08

TR－A－0068

Study on Functional Inner Hair Cell Model
内有毛細胞機能モデルの検討

Eric TARDIF and Tatsuya HIRAHARA
エレキ・ターディフ 平原達也

1990. 2. 2

ATR視聴覚機構研究所
〒619-02 京都府相楽郡精華町乾谷 ☎07749-5-1411

ATR Auditory and Visual Perception Research Laboratories
Inuidani, Seika-cho, Soraku-gun, Kyoto 619-02 Japan
Telephone: +81-7749-5-1411
Facsimile: +81-7749-5-1408
Telex: 5452-516 ATR J

Study on Functional Inner Hair Cell Model

Modification for Adaptive Q cochlea filter Bank

and

Improvement of its Noise Reduction Property

2 .2.1990

Eric Tardif

and

Tatsuya Hirahara

CONTENTS

1. Introduction

Despite a lot of effort put into building a machine which recognizes human speech as humans do, there is as yet no other system which processes speech with the precision and versatility of the human auditory system. One reason that good performance can not be achieved might be due to the spectrum analysis method used. In many systems, the short term power spectrum based on Fourier Transformation or coefficients obtained by the LPC (Linear Predictive Coding) analysis have usually been used as input vectors.

These traditional spectral analysis methods are well defined mathematically and very easy to use. However, they give a very different speech spectrum representation from the real speech signal representation in the auditory pathway because the auditory spectral analyzer is essentially a nonlinear system while the traditional spectral analyzers are linear systems. Hence, we believe that better results could be obtained if we had a spectral analysis method which simulates the signal transformation functions in the human auditory system.

The superiority of nature's design has been an interesting subject for intensive studies. A number of physiological and psychophysical studies have been performed on the auditory system [G.von Bekesy (1960), E.C.Carterette & M.P.Friedman (1978), B.C.J.Moore (1982), J.O.Pickles (1982) R.Carlson and B.Granstorm (1982), J.V.Tobias & E.D.Schubert (1983), J.B.Allen *et al.* (1985), C.Berlin (1985), A.R.Moller (1983, 1986), G.M.Edelman *et al.* (1988)].

Those results partially clarify how the auditory functions work, but many things still remain obscure or unknown, particulaly for the higher level auditory system. The auditory peripheral system gives us reliable information and data, such as the sound spectral transformation which takes place in the cochlea, where nonlinear spectral analysis is performed with a basilar membrane system and a hair cell system. Inner hair cells are the sound-to-impulse transducers, in which basilar membrane displacement is transformed into nerve impulse trains. Using this knowledge and data, many auditory models have been developped, some of which try to simulate the signal processing mechanisms or functions in the cochlear. Good surveys may be found in J.Ujihara (1976), J.B.Allen (1985), *Proceedings of the Montreal Symposium on Speech Recognition* (1986), and S.Greenberg (1988).

In this report, we are going to describe a functional model of the auditory peripheral system. We already have a nonlinear cochlear filter model which simulates the adaptive Q filtering characteristics of the basilar membrane system (T. Hirahara *et al.*, 1989). The spectrograms obtained by this model are encouraging. They always give appropriate speech spectrum representation compared with the traditional linear cochlear filter-bank or DFT spectrogram. Therefore, the next step is to put the inner hair cell model after this nonlinear cochlear filter model.

We focused on a computational inner hair cell model proposed by S. Seneff, since her model simulates principal inner hair cell functions very well with simple circuits. First, we trace her model precisely so as to find any advantage and/or disadvantage of the model. Next, we describe some modification of the model so as to use it with the nonlinear cochlear filter model. Then we examine the properties of the complete cochlear model, i.e. a nonlinear cochlear filter model followed by a nonlinear inner hair cell model. Finally, the application of this model to speech with noise added is studied and an improvement proposes.

2. The Peripheral Human Auditory System: anatomy and description

In this chapter a general review of the current understanding of some aspects of the human auditory system is presented. This review is necessarily limited in scope, and biased towards those aspects of the system that are most relevant for this report. The main purpose of the chapter is to provide motivation for the detailed design structure of the feature implemented computer model described in this technical report.

2.1. Basic structure and function of the peripheral auditory system.

The human peripheral auditory system consists of the outer, middle and inner ears, as schematized in figure 2.1.1. Sound travels past the pinna and down the auditory canal of the outer ear and hits the eardrum, or tympanic membrane, causing it to vibrate. The middle ear, containing three small bones, the malleus, incus, and stapes, acts as an impedance matching device to ensure efficient transfer of sound from the air to the cochlear fluid in the inner ear.

The smallest and last bone of the middle ear, the stapes, makes contact with the oval window at the base of the cochlea in the inner ear. The cochlea is a small coiled tube of approximately 2.5 turns, with a length of 3.5 cm. It has a bony rigid walls and is filled with incompressible fluids. It is divided along its length by two flexible membranes, Reissner's membrane and the basilar membrane. The basilar membrane grows in thickness from the base of the cochlea to the apex at the inner end of the coil. The membrane is about 100 times as stiff at the base as it is at the apex. Different portions of the membrane respond preferentially to different frequencies, with response to high frequencies being dominant at the stiff or basal end, and response to low frequencies exhibiting amplitude maxima at the apex. Thus the basilar membrane performs a frequency analysis of the incoming signal,

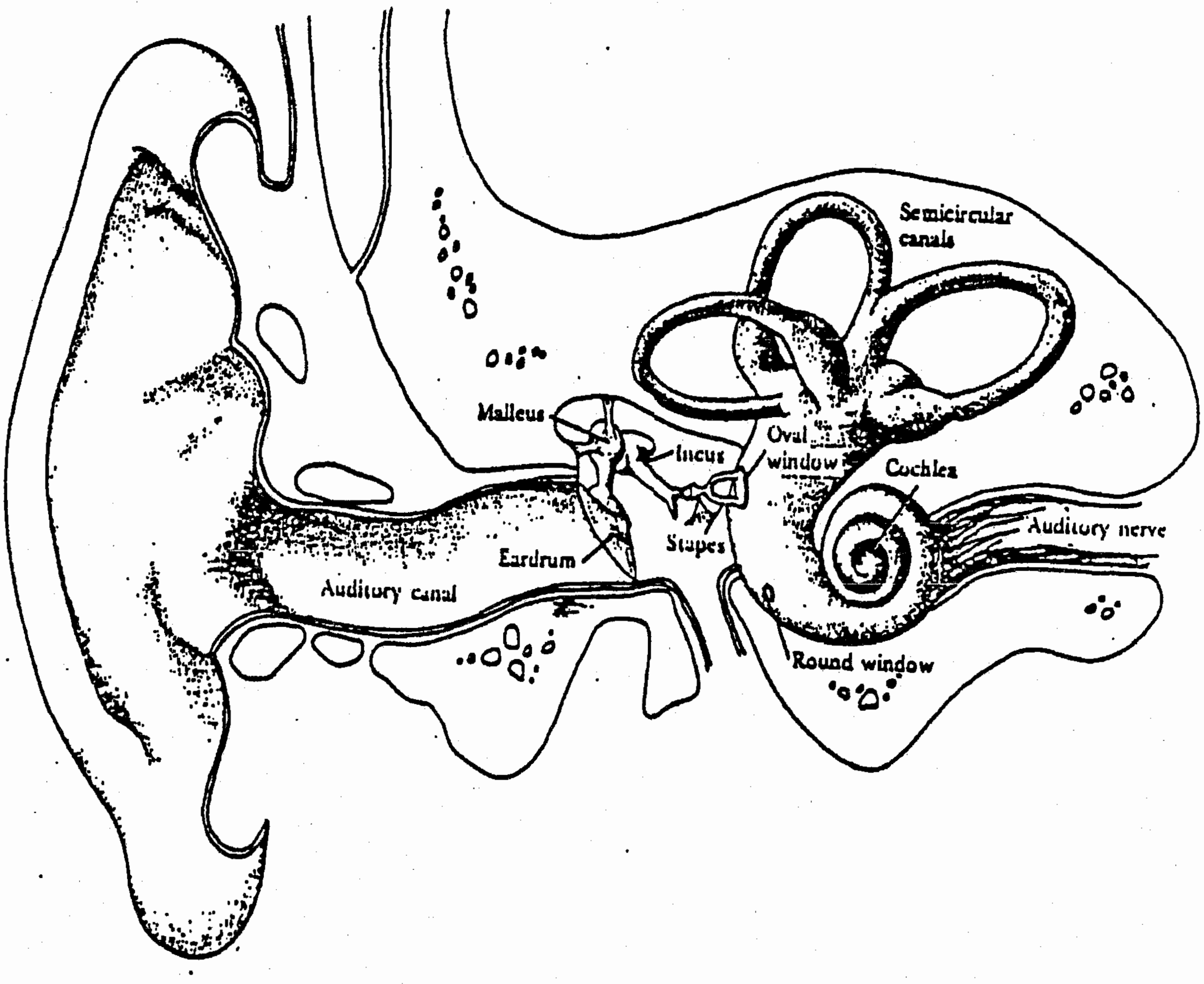


figure 2.1.1 : Illustration of the structure of the peripheral auditory system, showing the outer, middle and inner ear. (from MOORE, 1982, p.14).

where the frequency scale is mapped into the place dimension of the membrane.

Attached to the basilar membrane is the organ of Corti, a complex organ in which are contained the inner and outer hair cells, separated by an arch known as the tunnel of Corti. The outer haircells outnumber the inner ones by a factor of seven; however, the role of the outer cells remain obscure, although their role in feedback is somewhat understood. There are about 3500 inner hair cells, each with about 40 hairs. Action potentials are generated on a given auditory nerve fiber as a shearing motion created between the basilar membrane and the tectorial membrane, lying above the hairs. The nerve fibers originating at the hair cells feed into the 8th cranial auditory nerve. This terminates in the cochlear nucleus, the first of a series of synapses in the brain stem that constitute the central auditory system.

The tuning characteristics of the nerve fibers can be determined by a variety of different experimental techniques. The experimental results show that each fiber is tuned to a center frequency which bears a monotonic relationship with the distance along the basilar membrane from the base of the cochlea to the apex. The organization of the nerve fibers such that place has meaning in terms of frequency appears to be preserved at much higher levels of the auditory system, and thus this place information is considered to be important. The final result when a sound has been transmitted from the external environment to an auditory nerve fiber is a spike train of action potentials which captures the relevant time - frequency - amplitude characteristics of the fiber.

2.2. Auditory system elements : anatomy and physiology.

The figure 2.2.1a gives an overall diagram of the auditory system (DALLOS, 1973). For every element of the ear, the functions have been written on the top of the diagram while the supports of the information at every step stay underneath. In this part, we will describe some elements precisely which are involved in our model.

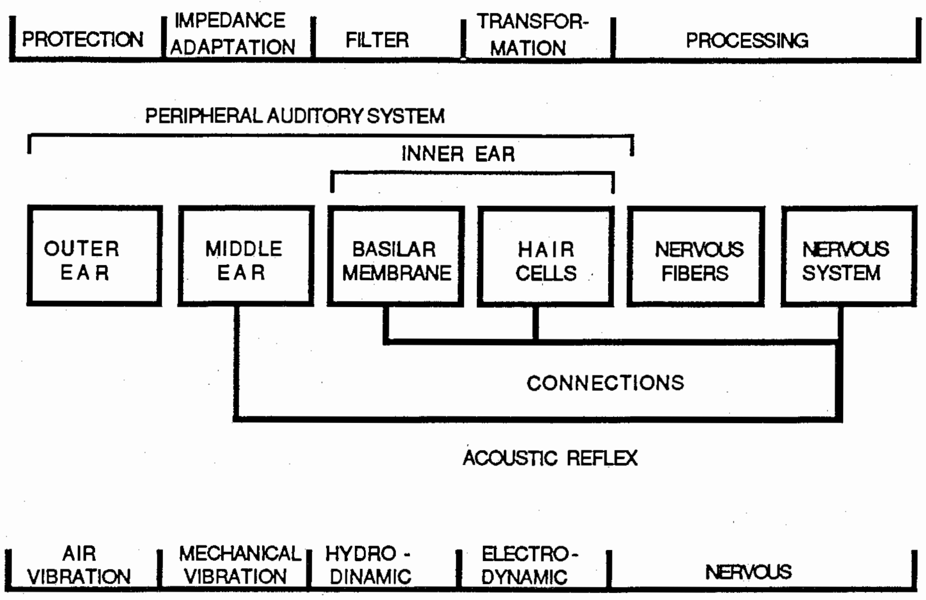


figure 2.2.1a : peripheral auditory diagram by DALLOS, 1973.

2.2.1 outer ear

The role of the outer ear is relatively modest in the hearing mechanism. The outer ear is made of two parts : the pinna and the outer canal. The pinna focuses the signal toward the auditory canal. It amplifies some frequency components and helps to localize the sound origin. On the other hand the auditory canal is essentially a protective organ. The outer ear is characterized by a transfer function shown in figure 2.2.1b. The final part of the outer ear is the eardrum.

2.2.2. middle ear

The middle ear is made up of two parts : a thickset membrane called the eardrum or tympanic membrane and a set of bones : the malleus, the incus and the stapes. At this step of the peripheral auditory system, the acoustic reflex protects the inner ear against high amplitude loudness. (POPELKA et al. , 1976). The middle ear transforms an air vibration into a pressure variation in the inner ear liquids : therefore it is responsible for impedance conversion between air (area) and liquid (area).

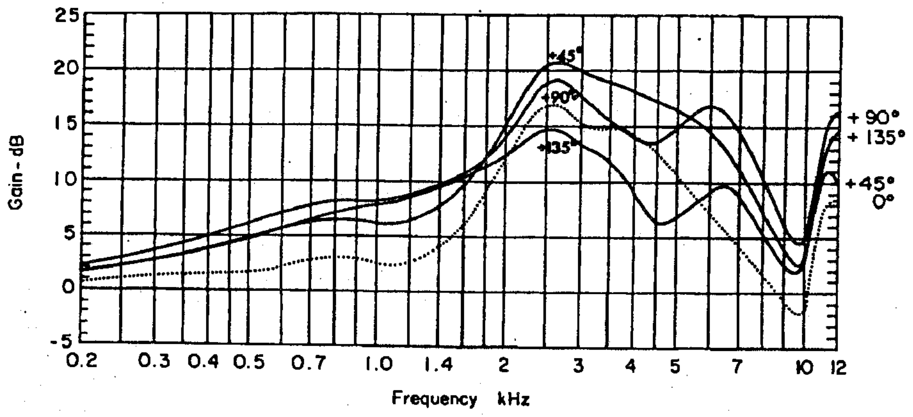


Figure 2.2.1b : The average pressure gain of the external ear in man. The gain in pressure at the eardrum over that in the free field in the horizontal plane ipsilateral to the ear (from James O. Pickles "an introduction to the physiology of hearing", 1982)

2.2.3. inner ear

The inner ear consists of two parts : a balance sensor made up of three semi-circular channels and the cochlea, an acoustic sensor. We will describe the cochlea in details in the following section.

2.2.3.1. the cochlea

Figure 2.2.5., which represents the cochlea shows three compartments separated by different areas which are filled with fluid. The vibrations of the oval window cause a movement of the cochlear fluids and the cochlear partitions (figure 2.2.2a). A travelling wave appears on the basilar membrane and spreads to the apex. This basilar membrane displacement becomes maximal at certain points depending on the input signal frequency, then dies suddenly (deep inside).

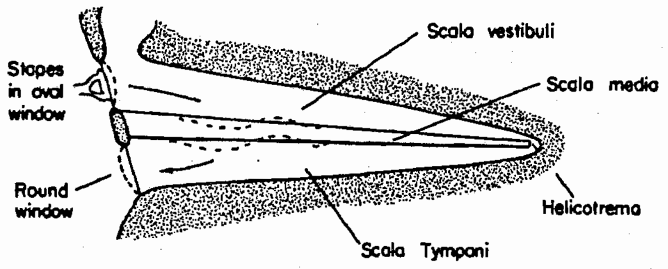


figure 2.2.2a : The path of vibrations in the cochlea (from James O. Pickles "an introduction to the physiology of hearing, 1982)

2.2.3.2. The basilar membrane

G. Von Bekesy, made the first direct microscopic observations of Basilar Membrane motion with cadavers, in 1960. He proved that this organ works as a frequency analyzer for input sounds. In the 1980s, several researchers measured the basilar membrane motions again with fresh cochlea using modern techniques : the Mossbauer method and the laser doppler method. Their results showed that basilar membrane frequency selectivity is as sharp as that measured at the 8th nerve. The sharp response of the Basilar Membrane frequency selectivity is a combination of its mechanical and electrophysiological properties.

The fluid displaced along the Basilar Membrane after a deflection of a partition deflects adjacent portions of the membrane; some experiments have shown if this fluid is removed the frequency selectivity gets worse. The outer Hair cells are also responsible for the sharp filtering selectivity as shown in figure 2.2.2d.

The principal mechanical property modifications along the Basilar Membrane can be summarized as follow:

- For a precise frequency, one point on the basilar membrane is excited with a maximal amplitude.
- This point is as far from the beginning of the membrane as the exciting frequency is low (figure 2.2.2b).

