp. 1395–1413.

[19] S. Moreau, J. Daniel, and S. Bertet, *3D sound field recording with High Order Ambisonics -objective measurements and validation of a 4th order spherical microphone* , in 120th AES Conference , Paris, may 2006.

[20] V. Pulkki, *Spatial Sound Reproduction with Directional Audio Coding* , J. Audio Eng. Soc., 55 (2007), pp. 503–516.

[21] A. Torger and A. Farina, *Real-Time Partitioned Convolution for Ambiophonics Surround Sound* , in 2001 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics , New York, oct. 2001.

[22] Wikipedia, *Smpte timecode — wikipedia, the free encyclopedia*, 2014. http://en.wikipedia.org/w/index.php?title=SMPTE_timecode&oldid=639775488, [Online; accessed 13-January-2015].

[23] W. Woszczyk, B. Leonard, and D. Ko, *Space Builder: An Impulse Response-Based Tool for Immersive 22.2 Channel Ambiance Design* , in 40th AES Conference , Tokyo, oct. 2010.

*Questa la scrivo in italiano.*
*È normale: quando capita fra le mani una qualsiasi tesi, ed ora come non mai se ne producono in quantità, si d‘a una scorsa all'indice, poi molto frettolosa al testo, ma alla fine si capita sempre lì, alla pagina dei "ringraziamenti". Dopo pagine e pagine di contenuti dalle ambizioni scientifiche dalle quali l'autore poco lascia trapelare di sé, si va in cerca di calore umano, di quei nomi, di quelle esperienze che hanno accompagnato la vita di colui che nella prima pagina si presenta come "candidato". Magari pensando di vedersi citati e provando una piccola emozione quando, scorrendo l'elenco di nomi, si trova il proprio.*
*La pagina dei ringraziamenti è, nella sua essenza, un luogo romantico, dove si può addirittura trovare conforto e serenità nello scoprire di aver avuto un ruolo, talora inaspettatamente importante, nella vita di qualcun altro.*
*Non mi sottrarrò al rito: dopotutto ringraziare è una delle prime regole di buona educazione che ci viene insegnata ed è indiscutibilmente bello sentirsi dire "grazie", non ha grande importanza il perché. Ciò che conta è dirne tanti di "grazie", poiché a nessuno è dovuto proprio nulla e tutto, in fondo, è un dono.*

*Dunque grazie, prima di tutto, ad Angelo Farina ed al suo geniale entusiasmo al di fuori di qualsiasi controllo, poi a tutti coloro con cui ho collaborato in questi anni di assegni Unipr e di cui non mancano i contributi nelle pagine precedenti: Enrico, Christian, Alberto, Andrea (ora in RCF, ma il prototipo del software l'ha fatto lui), Lorenzo C. (difficile elencare tutti i suoi contributi in questo progetto di ricerca), Lorenzo E. e Luca N.*
*Grazie anche al CRIT-RAI nella persona di Leonardo Scopece, senza cui questa ricerca non sarebbe cominciata e grazie al quale c'è tutta l'intenzione di proseguirla: delle telefonate ne avrei fatto volentieri a meno, ma se il software ha raggiunto certi obiettivi una gran parte del merito va a chi lo utilizza e ci crede.*

*L'occasione è propizia per dire grazie agli amici, che non solo è d'uso, ma esprime ciò che realmente avviene nella vita delle persone: quello che si fa è sempre frutto di un contesto e di esperienze negli ambiti più vari: io sono musicista e dirigo un coro. Non fossi stato un musicista, probabilmente non sarei mai entrato in contatto con il gruppo di acustica dell'Università. Dunque voglio ringraziare tutti i miei coristi, perché amici. Grazie anche a loro il legame con la musica in* *questi anni è rimasto forte e, nonostante le inevitabili difficoltà, le soddisfazioni ed il divertimento non sono mai mancati.*

*Mi piace ringraziare Michele, da diversi anni negli USA, non ci sentiamo praticamente mai, ma la sua "lezione" e soprattutto la sua passione sono presenti in ogni istante. Indimenticabili.*

*Non posso certo dimenticare i miei genitori, co-dedicatari di questo testo, senza i quali non sarebbe iniziato proprio nulla, ma l'ultimo, il più importante "grazie" va a Giovanna, che più di ogni altro mi ha spinto nel conseguire questo dottorato nonostante i miei piagnistei: è lei il primo ed il più energico motore di tutto ciò che faccio ed è grazie a lei se la nostra vita è una continua "ricerca" di musica.*

Typesetted on 2015 january 16 with $\LaTeX\,2_\varepsilon$