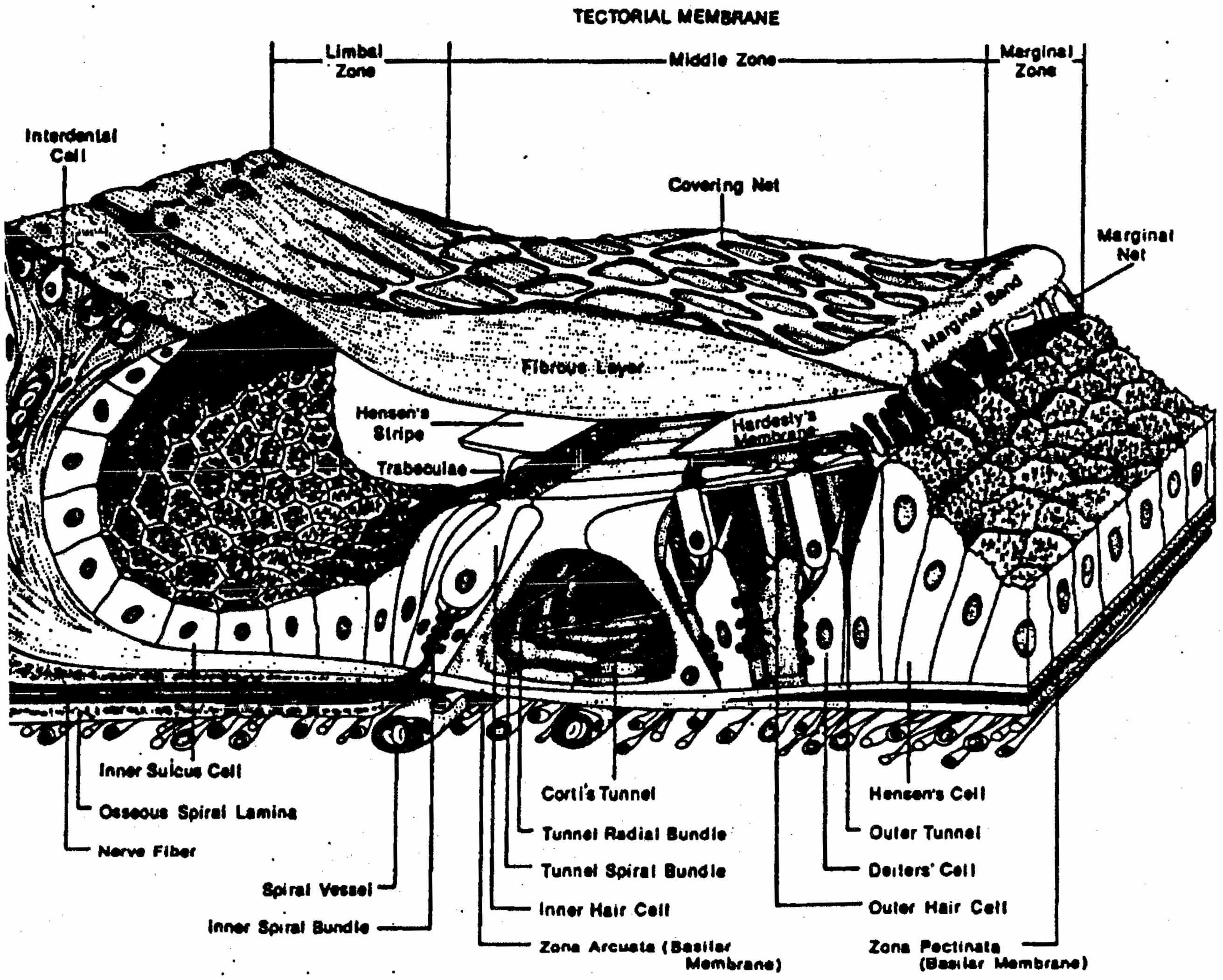


figure 2.2.2 : Schematic drawing of organ de Corti (reprinted from LIM, 1980)

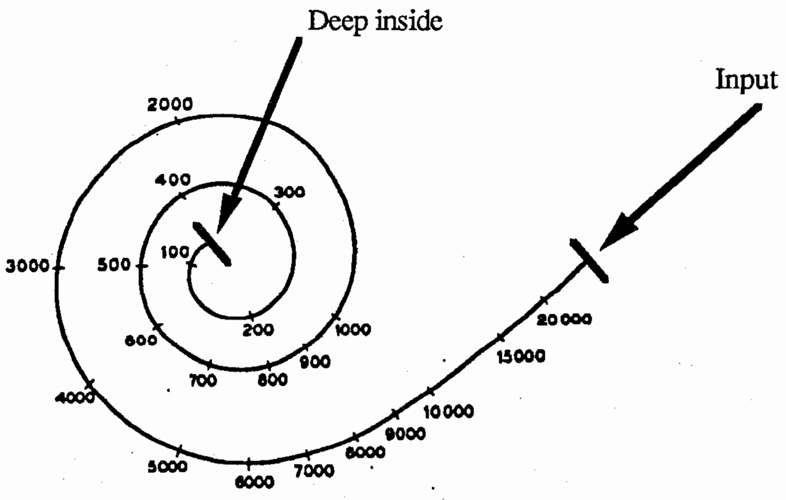


figure 2.2.2b : Frequency selectivity along the basilar membrane (from Handbook of perception vol 4 : Hearing)

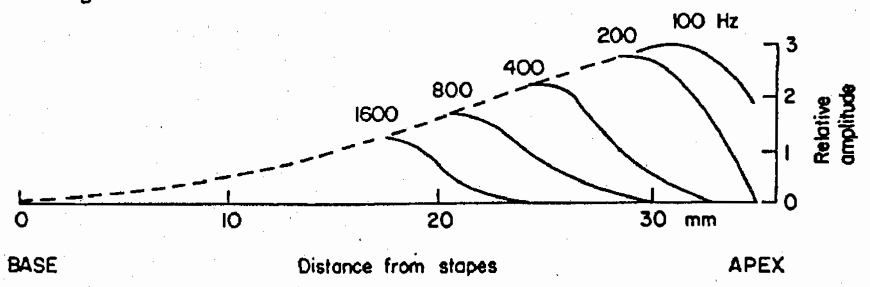


figure 2.2.2c : Displacement envelopes on the cochlear partition for tones of different frequencies (from James O. Pickles "an introduction to the physiology of hearing, 1982)

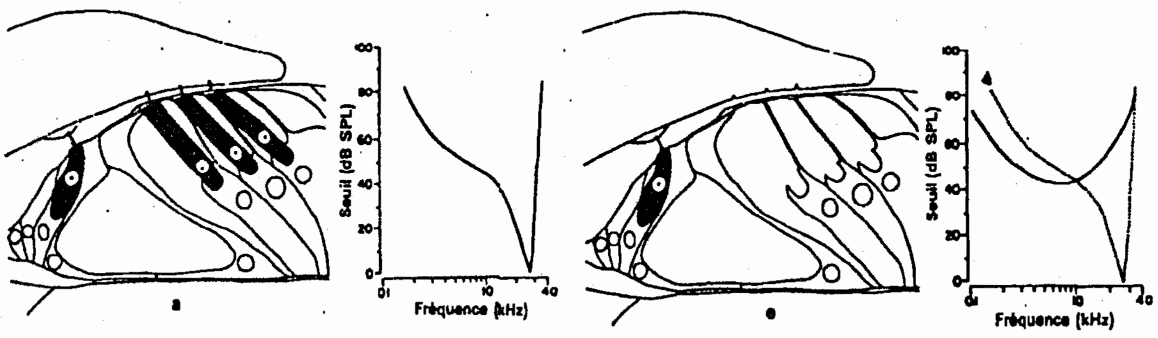


figure 2.2.2d : schematic representation of Organ of Corti and frequency selectivity. On the left : normal state is plotted; on the right : with total destruction of the outer Hair Cells.(from Liberman et Dodds, 1984)

2.2.2.3. The Hair Cells

There are specialized cells: the current flow through them reflects the instantaneous waveform of the sound stimulus (MOORE, 1986). Indeed, each Hair Cell is assumed to modulate a standing current in sympathy with the local motion of the Basilar Membrane (DAVIS, 1965). The figure 3.2.6 shows the arrangement of the hair cells. They are made up of two fundamental parts :

- The receptive area which detects the motion. This area is recognizable thanks to the precise description made by DALLOS (1976).
- The pre-synaptic area which imparts the spike according to the results of the receptive area. At present, we are not concerned with how these signals are generated by the cells. However, the shape of these signals is covered later in this report.

2.2.2.3.1. The inner hair cells (IHC)

There are about 3500 IHC in only one row. They are completely surrounded with other supporting cells along the basilar membrane. Though they are less numerous than the outer hair cells, they are mainly responsible for the signal transmission into the nerve fiber.

3.2.2.3.2. The outer hair cells (OHC)

There are about 20,000 OHC in 3, 4 or 5 rows. We now know that the OHC affect IHC excitation. However, the mechanism is not understood (H. D. CRANE, 1983, "The IHCs - TM connect, disconnect and efferent control").

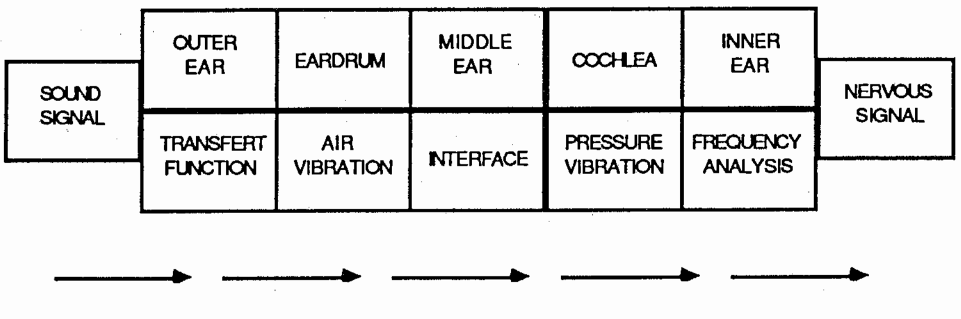


figure 2.2.4 : simplified diagram of the signal path into the ear

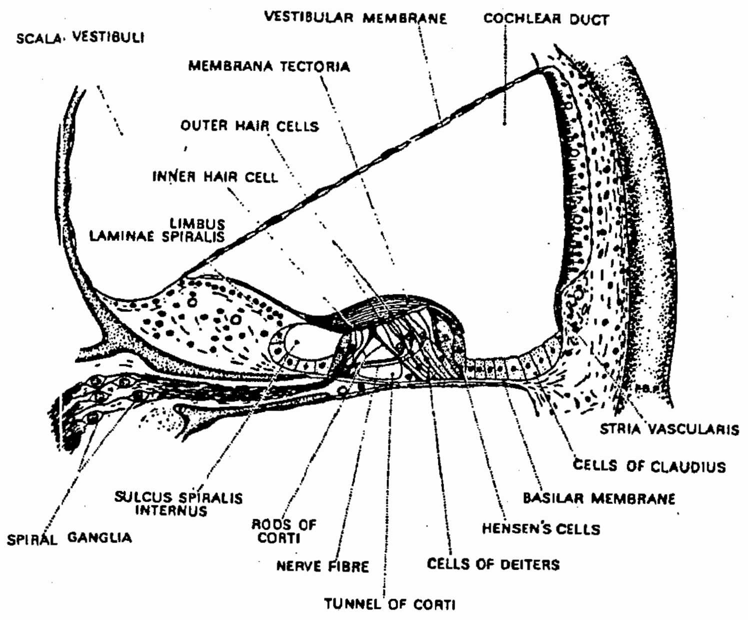


figure 2.2.5.: Cross section of the cochlea, showing the organ of Corti. The actual receptors are the hair cells lying on either side of the tunnel of Corti. From B. C. J. Moore "An introduction to the psychology of hearing"

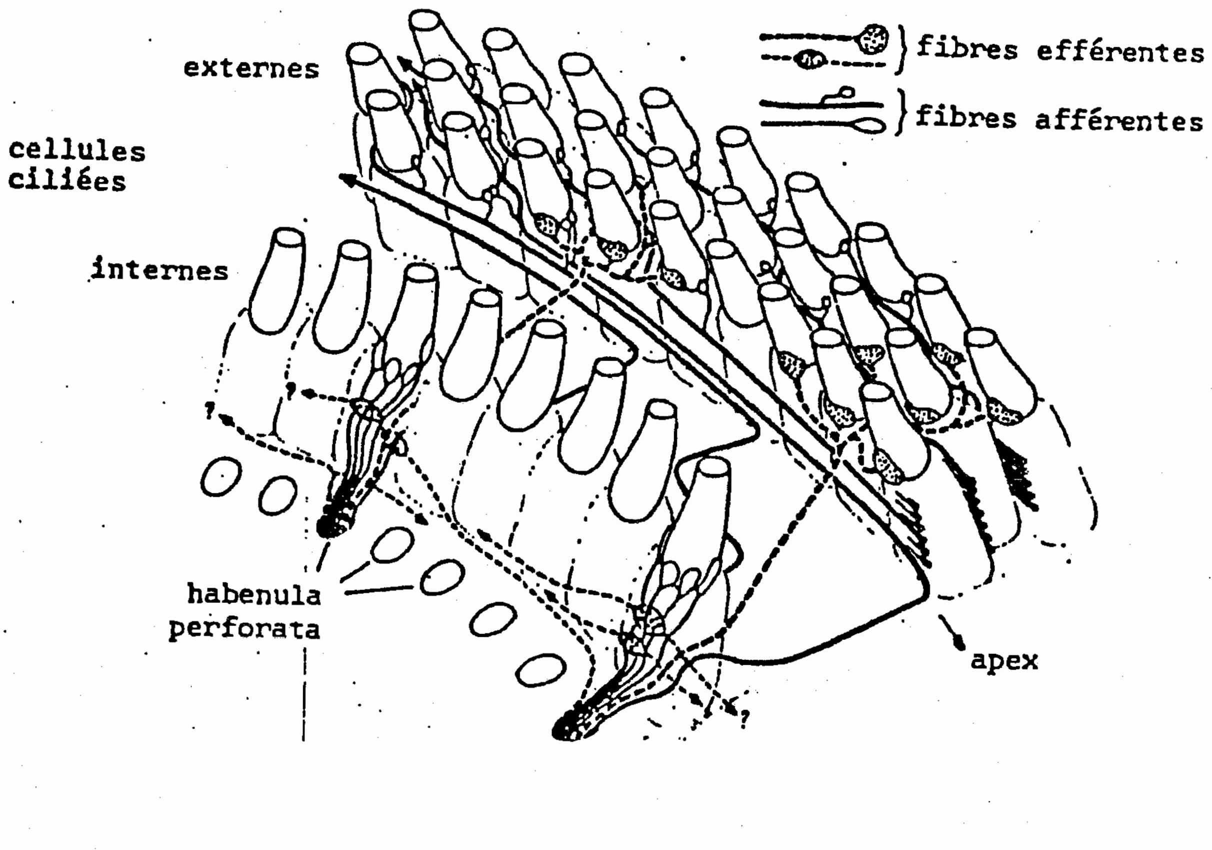


figure 2.2.6 : simplify diagram of outer and inner hair cells arrangement

3. Seneff's Inner Hair Cell (IHC) Model

3.1. Introduction

Following the Basilar Membrane model, each channel is processed independently through a non-linear stage to model the transformation from basilar membrane vibration to auditory nerve fiber response. In the hair cell, the motion of the Basilar membrane is transformed into a receptor potential. We are going to describe this second model of the human cochlea. This model follows HIRAHARA's Basilar Membrane model but will still improve the response of the total model with real speech data.

3.2. Overall description

This model is based on the properties of the human auditory system. Seneff used critical-band filters at the first stage followed by an inner hair-cell-model and finally an envelope detector [figure 3.2.1(a)]. In our case we use Hirahara's model instead of Seneff's critical-band filters [figure 3.2.1(b)].

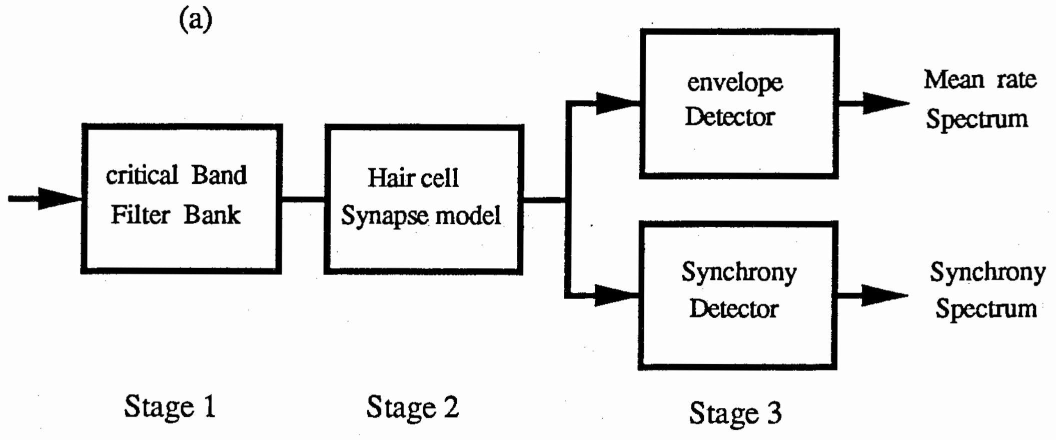
Our analysis system consists of a set of 61 channels which collectively cover the frequency range from 110 to 8800 Hz. The bandwidth of each channel is approximately 1/3 Bark. Every filter output is introduced into the Inner-Hair-Cell model.

3.3. Role of each stage in the model

In this section, the inner hair cell functional model proposed by Seneff will be discussed. Her model consists of five stages [figure 3.3.1] : a halfwave rectifier (HWR), a short term adaptation component proposed by Goldhor (GLD), a lowpass filter (LPF), a rapid Automatic Gain Control (AGC) and a Generalized Synchronity Detector (GSD). The output of the AGC provides the mean rate spectrogram and GSD output gives the synchronicity spectrogram. We investigated her model and pointed out its advantages and disadvantages. Then we proposed some improvements of the model to use it with AQ cochlea filter and to use it for noisy speech data.

All the stages of the model except the lowpass filter are nonlinear systems. Thus, the input levels for each stage is important.

(a)



(b)

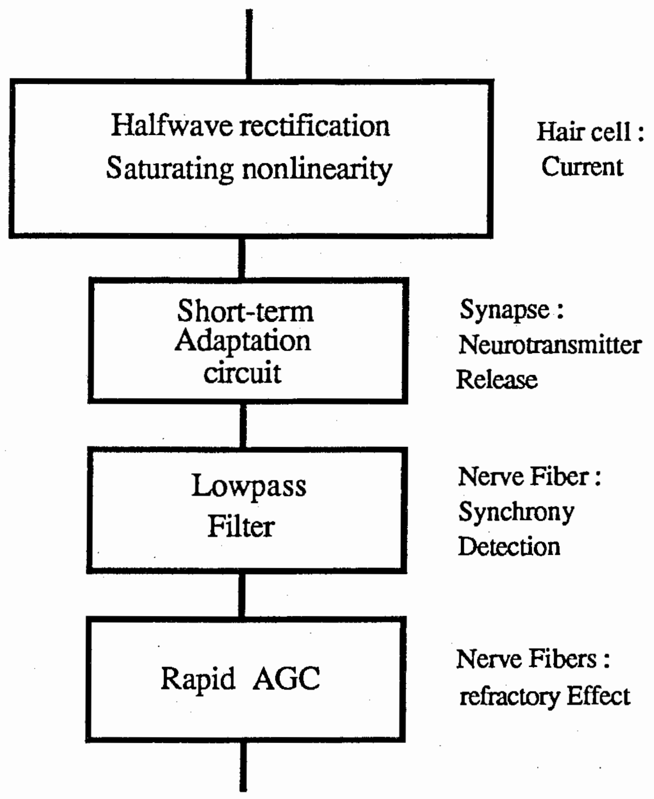

Halfwave rectification Saturating nonlinearity
Hair cell : Current
Short-term Adaptation circuit
Synapse : Neurotransmitter Release
Lowpass Filter
Nerve Fiber : Synchrony Detection
Rapid AGC
Nerve Fibers : refractory Effect

figure 3.2.1 : (a) The Seneff's computer model. (b) The subcomponents of Stage 2 suggested auditory system affiliations indicated at right.

3.3.1. Halfwave Rectifier. (HWR)

The HWR simulates the unidirectional discharge characteristics of the IHC; that is, the IHC discharges only when the hair cells are moved in certain directions by the basilar membrane motion displacement. The HWR is based on the measured hair-cell current response as a function of fixed displacement of the cilia as determined in the frog sacculus by Hudspeth & Corey, 1977.

3.3.1.1. principle and previous choice

The instantaneous HWR model is defined mathematically as follows :

$$y = 1 + A \bullet \mathrm{atan}(B \bullet x) \qquad \text{when } x>0 \qquad (1\text{-}1)$$

$$y = \exp(A \bullet B \bullet x) \qquad \text{when } x \leq 0 \qquad (1\text{-}2)$$

where : A : output level
B : input gain

The parameter B defines the slope at the origin of the tangent function therefore it can be viewed as an input gain. This function is exponential for negative signals and those between 0.0 and 1.0. It is linear but shifted for small positive signals and compressive for larger signals, saturating at $1 + A.\Pi/2$. Therefore, the Halfwave Rectifier looks like a normalizer: it reduces the negative input component, shifts the middle component according to the value B, and has a maximum amplitude output according to the value of A. The output signal is always positive.

According to the outputs of her Critical Band Filter model, Seneff has chosen :

$A = 10$
$B = 65$

3.3.1.2. experiments and conclusion

The HWR is the first step of the input signal processing into the IHC model. Therefore it plays an important role in signal processing. The response sensitivity is based on the value of the parameter value B, which determines the slope. Used with the AQ cochlear filter, the HWR is too sensitive. Therefore, we fixed the values as follows :

$A = 10$
$B = 4.0$

3.3.2. Goldhor's model

This is a non-linear model for short term adaptation proposed by Goldhor in 1985. Our IHC model is based on this principal element because it simulates the characteristics of the IHC response which depends on the time parameter.

3.3.2.1. Principle

This model consists of a simple non-linear circuit as shown in figure 3.3.2. The input - which is the output of the HWR described above - is the voltage source generator whereas the output is the current through the diode.

This model has two separate mechanisms that influence the concentration of a substance which could be thought of as a neurotransmitter or an ion characterized by μa and μb. This first constant reflects the flow of the hair cell supply source region which is proportional to the concentration gradient across the membrane. However, channels in this membrane are closed whenever the concentration in the supply region is too small (i.e. when the concentration gradient is negative). The substance is also lost throught natural decay at a rate that is proportional to its concentration within the region, with a proportionality constant μb. Mathematically, the process can be expressed as follows :

$$\frac{dVc(t)}{dt} = \mu a \bullet [S(t) - Vc(t)] - \mu b \bullet Vc(t) \qquad \text{when } S(t) \geq Vc(t)$$

and

$$\frac{dVc(t)}{dt} = - \mu b \bullet Vc(t) \qquad \text{when } S(t) < Vc(t)$$

where Vc(t) reflects the concentration of the substance within the region and S(t) is the concentration in the source region. Thus, the output of the system is the flow rate across the membrane:

$$\mu a \bullet [S(t) - Vc(t)]$$

which controls the probability of firing the nerve fiber.

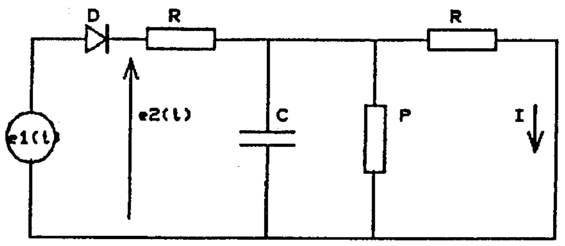


figure 3.3.2 : diagram of Goldhor's model

Let S(n) be the voltage at the generator and Vc(n) be the voltage at the capacitor.

first case : $S(n) \geq Vc(n)$

$$i(t) = i1(t) + i2(t) \quad \text{with} \quad \begin{cases} i1(t) = \dfrac{Vc(t)}{Re} \\ i2(t) = C \bullet \dfrac{dVc(t)}{dt} \end{cases}$$

therefore

$$i(n) = \frac{Vc(n)}{Re} + C \bullet \frac{dVc(n)}{dt}$$

remind of the Euler's approximation derivation formula :

$$\frac{dV(n)}{dt} = \frac{V(n) - V(n-1)}{\Delta t}$$ where Δt is the sample time constant

thus

$$i(n) = \frac{Vc(n)}{Re} + \frac{C}{\Delta t} \bullet [Vc(n) - Vc(n-1)] \qquad (1\text{-}3)$$

$$\Delta t \bullet Re \bullet i(n) = \Delta t \bullet Vc(n) + C \bullet Re \bullet [Vc(n) - Vc(n-1)]$$

from the mathematical expression written above, it comes on :

$$\frac{Vc(n) - Vc(n-1)}{\Delta t} = \mu a \bullet [S(n) - Vc(n)] - \mu b \bullet Vc(n)$$

$$Vc(n) \bullet [\frac{1}{\Delta t} + \mu a + \mu b] - \frac{1}{\Delta t} \bullet Vc(n-1) = \mu a \bullet S(n)$$

therefore

$$Vc(n) = \frac{1}{\frac{1}{\Delta t} + \mu a + \mu b} \bullet [\mu a \bullet S(n) + \frac{1}{\Delta t} \bullet Vc(n-1)] \qquad \text{when } S(n) \geq Vc(n)$$

initial condition : $Vc(0) = 0.0$ (1-4)

second case : $S(n) < Vc(n)$

$$\frac{dVc(t)}{dt} = -\mu b \bullet Vc(t)$$

thanks to Euler, we can write again :

$$\frac{Vc(n) - Vc(n-1)}{\Delta t} = -\mu b \bullet Vc(t)$$

therefore

$$(\frac{1}{\Delta t} + \mu b) \bullet Vc(n) = \frac{1}{\Delta t} \bullet Vc(n-1)$$

$$Vc(n) = \frac{1}{1 + \mu b \bullet \Delta t} \bullet Vc(n-1) \qquad \text{when} \quad S(n) < Vc(n) \qquad (1\text{-}5)$$

in this case, the diod is locked on therefore :

$i(n) = 0.0$ (1-6)

let's look at the parameters values :

$$\frac{1}{\mu b} = \tau_2 = Re.C$$

C is fixed, therefore $Re = \frac{\tau_2}{C}$

$$\frac{1}{\mu a + \mu b} = \frac{R \bullet Re}{R + Re} \bullet C = \tau_1$$

$$R = \frac{Re}{\frac{Re.C}{\tau_1} - 1}$$

From these equations, a discrete realization of Goldhor's model is implemented in our program. Goldhor showed that his model follows certain properties of short term adaptation which have been observed for auditory data (Smith & Zwislocki, 1975). Thus, when the signal S(t) is applied, the flow rate is initially very high and then decreases exponentially with the time constant τ_1 to a steady state value. After the signal is turned off, the concentration gradient becomes negative and the flow rate remains zero until the capacitor voltage Vc(t) decays exponentially with the time constant τ_2 to a spontaneous concentration level.

3.3.2.2. Previous choice

The values for the time constants τ_1 and τ_2 are determinated so that the model feature resembles the auditory nerve response (Harris & Dallos, 1979). For this reason Seneff in her model has fixed the values as follows :

$$\begin{cases} \tau_1 = 15 \text{ ms} \\ \tau_2 = 120 \text{ ms} \end{cases}$$

therefore

$$\begin{cases} \mu a = 66.54 \\ \mu b = 8.33 \end{cases}$$

we obtain the following parameter values

C = 1 mF (fixed)
Re = 120 Ω
R = 17.1 Ω

3.3.2.3. study.

The input of our computational Goldhor model is the output of the halfwave rectifier described above. With the parameters values finally chosen. In this experimental study, it is seen that the τ_1 and τ_2 parameters values are very important to focus the sensitivity of the Goldhor's model response.

Role of the Time Constants τ_1 and τ_2

In the simplest terms, we can understand the Goldhor's model as a diode where the threshold varies as a function of three parameters: the two time constant values and the amplitude of the input signal. (figure 3.3.2)

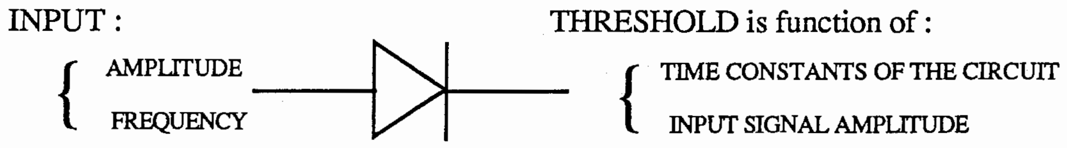


figure 3.3.2 : simplest diagram of the principle of Goldhor's model

Therefore, the threshold of the diode is the voltage at the capacitor and, in our case, the output of our model is the current through the diode. Thus, there is a current i(t) when the diode is locked on, i.e., when the input amplitude signal is greater than the instantaneous threshold.

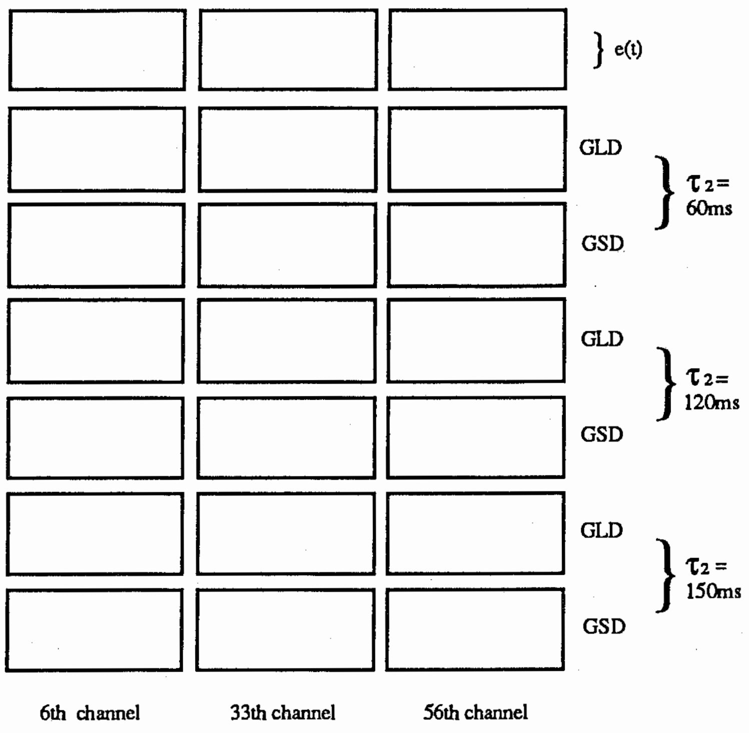
3.3.2.4 : experiments and conclusion

We tried several values of τ_2 experimentaly for different channels in order to observ the effects on the outputs of the GLD and the GSD (we will talk about this component later). The figure 3.3.3 shows some important results. We will always preface the experiment results with a simplify figure showing how we plotted these results. As shown in this figure, we plotted the results for three channels (6th/f=203Hz; 33rd/f=1456Hz and 56th/f=5111Hz) and the corresponding inputs are plotted at the top of every column. We can see the τ_2=60ms (lines 2&3), τ_2=120ms (l. 4&5) and τ_2=150ms (l. 6&7).

We observed the signal for 1s, that is to say one division on these graphs is equal to 0.1s. Looking at the GLD output between t=0.1s and t=0.3s, we can see for every channel that when τ_2=60ms, the GLD is more sensitive as shown between t=0.1s and t=0.3s. The GSD is also affected by the variations of this parameter. Therefore, the time constant discharge parameter τ_2 is very important because it simulates the response of the actual auditory system which in reality contains time adaptation according to the signal amplitude input. In our final version for this stage we will keep these values. Conclusion:

$$\tau_1 = 15\text{ ms} \qquad \tau_2 = 120\text{ ms}$$

The output of these adaptation stages is the input of the next step which is the Lowpass Filter.



Organisation of the figure 3.3.3 plotted on the next page

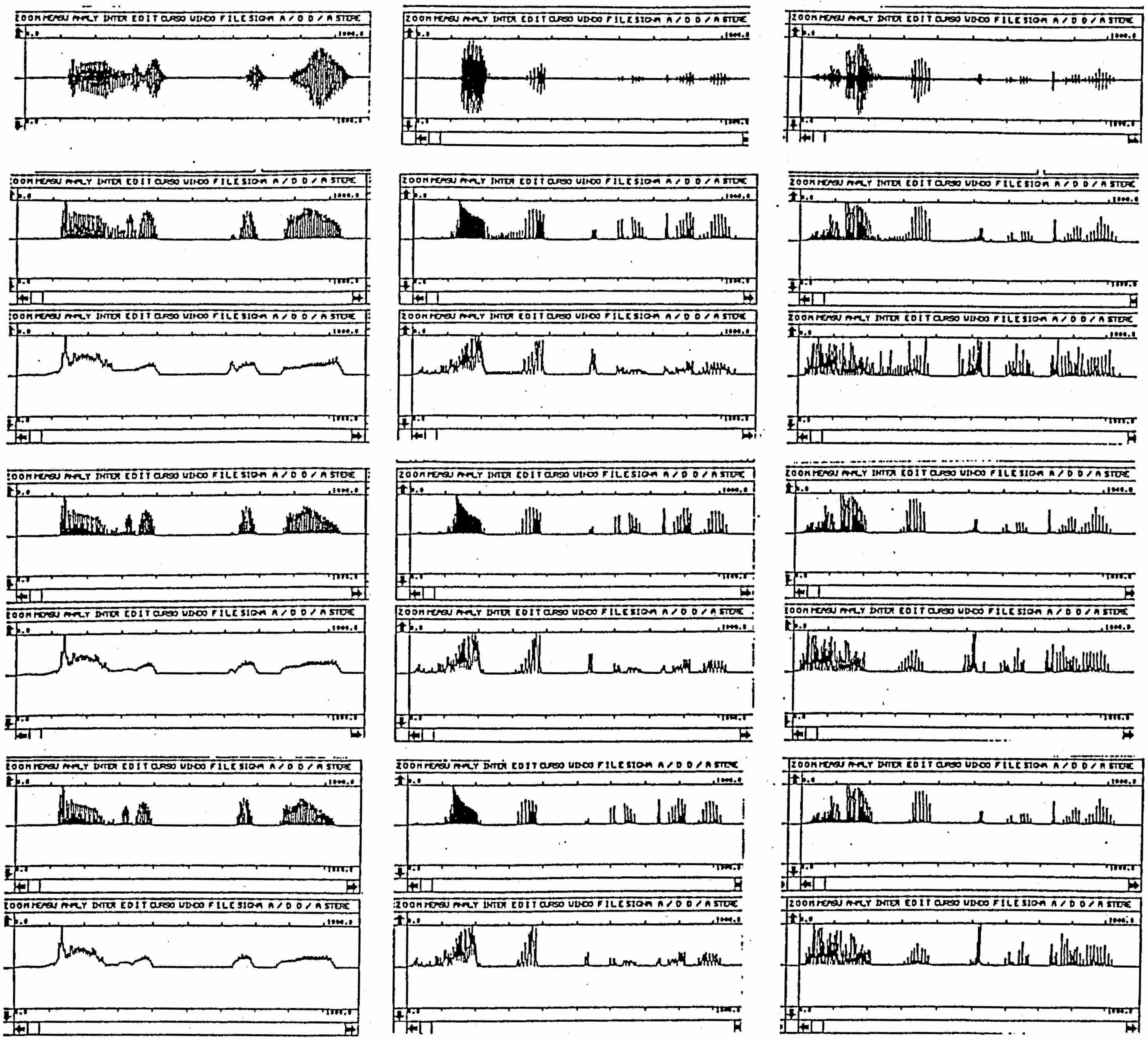


figure 3.3.3 : influence of the discharge time constant in Goldhor's model

3.3.3. The Low Pass Filter (LPF)

The lowpass filter is a first order LPF. The shape of the signal included in the envelope of Goldhor's model output is very sharp because the diode locks in instantaneously. Therefore, the current is also instantaneous. Thus, Seneff included an LPF in order to reduce synchrony to high frequency stimuli, and to smooth the wave shape.

3.3.3.1. Principle

The LPF used by Seneff in her model, is mathematically defined as follows:

$$H(z) = \left(\frac{1 - \alpha}{1 - \alpha \bullet z^{-1}}\right)^{nLP}$$

The above is a cascade of nLP integrators. We used a first order lowpass filter in our model defined as follows:

$$y(s) = \frac{1}{1 + \tau p} \cdot x(s)$$

The equation in the discrete domain for the resulting transfer function is:

$$y(n) = \frac{x(n) + \frac{\tau \,.\, y(n-1)}{\Delta t}}{1 + \frac{\tau}{\Delta t}} \qquad (1\text{-}7)$$

3.3.3.2. Experiments and conclusion

The LPF affects only the contents of the envelop as provided but does not change the form of the envelope which gives the spectrogram representation. However, we like Seneff,. will use the output of this filter as the input of the AGC for the moment.

3.3.4. Automatic Gain Control (AGC)

The next component of the model employed by Seneff is the AGC.

3.3.4.1 Principle and previous choice

AGC is defined mathematically as follows :

$$y(n) = \frac{x(n)}{1 + K_{AGC}.<x(n)>} \qquad (1-8)$$

where x(n) is the current value of the input and <x(n)> symbolises the "expected value of x(n)" obtained by processing x(n) through a first order LPF with time constant τ_{AGC} K_{AGC}. These two last parameters are responsible for the more or less sharp form of the envelope at the ouput of the AGC.

3.3.4.2. Experiments and conclusion

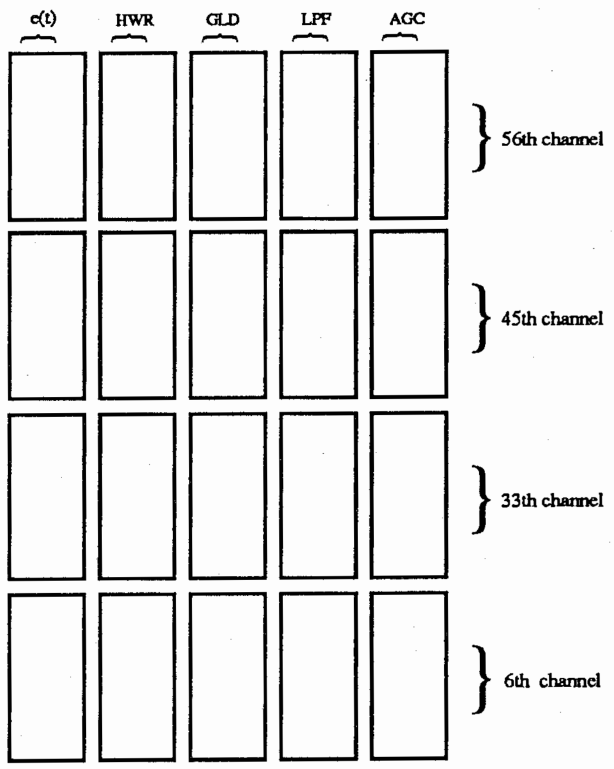
Figure 3.3.4 shows every step of the model up to the AGC using real speech data without noise. We can compare every step of the signal processed into the model used by Seneff. In the figure we used the same constant as Seneff for the AGC parameters that is :

$\tau_{AGC} = 3$ ms & $K_{AGC} = 0.002$

We noted that in our case these values do not affect the shape of the signal because Kagc is too small compared to 1.0. To get the shape obtained by Seneff in her report we used:

$\tau_{AGC} = 0.3$ ms $\quad K_{AGC} = 1000$

About 15minutes clculating time for the entire 61 channels is. The results are similar to these described by Seneff; however, at this step we can already ask ourselves, about the necessity of some components in our case.



Organisation of the figure 3.3.4 drawed on the next page

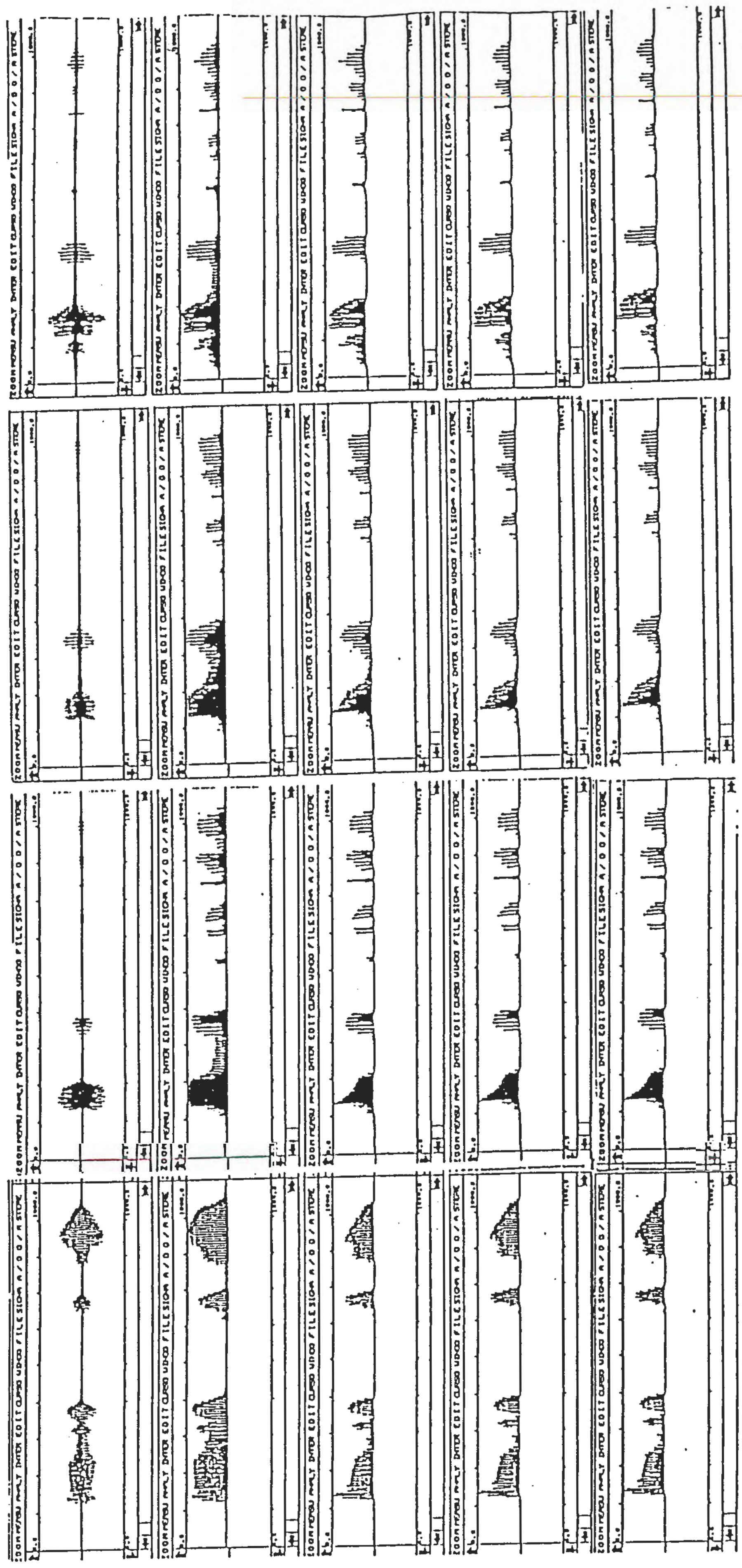


figure 3.3.4 : every step of the model : (a) the input which is real speech data after passing through the 34th channel of the Q adaptive lowpass filter, (b) after halfwave rectification, (c) after short term adaptation, that is Goldhor's model, (d) after lowpass filtering and (e) after the AGC.

3.3.5. Generalized Synchrony Detector (GSD)

The last step of this model is made up of a circuit which detects the periodicity of the signal. Therefore, we will obtain a precise representation similar to a spectrogram while always remaining in the time field calculation. The principle of this component is ingenious because it is very simple

3.3.5.1. Principle.

The diagram of the GSD is shown in figure 3.3.5. The input of the GSD is, for the moment, the output of the AGC. The principle of the GSD is as follows :

When the frequency input signal is equal to the inverse of the GSD delay, the difference becomes equal to 0.0 and the sum equal to some value. Then the sum is divided by the difference, which is equal to 0.0 in the case of periodicity, thus the output signal becomes very large. In fact, we add an integration function at the output of the difference and the sum; otherwise, the GSD would be too sensitive. Figure 3.3.15 shows the role played by the integration window length. We will discuss that in the next section. The last step of the GSD calculation is a saturating halfwave component which is important for the high channel response : the figure 3.3.16 shows that. the function employed for this stage is similar to the function employed for the first step of this model (see paragraph 3.3.1).

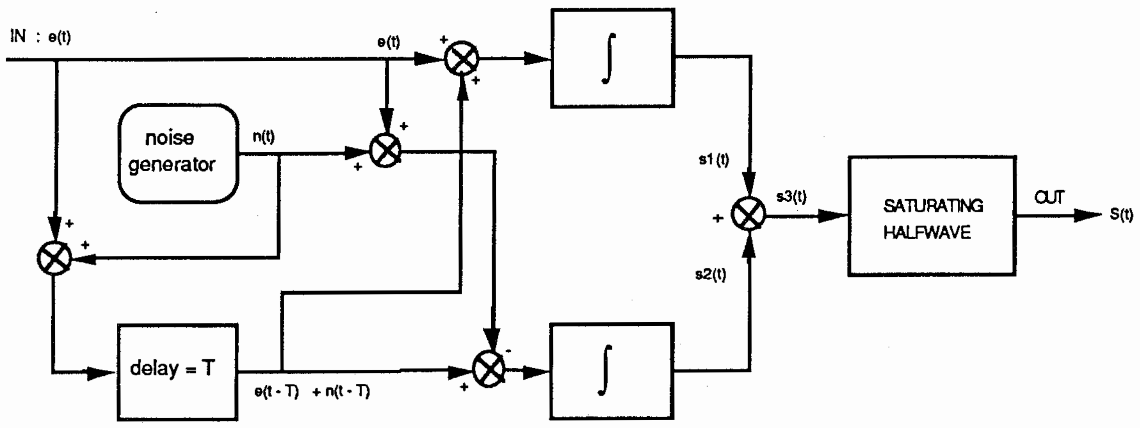


figure 3.3.5 : schematic diagram of the Generalized Synchrony Detector

(A) The Delay T : Role and Calculation

The delay varies according to the channel; thus there are as many hair cell models as there are filter channel outputs because each hair cell model is connected to the corresponding filter output. The delay varies between :

$f_{input} \cong 101.5$ Hz   ie $T \cong 10$ ms   and

$f_{input} \cong 8.6$ kHz   ie $T \cong 122\ \mu s$

Figure 3.3.6 shows each filter cutoff frequency value and the corresponding period according to the channel number, and finally the corresponding number of items necessary for the delay. Figure 3.3.7 shows every step necessary for calculating every delay T and figure 3.3.8 represents the 61 channel function of the corresponding filter frequency values.

The delay T necessary for comparing the signal in the GSD processing is calculated with the sampling period. This delay precision depends on the sampling frequency, that is to say the precision will be very high for the low frequencies (because the period is large there is an important sample number). On the other hand the precision decreases as the channel output frequency increases. The graph in figure 3.3.10 shows the number of samples according to the filter cutoff frequencies; figure 3.3.9 shows the % error made according to the filter cutoff frequencies. These errors are calculated as follow:

$$\varepsilon = \left| \frac{1}{f_{input}} - (\text{integer}) \frac{f_{sampling}}{f_{input}} \cdot \Delta t \right| \qquad (1\text{-}10)$$

The second part of this equation is the formula used in our program for calculating the GSD computing delay. As we can see in figure 3.3.10 the precision decreases from 198 samples to 15 samples in the first half (that is to say about a 180 sample variation) and stays between 14 samples and 2 samples for the second half (only a 12 sample variation). Therefore, as we can see in figure 3.3.9, in this second domain the probability of a possible error is maximal compared to the actual

	fInput (Hz)
1	
2	
3	101.350
4	135.311
5	169.438
6	203.774
7	238.367
8	273.265
9	308.519
10	344.178
11	380.298
12	416.933
13	454.143
14	491.988
15	530.538
16	569.854
17	610.017
18	651.102
19	693.196
20	736.387
21	780.774
22	826.462
23	873.865
24	922.207
25	972.522
26	1024.657
27	1078.773
28	1135.045
29	1193.667
30	1254.848
31	1318.823
32	1385.847
33	1456.203
34	1530.201
35	1608.184
36	1690.531
37	1777.658
38	1870.020
39	1968.118
40	2072.495
41	2183.741
42	2302.488
43	2429.408
44	2565.204
45	2710.599
46	2866.316
47	3033.065
48	3211.506
49	3402.232
50	3605.742
51	3822.423
52	4052.560
53	4296.328
54	4553.893
55	4825.430
56	5111.241
57	5411.859
58	5728.164
59	6061.511
60	6413.868
61	6787.961
62	7187.499
63	7617.413
64	8084.248
65	8596.708

figure 3.3.6 : 1- channel number. 2- corresponding channel cutoff frequency.

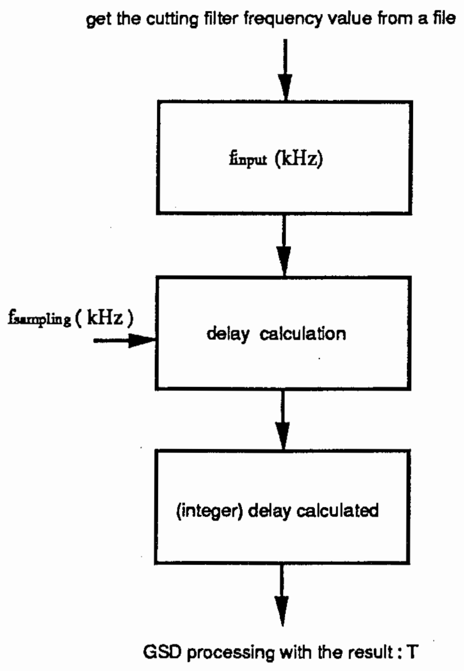

get the cutting filter frequency value from a file
finput (kHz)
fsampling (kHz)
delay calculation
(integer) delay calculated
GSD processing with the result : T

figure 3.3.7 : GSD delay calculation procedure

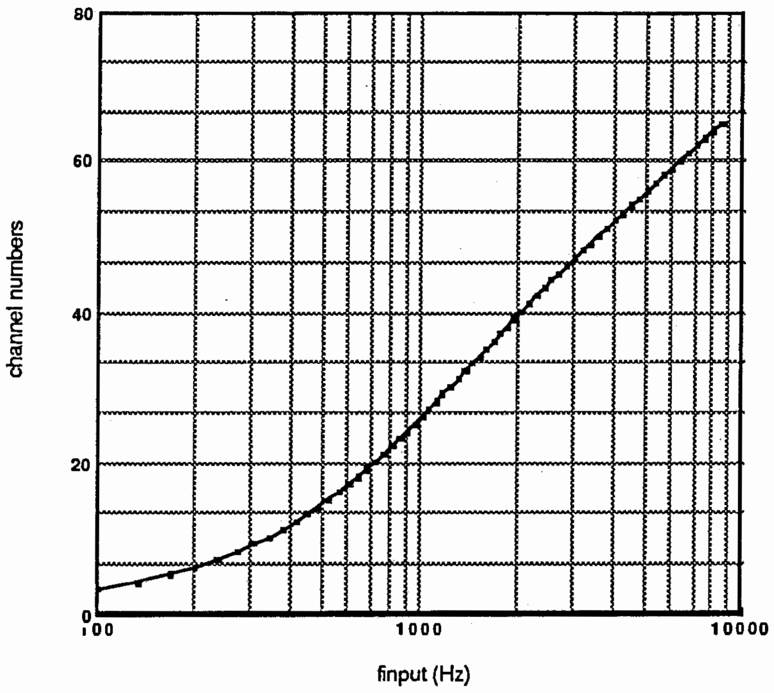


figure 3.3.8 : channel numbers and corresponding cut off filter frequency values

frequency and improves from 3% (maximal value above the 27th channel) to 50% (from the 60th channel). Note that in this last domain, we are just at Shannon's limit because there are only 2 items per period. (remember Shannon has shown that when a signal is sampled, at least 2 samples per period are necessary to recognize the previous signal). In our application, we use a 20kHz sampling frequency, that is to say the sample period is constant and equal to:

$\Delta t = 50\ \mu s$ corresponding to $f_{slampling} = 20$ kHz

The next step in the GSD signal processing is the integration. As we are going to see in these two sections, considering the preceding remarks the integration length and the saturating halfwave rectifier are very important for the final result.

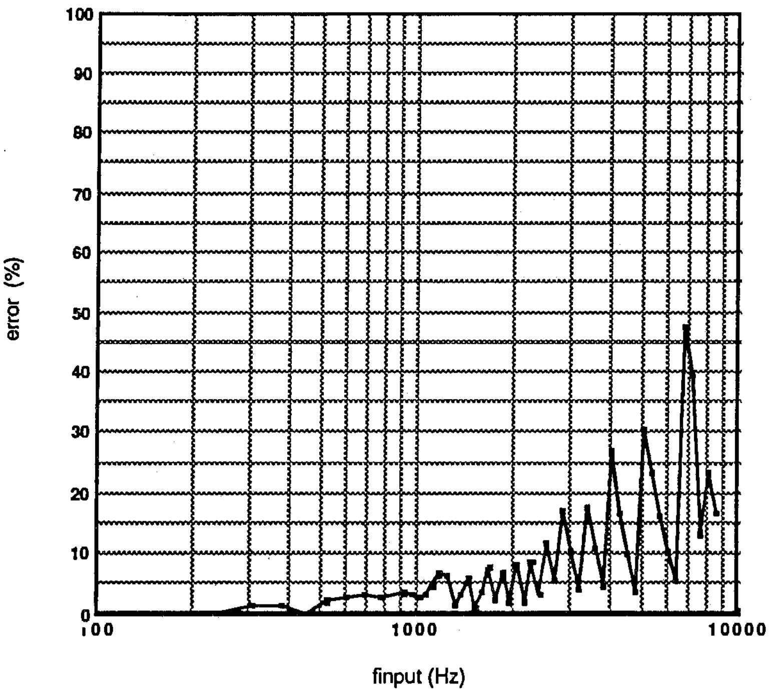


figure 3.3.9 : error made during the computing delay calculation. We have drawn the error value as a function of frequency.

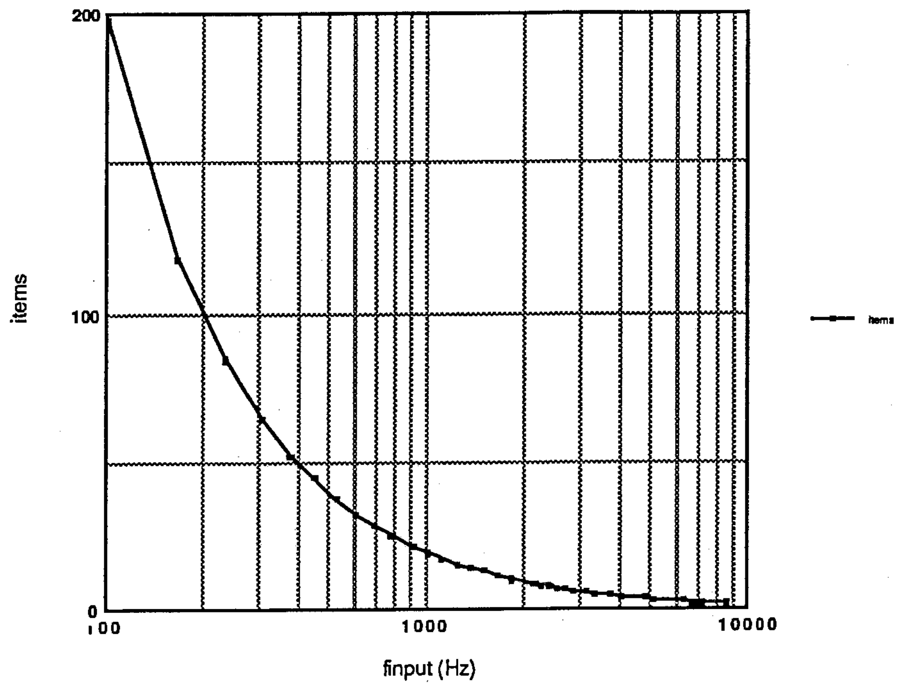


figure 3.3.10 : number of samples as a function of the filters frequency

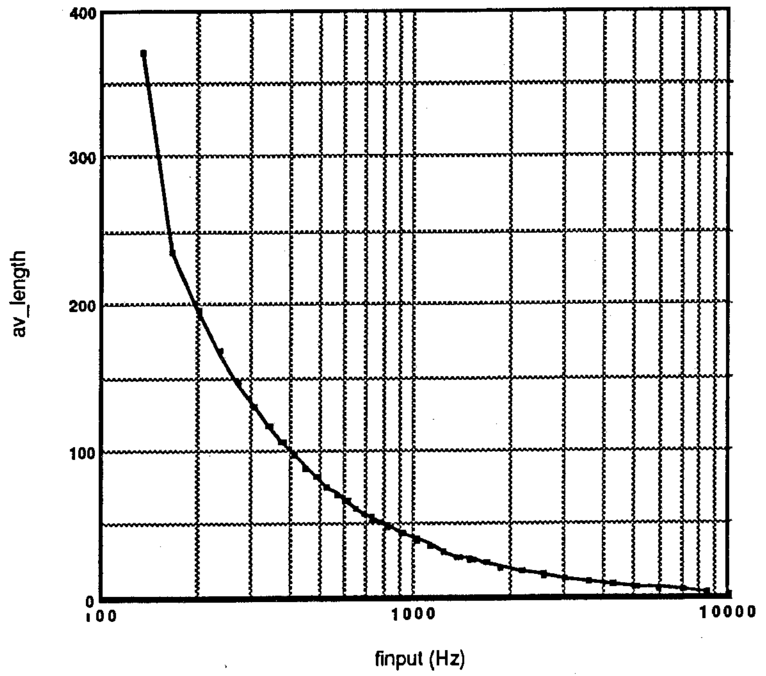


figure 3.3.11 : relation between the average length and the frequency

(B) Role of the Integration Window Length

Without integration processing at the output of + and - comparisons (as in the previous GSD model), the output of the quotient +/- becomes very large instantaneously when the 2 comparing values are equal. However, these 2 values can be equal by accident without the signal being periodic : in this case we would observe some anomylous impulses at the output (figure 3.3.12). The integrators look like a kind of signal average, that is to say every integrator records a number of sampling values according to the window length integration ("average length") and therefore smooths the quotients +/- results. As noted above, the errors are eliminated by this calculation because a chance punctual equal value given will be "digested" in the integration window length. Consequently, the quotient +/- is not the quotient of instantaneous comparison values but the quotient of average comparison values. In figures 3.3.12 and 3.3.13 above, the notations are the same as those used in figure 6.4.10. One division corresponds to the delay T. The signal deleted + the previous signal are represented on the same diagram in order to understand the principle of the comparison. Thus, at every intersection between e(t) and e(t-T), the difference becomes equal to 0. Therefore, the quotient +/- becomes very large..

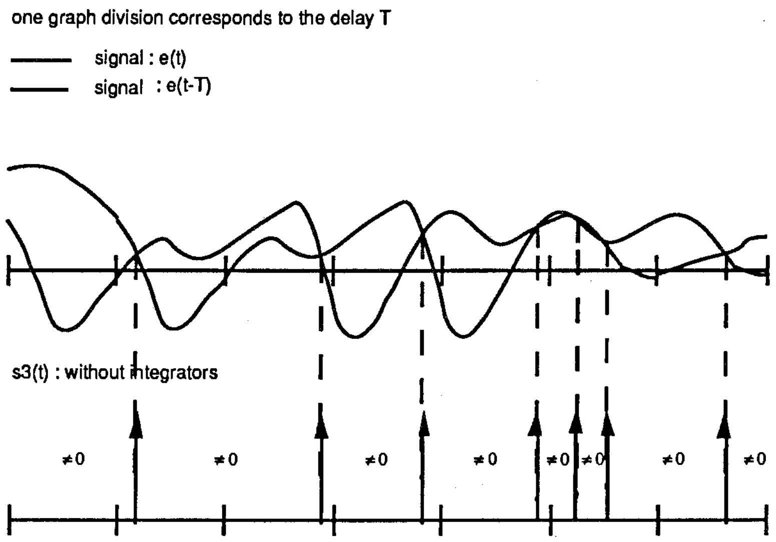

one graph division corresponds to the delay T
signal : e(t)
signal : e(t-T)
s3(t) : without integrators
≠0
≠0
≠0
≠0
≠0
≠0
≠0
≠0

figure 3.3.12 : instantaneous large errors without integration processing

The figure 3.3.13 compares the output with integration and without integration processing for some signal having a periodicity over 2T. We have again drawn e(t) and e(t-T) on the same diagram and the temporary periodicity is clearly as a bold line. Without integration processing, the output s(t) becomes very large every time there is an intersection even if the signal is not periodic. However,with integration processing, s(t) becomes very large only when the real signal periodicity and accidental intersections are completely smoothed. The response time of s(t) - therefore the sensitivity - depends on the average length of the integral which determines the backward observing window time; this average length is defined as follows :

$$\text{av\_length} = \text{(integer)}\ (2 \,.\, \frac{f_{\text{sampling}}}{f_{\text{input}}})$$

That is to say, the average length depends on the filter cutoff frequency which is the input frequency for our hair cell model. The average length is an integer, and the result like the delay T, will be a sampling number. Figure 3.3.18 shows the relation between the average length and the input filter cutoff frequency

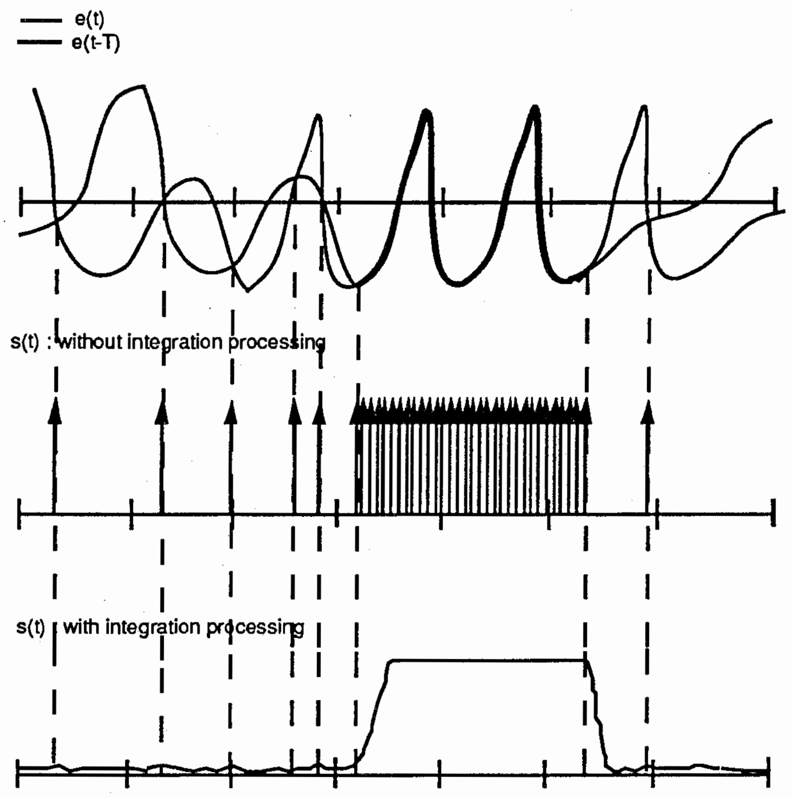

e(t)
e(t-T)
s(t) : without integration processing
s(t) with integration processing

figure 3.3.13 : s(t) with integration processing and saturating halfwave rectifier

This curve representing the average length as a function of frequency has the same shape as the curve represented in figure 3.3.10 which also shows the number of samples as a function of the frequency. Indeed :

$$\text{av\_length} = 2 \,.\, \text{items\_number} \quad \text{(at every frequency)}$$

that is to say we are averaging two periods so as to compensate for improv sensitivity at high frequencies when there are too few samples for precise comparison. The figure 3.3.14 shows how the average length value affects the result of the GSD. For this experiment we have changed the value as follows :

$$\text{ave\_length} = \frac{1}{x} \,.\, \frac{f_{\text{sampling}}}{f_{\text{input}}}$$

where x varies between 4 (at the top of the figure) and 0.5 (at the bottom of the figure), that is to say av_length(x=4) = av_length(x=0.5)/8. Two channels are plotted side by side and the corresponding input real speech data are also represented at the top. The sentence used is always "*sa n ga tsu ko ko no*".

first : remember that the 33rd channel corresponds to f=1.456 kHz and the 56th channel corresponds to f=5.111 kHz

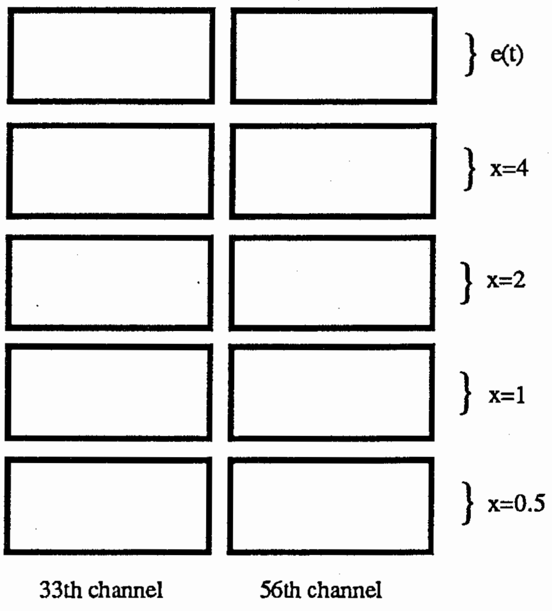
When x=4, the GSD is very sensitive, is seen mainly in the highest channel represented. In this case the different consonant amplitudes do not clearly appear while for the 33rd channel when x=4 then always appear. The GSD is also sensitive but the error caused by the number of samples defining the delay T is less important. Therefore, the GSD output shape difference between x=4 and x=0.5 also appears but is less evident than for the high channels.

Consequently, the first letter of the sentence "s" plotted at the beginning of the graph and the "**tsu**" represented in the middle of the graph are more clearly represented on the high channel number 56 when x=0.5. In this case, the average length which is equal to two period samples, compensates for the low sample number giving the delay T used for the comparison into the GSD processing. We can also observe how the average length improves the performance of the signal periodic detection for the 33rd channel. Indeed the two peaks representing the "**tsu**" appearing in the middle of the drawing were very important compared to the other componants for x=4 while the amplitude of every consonent for x=0.5 is more clearly represented according with what we were expecting for this channel.

In conclusion the average length enables the GSD to underscore the importance of every consonant according to the channel frequency, which is very important for speech recognition. The average length also compensates for the sample number corresponding to the delay T which improves with the frequency values.

Therefore, we finally choose : **x = 0.5**

The final output depends also on the final non-linear function as we are about to see in the following section.



$$\text{AVE\_LENGTH} = \frac{f_{\text{sampling}}}{f_{\text{input}}} \cdot \frac{1}{x}$$

Organisation of the figure 3.3.14 plotted on the following page.

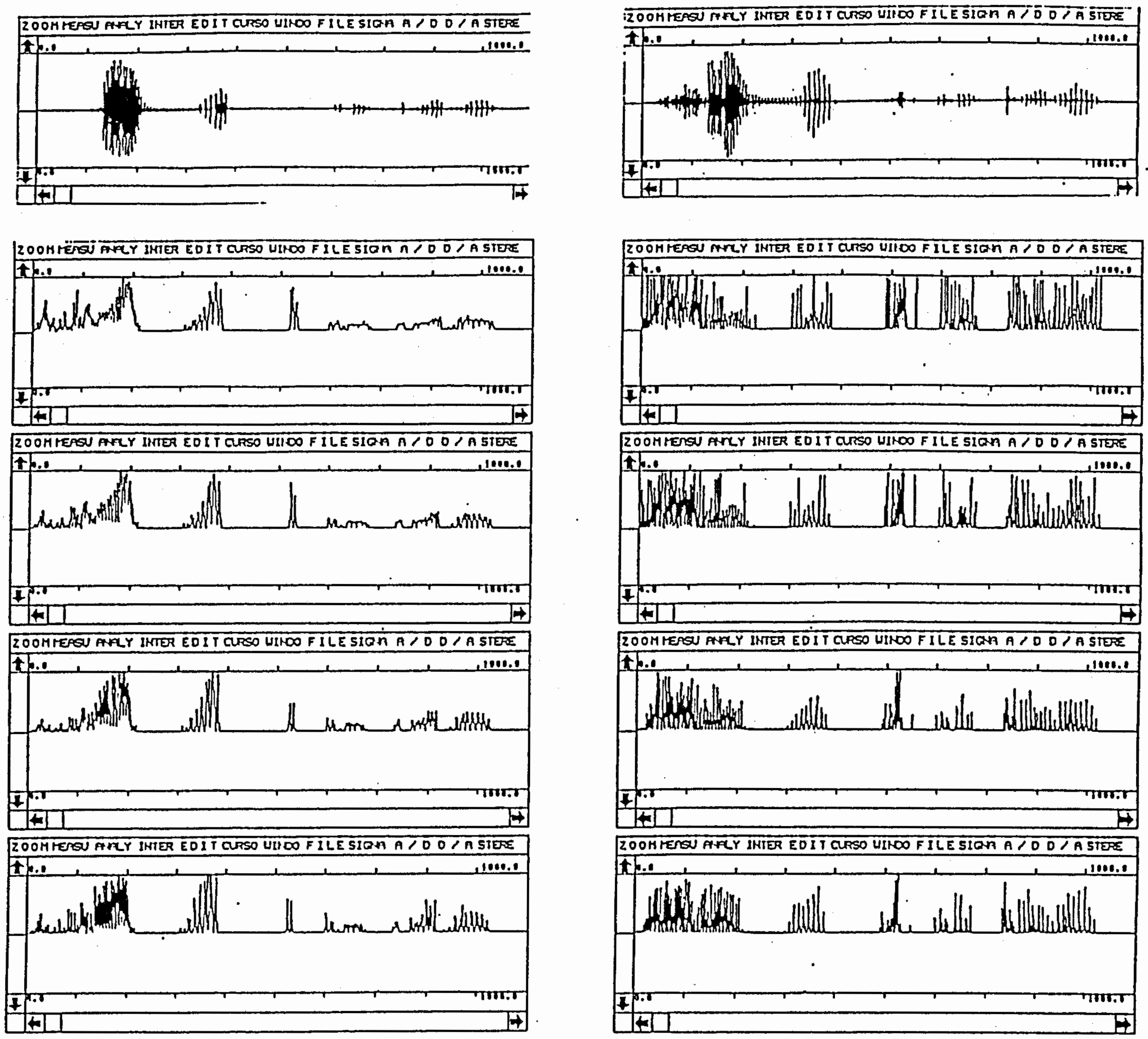

ZOOM MEASU ANALY INTER EDIT CURSO WINDO FILE SIGNA A / D D / A STERE
1000.0

figure 3.3.14 : role of the integration length window

(C) Role of the Satulating Half Wave Rectifier

As we can see in figure 3.3.15 this last component of the GSD model is very important for the high channel because it "normalizes" the amplitude of the quotient +/- output increasing the low amplitudes while the high amplitudes are limited at the maximum of the transfer function. This figure shows the shape of the signal without and with halfwave saturating rectifier for a low channel and a high channel. Therefore we can note that the shape of the signal is almost the same with and without rectifier for the 6th channel whereas for the channel 56th the corresponding channel spectrogram appears thanks to this component. In this way we have used the same kind of non linear function described in the chapter 3.3.1 for two reasons :

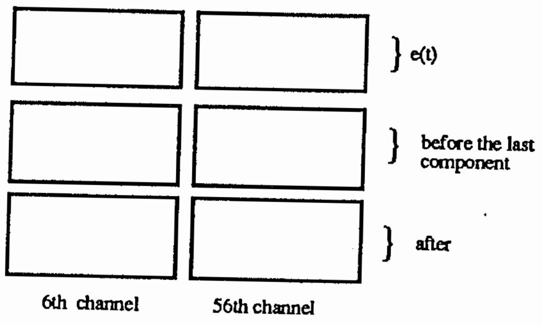
- this function was already implemented in our program
- it seems to be appropriate for what we want to do because we can easily adapt for every channel the shape of the transfer function

Therefore the final saturating halfwave rectifier is defined mathematically as follows where we use only the positive part of the arc tangent function because the signal coming from the quotient +/- is always positive :

$$y = \frac{2000}{\Pi} \cdot \mathrm{atang}\left(\frac{x}{\mathrm{slope}}\right)$$

The parameter we have called "slope" is variable according to the channel number; thus we adjust the final detector sensitivity according to the frequency value. Figure 3.3.18 shows several experimental examples where the value of slope varies for the same conditions of input signal. These results confirm our preceding remarks; that is to say if we look at the 56th channel we can see the appearance of the expected signal as fast as the parameter slope is decreasing, therefore the sensitivity detection is increasing. In our experiment the slope varies between 3.33 and 20 (figure 3.3.16).

Therefore we had to do a compromise solution between the average length (which reduce the important comparison sensitivity for the high channel) and the **slope** which detects the low amplitudes at the output of the quotient +/- in our application. The relation giving the choice of the slope according to the channel number is represented in the figure 3.3.17



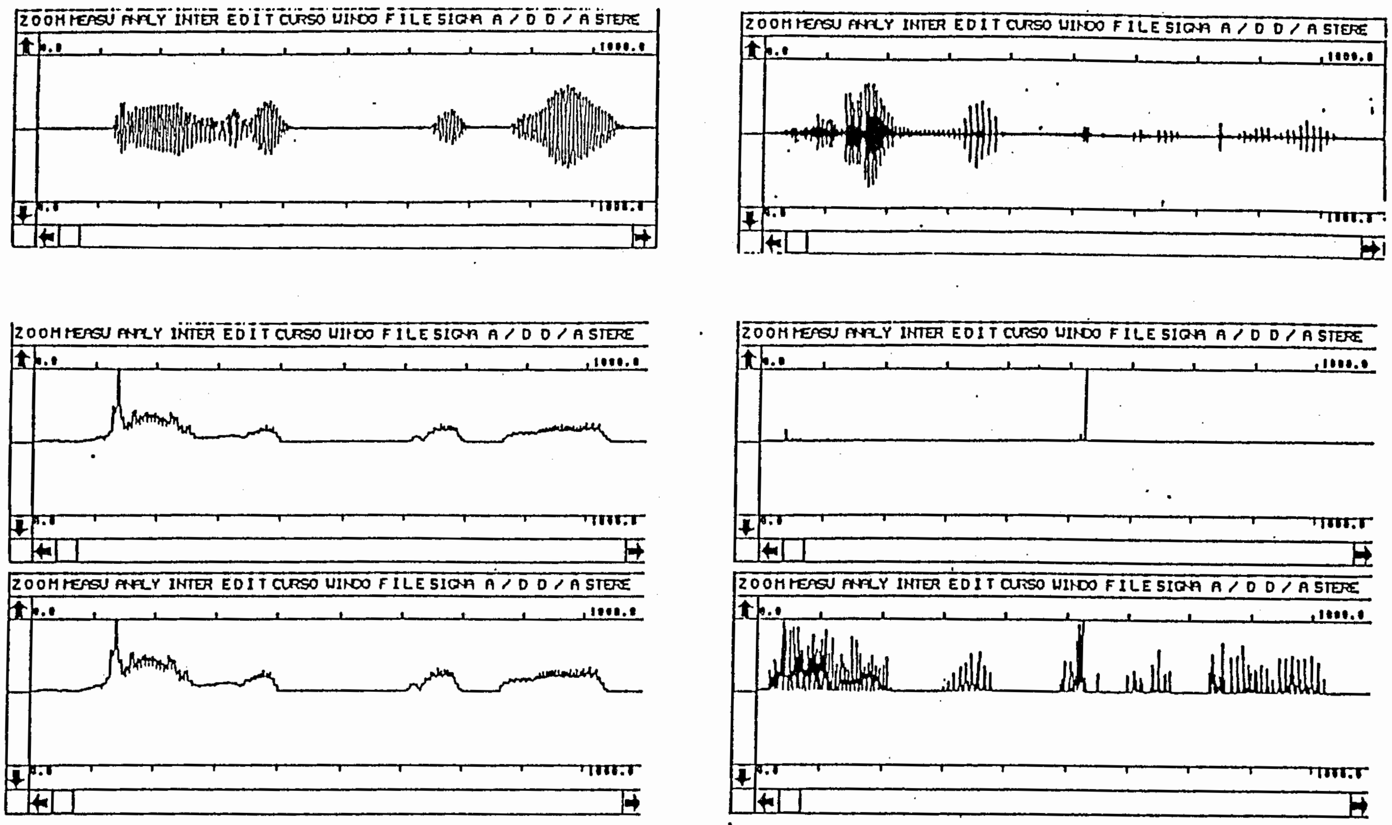


figure 3.3.15 : role of the GSD last component. We plotted on left : signals corresponding to the 6th channel and on rigth the 56th channel. In both case the real speech input waveform are on the top of these figures.

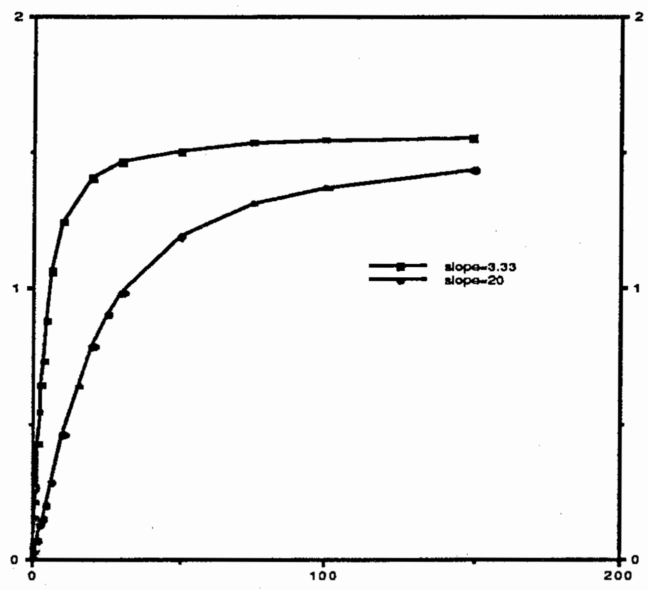


figure 3.3.16 : transfert function of the saturating halfwave rectifier. The two variation limits of the parameter "**slope**" used in our program are plotted on this figure.

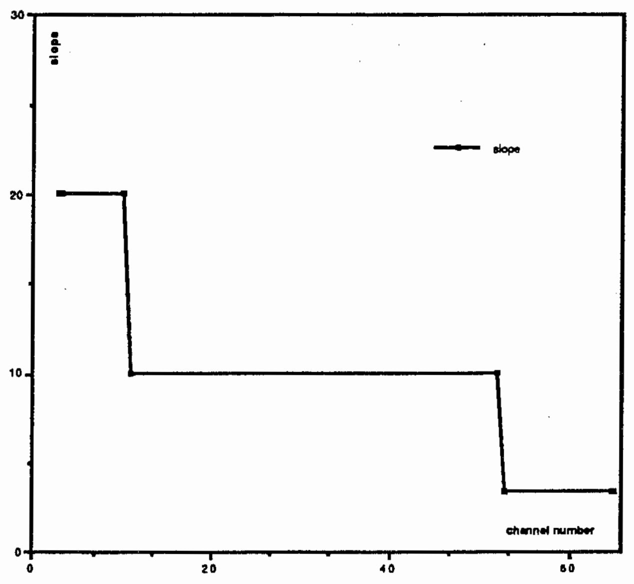


figure 3.3.17 : parameter "**slope**" variation according to the channel number.

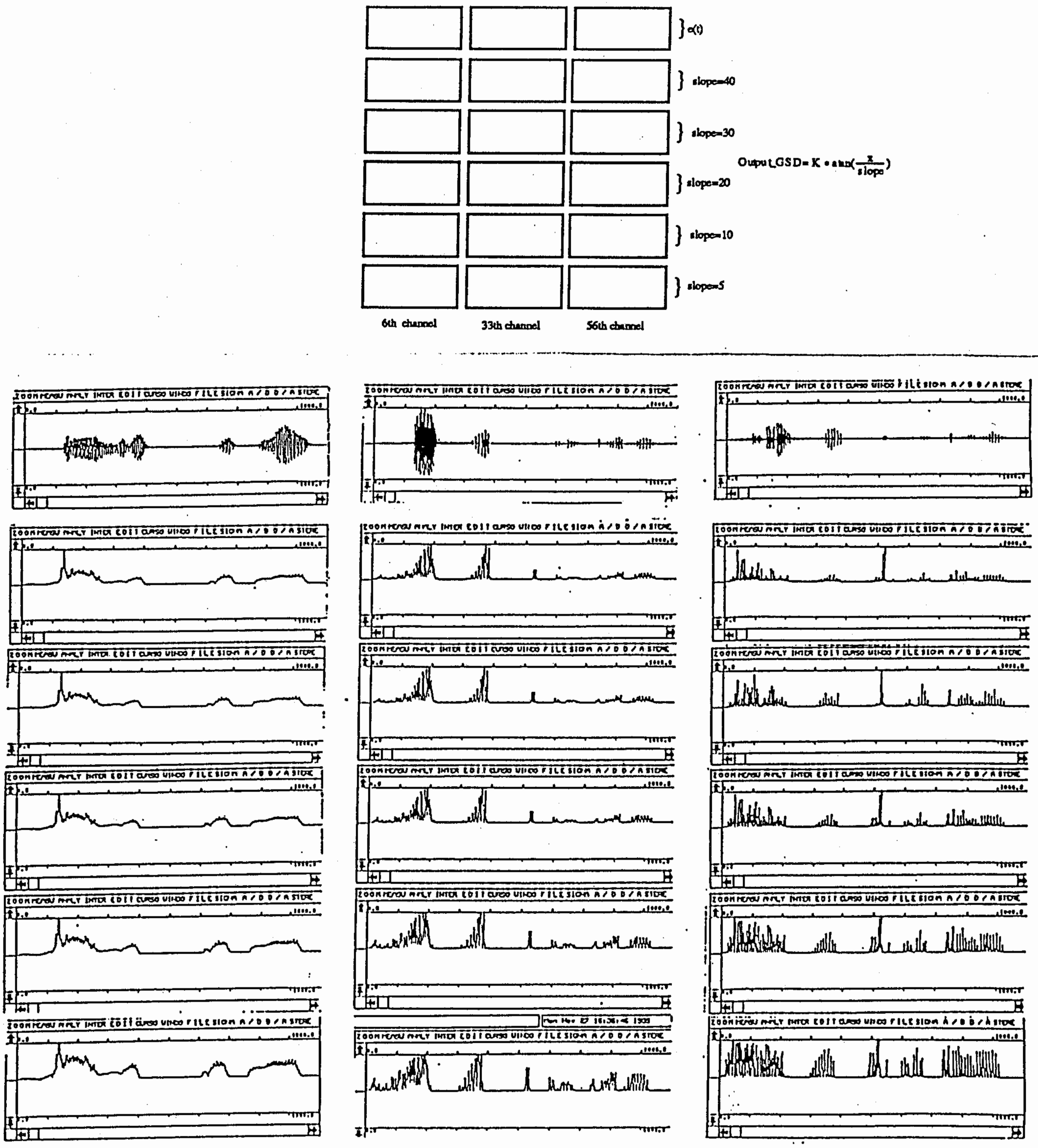


figure 3.3.18 : variation of parameter "**slope**" value showing the sensibility increasing when "**slope**" decreases.

(D) Previous Model and Improvement by Adding a Random Noise Generator

Refer to the figure 3.3.5. The previous GSD model described by Seneff in her paper did not use the noise generator plotted in our diagram. In order to explain why we added this generator we must look at GSD operation without this component. Where the input is a DC signal, let :

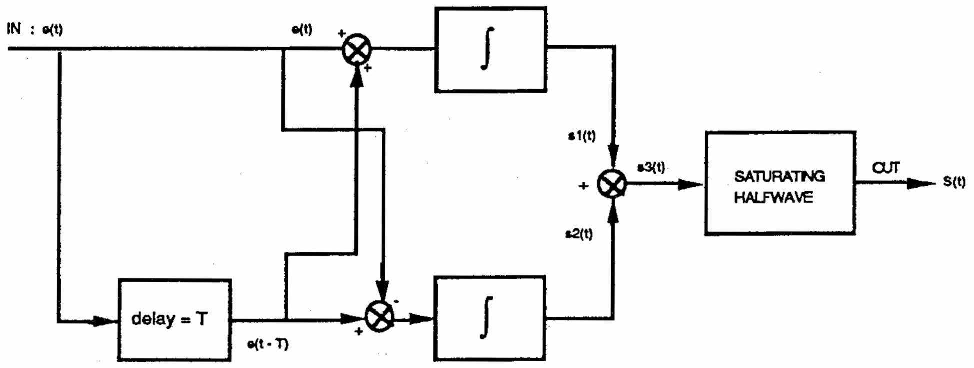


figure 3.3.19 : previous GSD without a noise generator

$$e(t) = \text{constant} \qquad (\text{during } t \geq \frac{1}{\text{finput}})$$

That is to say, let some signal appear at the input which suddenly becomes constant during a time exceeding T. This is not rare because e(t) constant also means e(t) equal to zero, that is to say no speech. When we speak, as everyone knows, we often pause. Indeed, this appears at the top of every waveform plotted above.

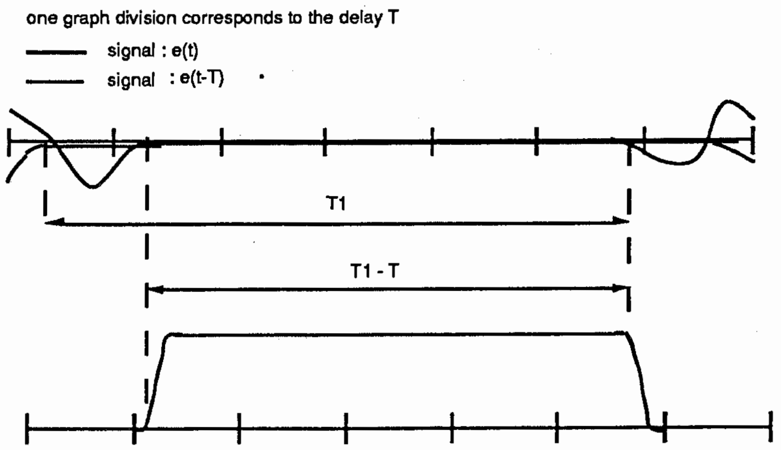


figure 3.3.20 : GSD working with a DC input component and no random noise generator

In figure 3.3.20, the input is a constant equal to 0 in the interval T1. Therefore, the input signal e(t) and the delayed signal e(t-T) are equal during (T1-T). However, the signal does not have the periodicity we are looking for. Remember that the input of our hair cell model is the output of the Basilar membrane made of Q adaptive lowpass filters. We can have a DC component input while the periodicity multiple of every frequency channel (which would also give e(t) - e(t-T) = 0) are attenuated by these Lowpass Filters. e(t) is processed into the GSD in the following steps:

Let $e(t) = \text{constant}$ during $t = 2T$

then $$\left.\begin{array}{l} e(t) - e(t-T) \rightarrow 0 \\ e(t) + e(t-T) \rightarrow 0 \end{array}\right\} \text{ during } t = T$$

therefore $\dfrac{e(t) + e(t+T)}{e(t) + e(t-T)}$ becomes very large

finally $s(t) = \dfrac{\Pi}{2}$ (maximun value of the saturating halfwave rectifier)

Moreover in this case the probability of getting the maximal value s(t) for the high frequencies when we apply a DC component is greater than for the low frequencies because T (and of course also the average length) becomes small. Therefore, the time during which the input signal could be constant in order to increase the output is less important than for the low frequencies.

The principle of the random noise generator which we added in order to check that is simple. With regard again to the preceding steps processed into the modified GSD :

Let $e(t) = \text{constant}$ during $t = 2T$

$$\text{now} \quad s3(t) = \frac{\int_{\text{av\_length}} (e(t-T) - e(t))}{\int_{\text{av\_length}} [(e(t-T) + n(t-T)) - (e(t) + n(t))]}$$

Thus when the input is a DC component equal to 0, s3(t) does not become very large because the random noise generator has different values for n(t-T) and n(t). The difference is not 0 while the sum is equal to 0.

3.3.5.2 conclusion

Seneff's hair cell model was made up of the four elements described in the preceding sections and the Generalized Synchrony Detector was not modified. In this stage, we studied the role of every element in order to see how they modify the input signal from the Q adaptive Basilar Membrane model. Then we put all the elements together and looked at the output s(t). Remember that the Basilar Membrane model is made up of 64 channels of which 61 are used as inputs to the Hair Cell model. Therefore, this model will have to run successively 61 times and at every time the five elements are also successively active. In these conditions the program needs more than twenty minutes for the complete processing.

3.4 Diagram of the Model at this Step of the Report: -Comparison with the Previous Model.-

Looking at the final result we noted the model worked as well without Lowpass filter and AGC as with these elements because these two elements modify the curve included in the envelope. On the other hand, we conserved the halfwave rectifier which is an interface and simulates the unidirectionnal discharge. The Goldhor model, which was the first Hair Cell model, simulates the inner ear time response and the GSD is used for precise period detection.

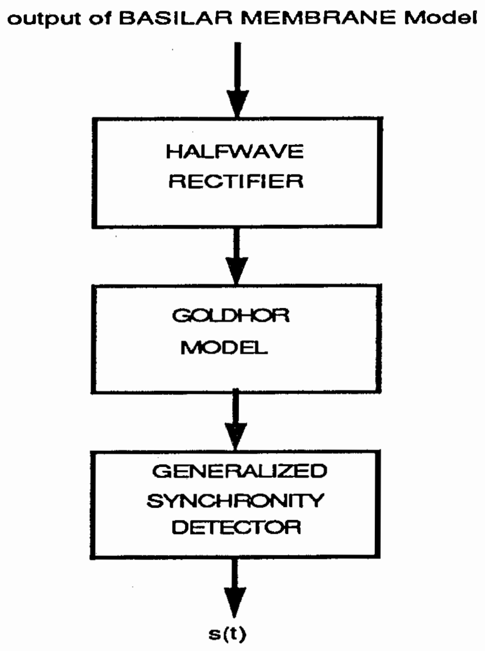

output of BASILAR MEMBRANE Model
HALFWAVE RECTIFIER
GOLDHOR MODEL
GENERALIZED SYNCHRONITY DETECTOR
s(t)

figure 3.4.1 : Diagram of the model used from this point according to our conclusions

3.5. Computer Program

Remember that our program simulates Hair Cell operation along the Basilar Membrane (figure 3.5.1). The program progresses through every channel which equivalent to a different frequency at each moment. Thus, we need to know the corresponding frequency value when the program begins its processing for every channel and also the 20,000 sampled values per channel (figure 3.5.2). In order to process the data coming from the Basilar Membrane model, we organized these results as follow :

- one file contains all the frequencies, saved in ASCII.
- the second contains the 64 times 20,000 sample values (stored in binary values)

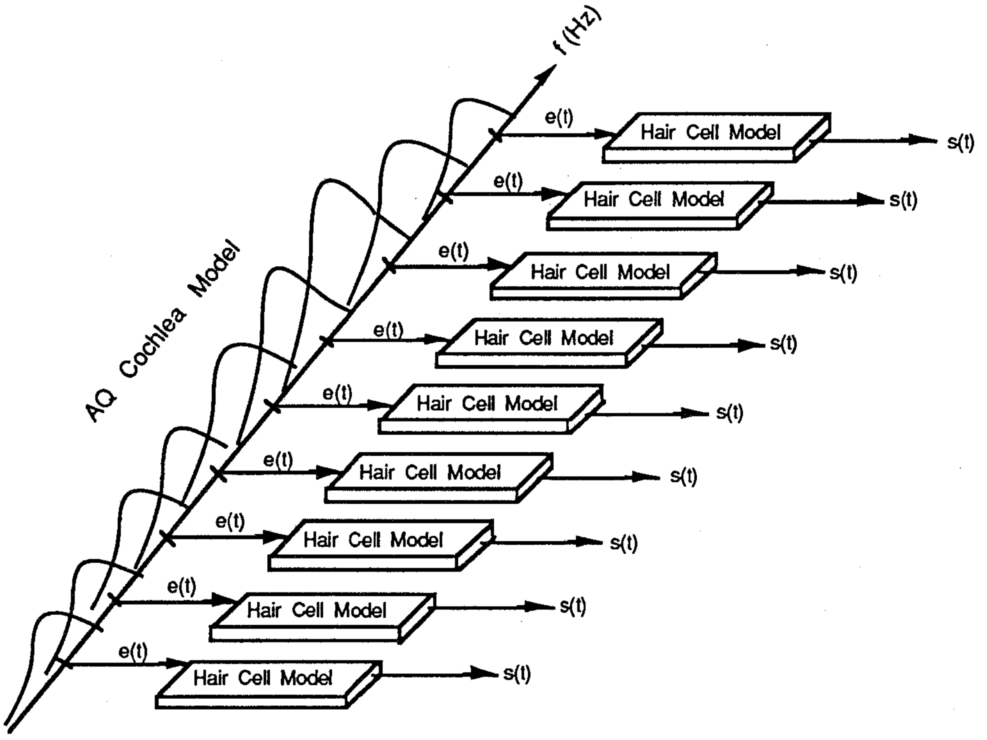


figure 3.5.1 : principle diagram of the AQ cochlea model and the hair cell models working together.

At this stage of the report, we presented briefly how we organized our program in order to better understand under what conditions the following experiments were performed. Therefore, with the hair cell model plotted in figure 3.5.1, we performed several experiments as described below. The results lead to further improvements of this model.

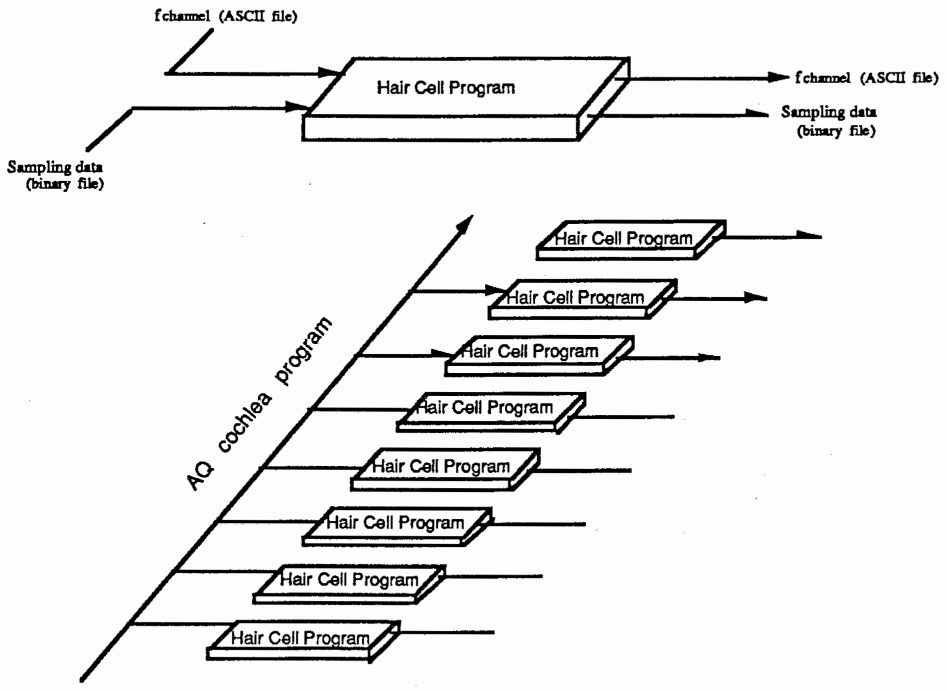


figure 3.5.2 : schematic program organisation

3.6 Working with Clear Speech Data.

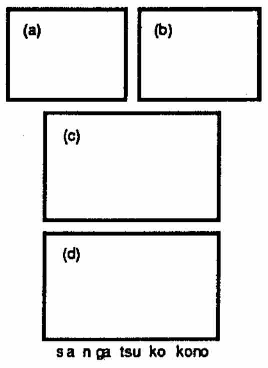
Input speech is the Japanese sentence "**sangatsu kokono**" (March ninth) spoken by a male recorded without noise. Figure 3.6.1 shows half of the total inputs from the AQ cochlea model (a), and step by step the corresponding outputs: halfwave rectifier (b), Goldhor model (c) and GSD (d).

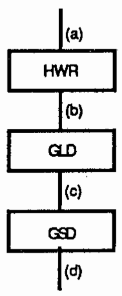
3.6.1. experiments

Previously we also used this real speech data input in order to adjust the parameters channel by channel. We are now looking at the global results with the overall inputs using these parameter values. Overall , the program takes about 15 minutes and about 70% of that time is taken up by the GSD computation. Figure 3.5.2 shows that the program put the data into a file while the frequencies were saved in an ASCII file. Then a second program called *"display"* printed the global synchrony spectrum (figure 3.5.3) from these files. In these figure 1 over 2 channel have been represented and every step of the model calculation has been plotted in order to clearly see the signal processing.

3.6.2. conclusion

The lower frequency is at the bottom of the diagrams and the highest is at the top. Half of the channels have been represented. We use the same scale for every channel in order to compare the amplitudes in every case. The four steps of the signal calculation have been plotted as follow.





The figure above represents how we organized these diagrams. (d) shows the syllables related to time. The figure 3.6.1(a) shows the output of every AQ filter. This figure is another representation of the spectrogram, that is, every channel are displayed line by line. We can see the amplitude of the outputs according to the voice intensity.for each cutoff frequency. Thus, the parts corresponding to the "a" ("san" and "ga") clearly appear for almost every frequency. The end of the sentence "**ko ko no**" is also visible in the lower half of the graph while the "s" ("san" and "tsu") graphic representations are almost completely omitted. The figure 3.6.1(b) is the output of the halfwave rectifier which amplifies the positive low amplitudes and normalizes the large ones at $\pi/2$. Therefore, the parts omitted before begin to appear because they were present before but were too insignificant compared to the maximun values.

Consider the last two figures 3.6.1(c) and (d) and the original sentence. The output of the GLD is already interesting because it contains more information than the previous diagrams. However, the parts corresponding to the "s" whose should be appear for the high channels are still not clearly represented at this stage of the model. The GSD detects the corresponding cutoff period of the signal and in this way, the amplitude is proportional to the corresponding channel period at this stage of the model. Thus, "s", and "**ko ko no**" are clearly represented for the high channels. If we compare the input (a) and the output (d) of our model, we can see that this last one contains more information about the frequency characteristics of the input speech signal.

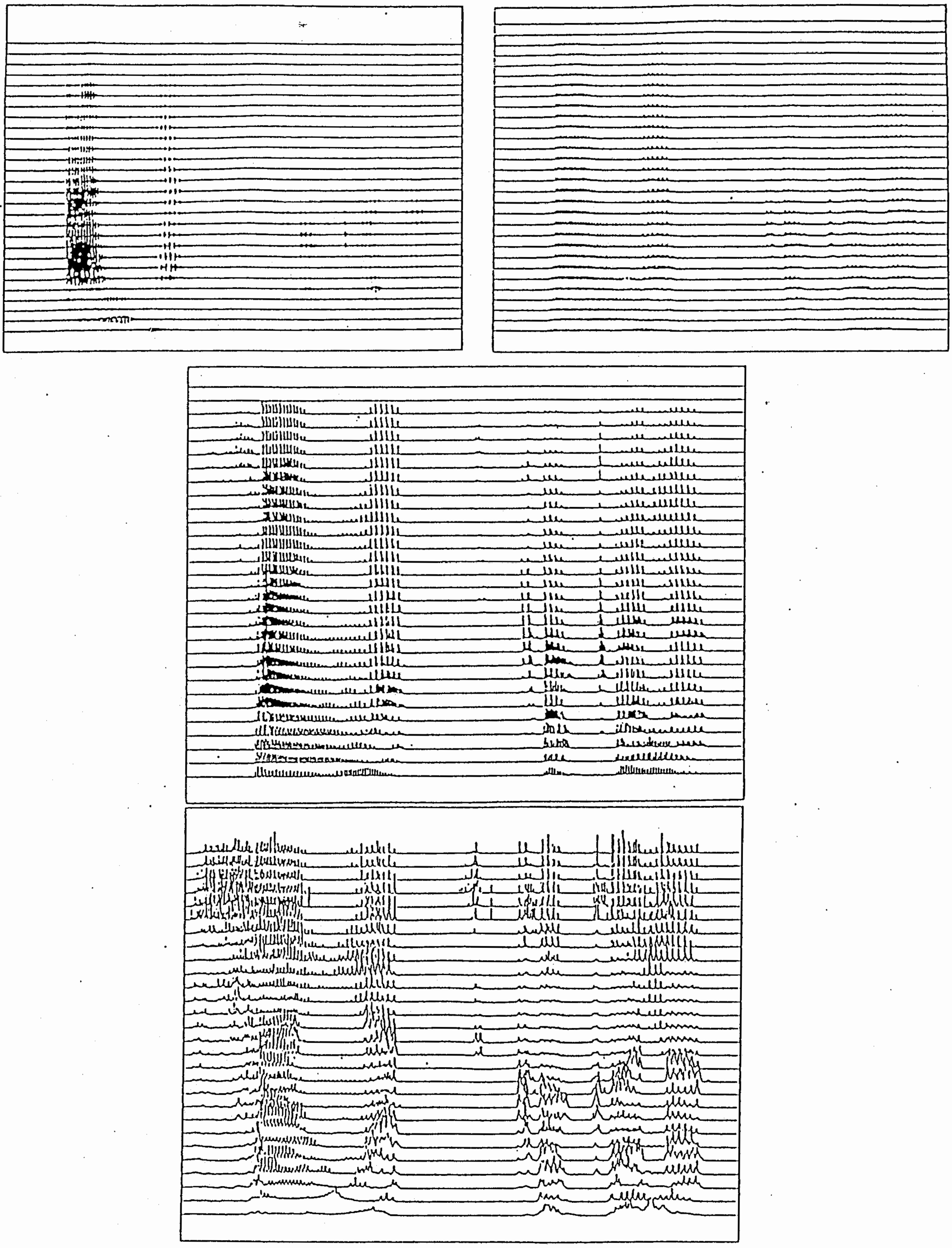


figure 3.6.1 : global working with clear speech data. Every step of the Inner Hair Cell model are plotted. (a) input of the Inner Hair Cell model. (b) output of the halfwave rectifier. (c) output of the Goldhor's model. (d) output of the GSD

3.7. Working in Noisy Environments

Noise is a sound added to the information signal coming into the ear. We must take noise into account when building an auditory functional model because noise is always present in the real environment. In this section, we describe the test performed with this model under noisy conditions.

3.7.1. experiment

We used a file containing the same real speech data with a strong noise added and compared the results both with and without noise. The noise used is a white Gaussian noise having a global S/N ratio approximatly equal to 6dB. This means that for the high frequencies, the speech signal spectrum goes above the constant noise spectrum.

3.7.2. conclusion

At this point of the report, the model can not see the difference between noise and voice signal. The HWR amplifies the positive amplitudes, and it also amplifies the noise. At the output of Goldhor's model, the signal corresponding to the voice input signal can be guessed only when the voice amplitude is more important than the noise amplitude. Consider the global synchrony spectrum in noisy conditions without correction shown in figure 4.7, we can see how the noisy components affect the result as the frequency increases

Thus this functional inner hair cell model works very well for clean speech. However, in the case of the noise component, the result in the high frequency area is affected for two reasons: (1)the amplitude of the noisy component increases as the cutoff frequency increases (because the spectrum of the noise is constant). (2)the GSD works well for the low frequencies.

The results obtained with clean speech data were encouraging. Therefore, in the next section, an improvement of the model for working in noisy conditions will be presented.

4. Modification of GSD for better Noise Reduction

4.1 Introduction

The noise problem consists of three parts: the source, the path, the receiver. Eliminating the problem at the source should be the direct answer but in our case it is assumed that the source cannot be removed. Thus treating the path - our model does - will be used to provide the solution. The receiver is the brain, i.e. the processing computer that follows the auditory model. Therefore the goal is to give as clear a signal as possible to the receiver under conditions of noise added.

Humans have the ability to listen to a voice under noisy conditions because he can selectively focus attention only on the information desired. In this way, several experiments have been done on the auditory peripheral system working under noisy conditions. We also know that noise reduction takes place almost totally in the brain (figure 4.1a). In the case of the functional model studied in this laboratory, the higher level (corresponding to the brain) requires data to be as clear as possible. Therefore, the proportions are inverted as shown in figure 4.1b. In this section we are going to describe how we modified the IHC functional model described above (i.e. HWR and GLD and GSD) in order to get only the periodic voice signal with additional noise.

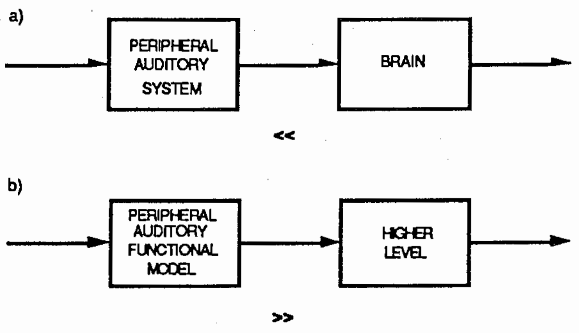

a)
PERIPHERAL AUDITORY SYSTEM
BRAIN
<<
b)
PERIPHERAL AUDITORY FUNCTIONAL MODEL
HIGHER LEVEL
>>

figure 4.1 : In which proportions the information is extracted from noisy environment. (a) case of the human. (b) ideal case of the functional auditory model.

We started from the AQ cochlea filter and IHC functional model and used the properties of the Dirac impulse train filter with Goldhor's model.

4.2 Principle

First, concerning Goldhor's model, remember that we are using the generator voltage as input for the model and the current through the diode for the output. This means that when the input is a real Dirac impulse, it also produces a real Dirac impulse because the capacitor is shorted but the voltage of the capacitor is not affected. On the other hand, for an impulse whose width is only slighly greater than an ideal Dirac impulse, the output becomes an impulse having almost the same width because the voltage of the capacitor - thus the threshold for the diode - is increased. This means that the sensitivity of Goldhor's model is also increased. Thus, we modulate the impulse amplitudes according to the input, that is to say when there is a speech component, the impulse amplitude becomes small in order to conserve the information, and vice versa.

Moreover, the width of the impulse (or the maximal amplitude) varies as the channel is increased in order to mask the noise signal (figure 4.4). Thus, if the time between two impulses is large compared to the time constants of Goldhor's model, we observe at every input impulse the response of the model at one impulse. Therefore, the time separating two impulses is set at 0.1s corresponding to the discharge time constant of Goldhor's model.

On the other hand, consider the spectrum in the case of a Dirac impulse train with a period equal to 0.1s: this is also a Dirac impulse train spaced at 10Hz. Therefore, the impulse train maintains only the periodic components of the signal h(t) by sampling its spectrum every 10Hz whereas the noise is removed. In these conditions, we perform a "selective masking". That is, at the output of the GLD model we first find the previous Dirac impulses train again - because the GLD response for a Dirac is a Dirac - and the speech signal obtained from the spectrum sampled. The Dirac impulse train is then removed by subtracting the signal from the GLD model from the previous impulse train multiplied by a coefficient K and the negative components are eliminated by an inverted diode. Finally, this signal is put into the GSD. Figure 4.2 illustrates this reasoning concerning this part of our functional model . We plotted the frequency area and the time area in the same diagram in order to better understand the reality. Figure 4.3 then compares our previous functional model work (left side) and the IHC functional model which removes noise by adding Dirac correction (right side).

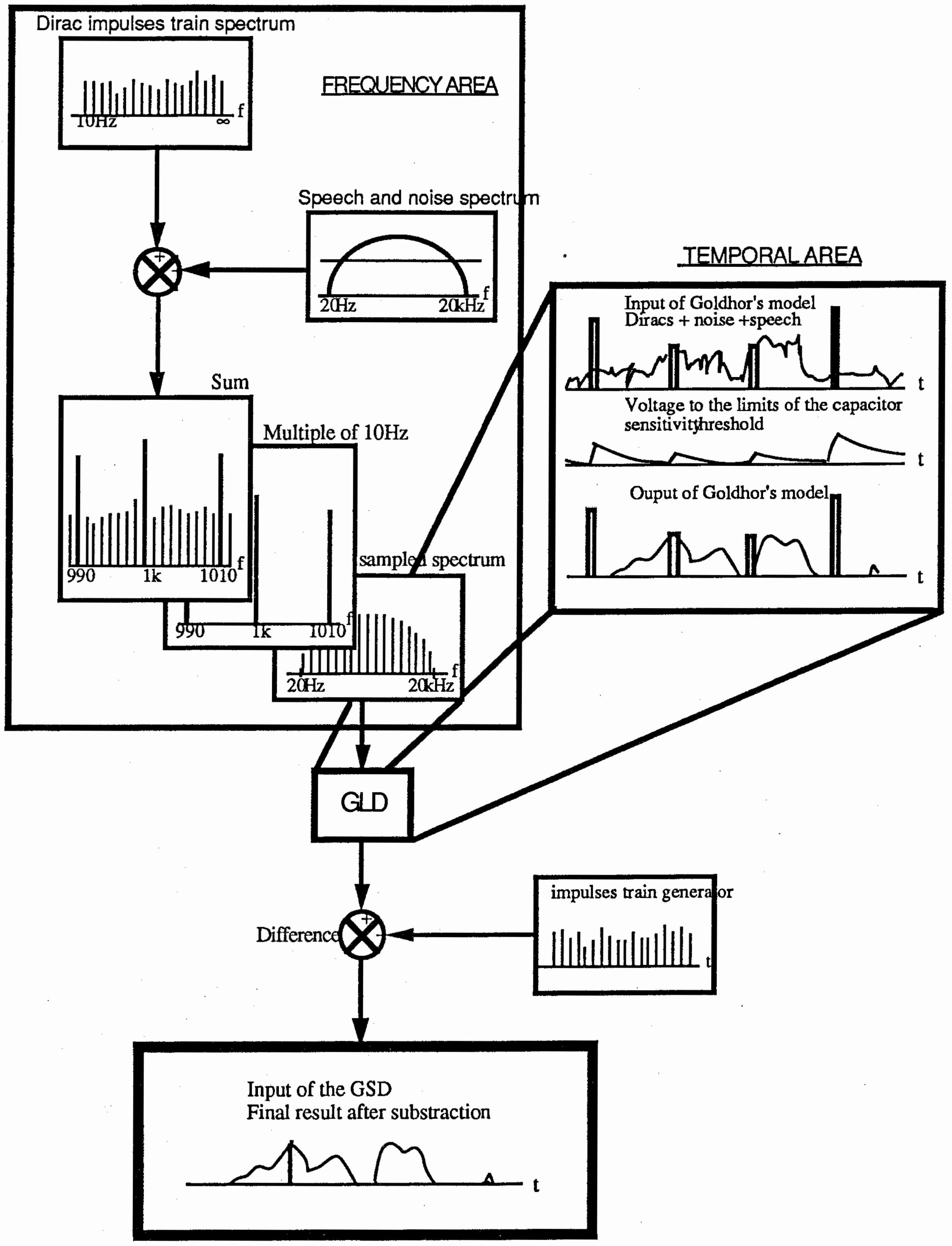


figure 4.2 : detail of the GSD working with impulse train correction

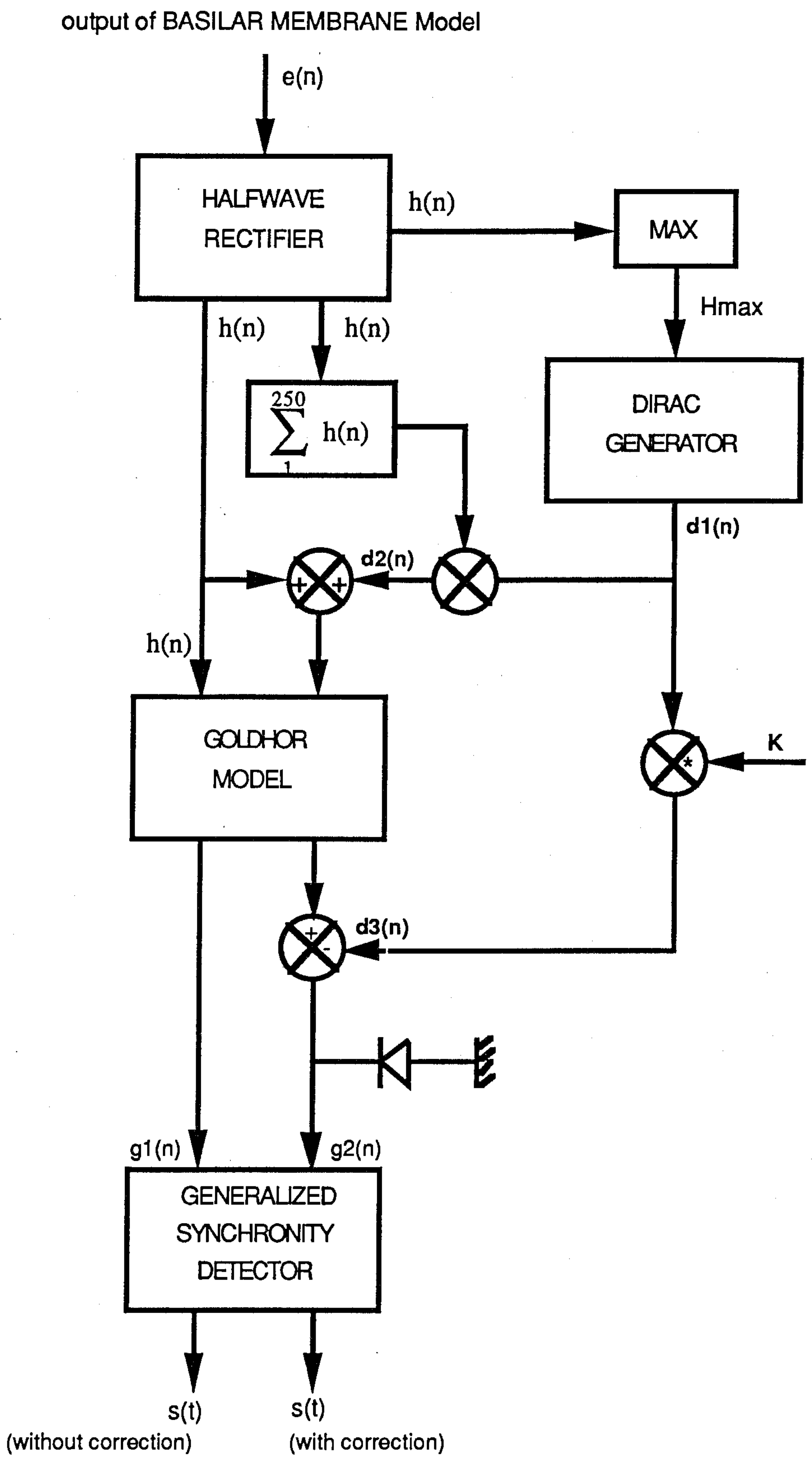


figure 4.3 : Inner hair cell functional model removing noise by adding Dirac correction.

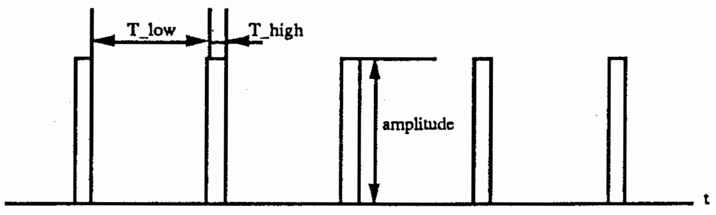
4.3 Experiments

These experiments were done under the same conditions as in the previous section. We always used a signal having S/N=6dB. Thus, we will be able to compare these results later. The program does take more time with this modification when it is running for every channel. Figure 4.5 represents every step of the signal processed into this model for three channels. **The input e(n)** composed of real speech data with additional noise is plotted at the top of the figure. Each columm corresponds to the 10th, 20th and 45th AQ cochlea functional model channel. We memorize the maximal value Hmax for every channel output of the HWR processing [**h(n) plotted on the 2nd line**]. This is put into the impulse generator while the sum of the following 250 samples of h(n) is recorded after every impulse onset. Thus the Dirac impulses [**d1(n)**] coming from the generator are modulated with these sums. Therefore, these impulse amplitudes depend on the following signal s(n) amplitudes in such a way that we can see how the impulse amplitudes become large when there is no speech signal data and vice versa [**d2(n) /3th line**]. Then, the resulting signal d2(n) is added to h(n) [**4th line**] and put into the GLD model. The 5th line represents the output of the GLD model and we can already see how the parts corresponding to the large amplitude noisy signal are removed while the impulse train subsists. The **6th and 7th lines** show the result after subtracting the Dirac pulses. Finally, **the last figure** shows the result after GSD processing. In this figure we can see how the noise has been removed.

The impulse parameters used in our program were fixed, except for the maximal amplitude which increased according to the channel number caused by parameter K. The figure 4.4 shows how this parameter varies while formula XX shows how the amplitude of the impulses is calculated according to all the parameters.

T_low = 0.1s (number_samples = 2000)

T_high = 2.5 ms (number_samples = 50)



$$\text{Impulse\_amplitude} = \frac{K \bullet H\max}{\frac{1}{250} \bullet \sum_{250} h(n)} \qquad (XX)$$

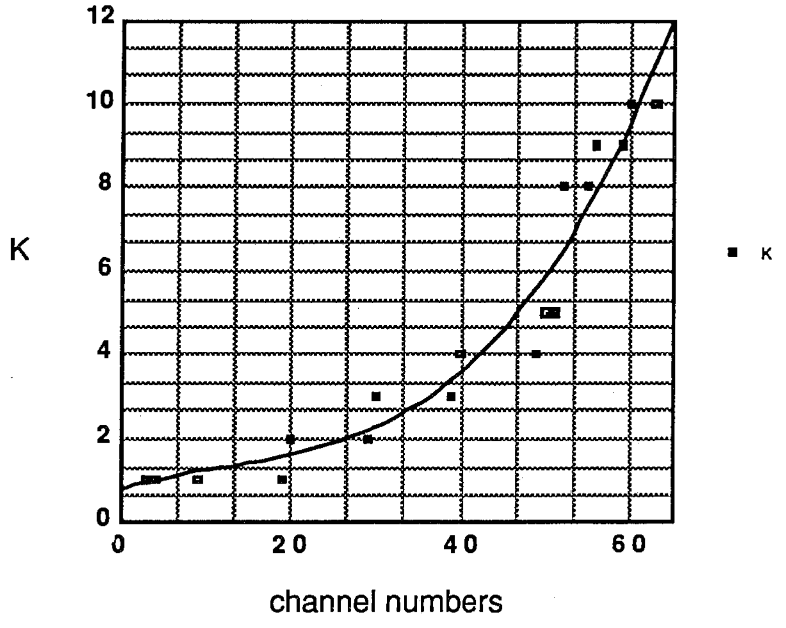
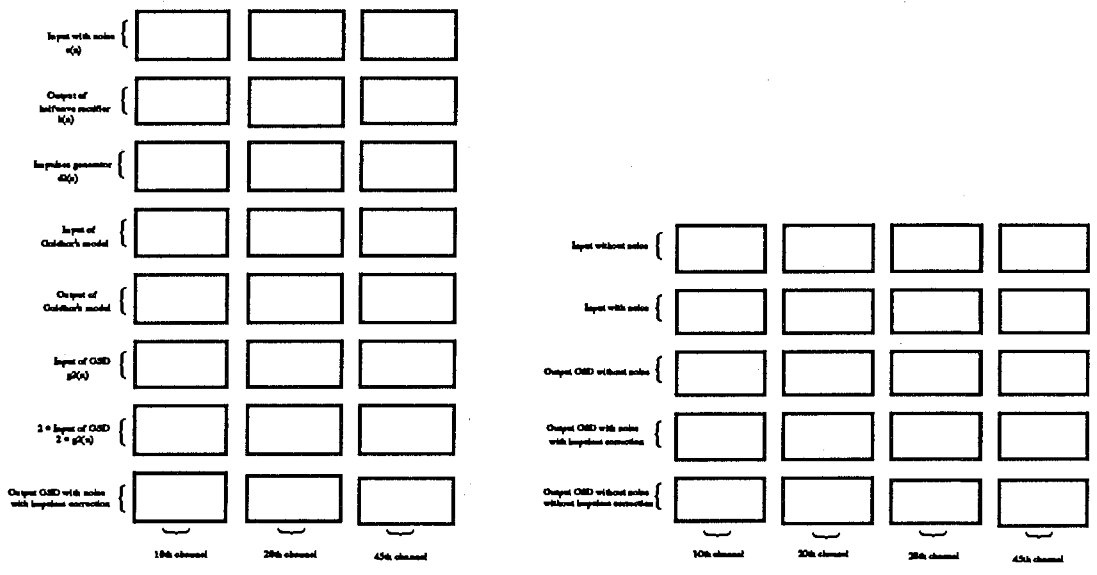


figure 4.4.: increasing impulse amplitudes as the channel number increasies caused by the parameter K.

The program is written so that we can quickly compare, channel by channel, the results obtained with and without impulse correction for the same input data. That is what it is done on the figure 6.9.6.



Input with noise x(n)
Output of halfwave rectifier h(n)
Impulses generator d(n)
Input of Goldstein's model
Output of Goldstein's model
Input of OSD g(n)
2 * Input of OSD 2 * g(n)
Output OSD with noise with impulses correction
10th channel
20th channel
45th channel
Input without noise
Input with noise
Output OSD without noise
Output OSD with noise with impulses correction
Output OSD without noise without impulses correction
10th channel
20th channel
28th channel
45th channel

Organisation of the figures 4.5 and 4.6 plotted on the following page.

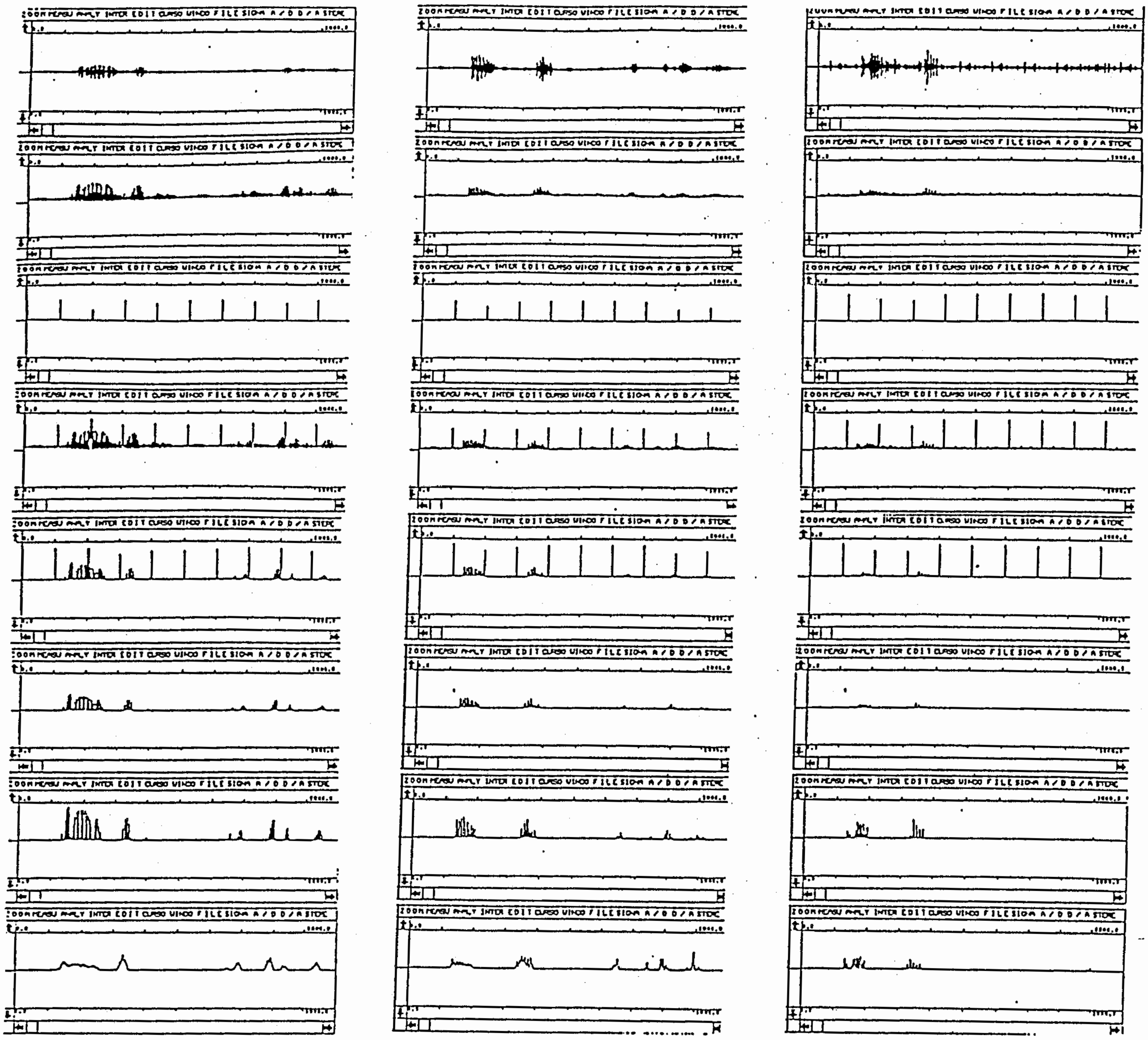


figure 4.5 : every step of the signal processed in the IHC model by adding Dirac correction. Following the notations used on the figure 6.9.1, we plotted from top to bottom : the input of the IHC model e(n) / the output of the HWR h(n) / d2(n) / h(n)+d2(n) / g2(n)+d3(n) / g2(n) / 2•g2(n) / and finally s(n).

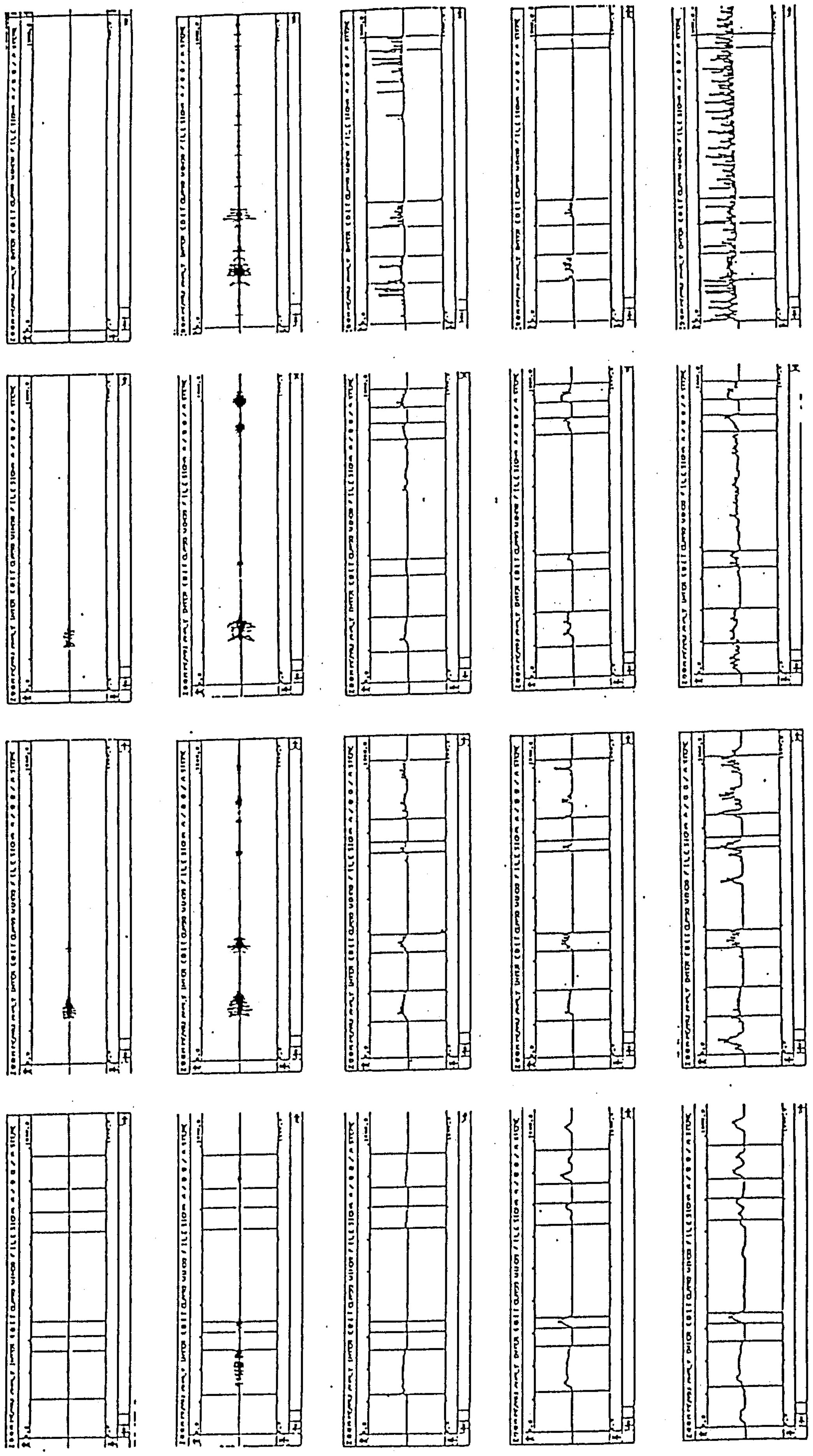


figure 4.6 : comparison of the results obtained with impulse correction and without impulse correction for four channels following the array plotted on the page XX.

4.4 Conclusion

Figure 4.6 shows how the impulse correction modifies the result and removes the noise compared to the preceding result. We have pointed to the different parts of the signal in order to easily compare every area. **The inputs** of our IHC functional model are plotted on the first two lines with the same scale and we can see how the noise amplitude improves as the channel number increases. On the last line representing **outputs of the GSD in noise condition**, we can see how the noise disrupts the GSD without impulse correction since it is impossible to separate the speech data effects and the noise effects after processing. Therefore, the last line of the GSD output shows how the noise affects the model without correction as the channel number increases.

The fourth line represents the **outputs of the GSD with correction, given noisy input.** We can see two important results that follow from using the corrector : (1) In the areas where there was no speech in the original signal, the output of the GSD is 0.0, indicating that noise has been fully removed in those areas. (2) In the areas where speech was present in the original signal, the output of the GSD is very similar to the speech + noise portion of the fifth line in the diagrams because it is impossible to obtain the same detail shapes since the input as been affected by the noise added to the previous speech signal.

Comparing the third line and the fourth line, we rediscover the same global forms, that is the output of the GSD with correction reacts in the speech areas and vice versa. Moreover, we noted that the rhythm of the sentence "*san na tsu ko ko no*" is conserved at the output of the GSD using correction whereas it was impossible to listen to it in the case of the fifth line.

Figure 4.7 represents the global synchrony spectral diagrams obtained in the three cases. Half of the total channels have been represented. The synchrony spectrum obtained in the case of our previous model given clear input is plotted at the top of the figure. The same model used with noisy input gives the diagram plotted in the middle of the figure. Finally, the synchrony spectrum with noisy input with correction is visible below. If we compare these results, we can observe how the noise appears especially for the high channels in the areas where there are no speech or very low speech amplitude compared to the noise. We can see that at the beginning and in the middle of the sentence. These parts have been removed by the impulse correction.

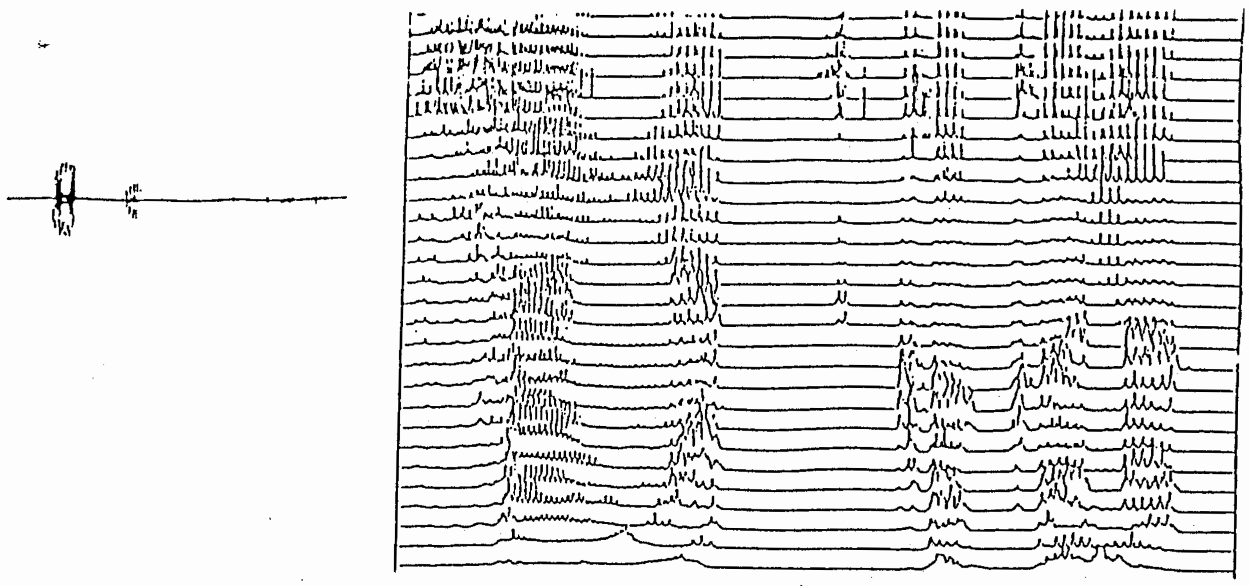
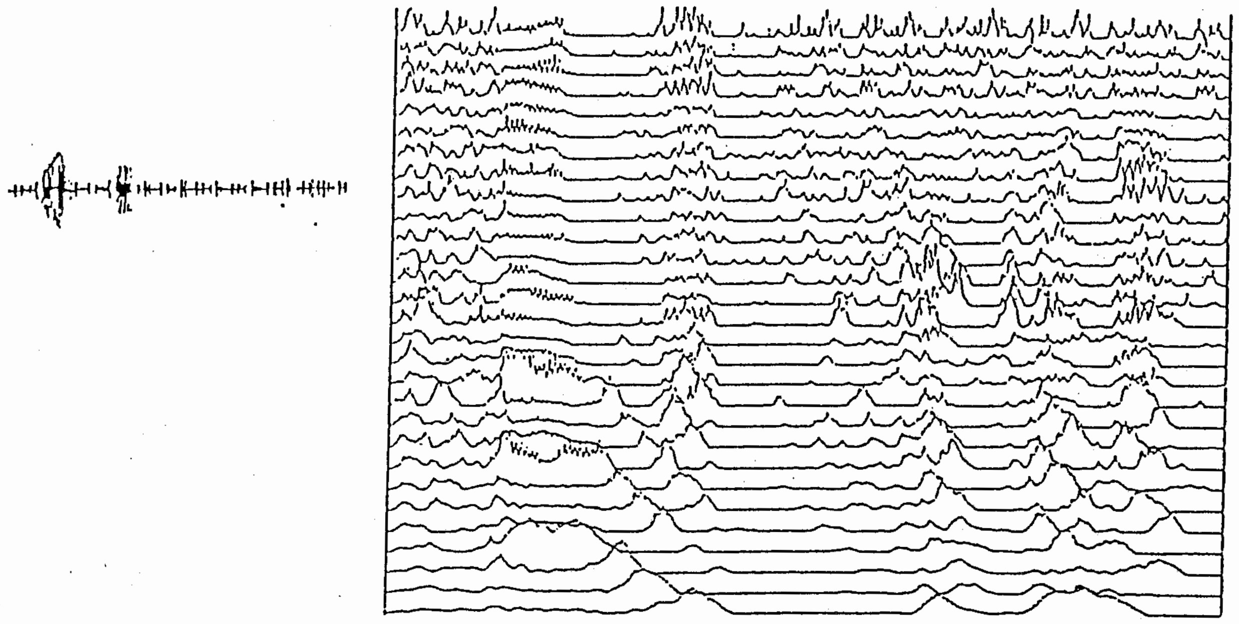
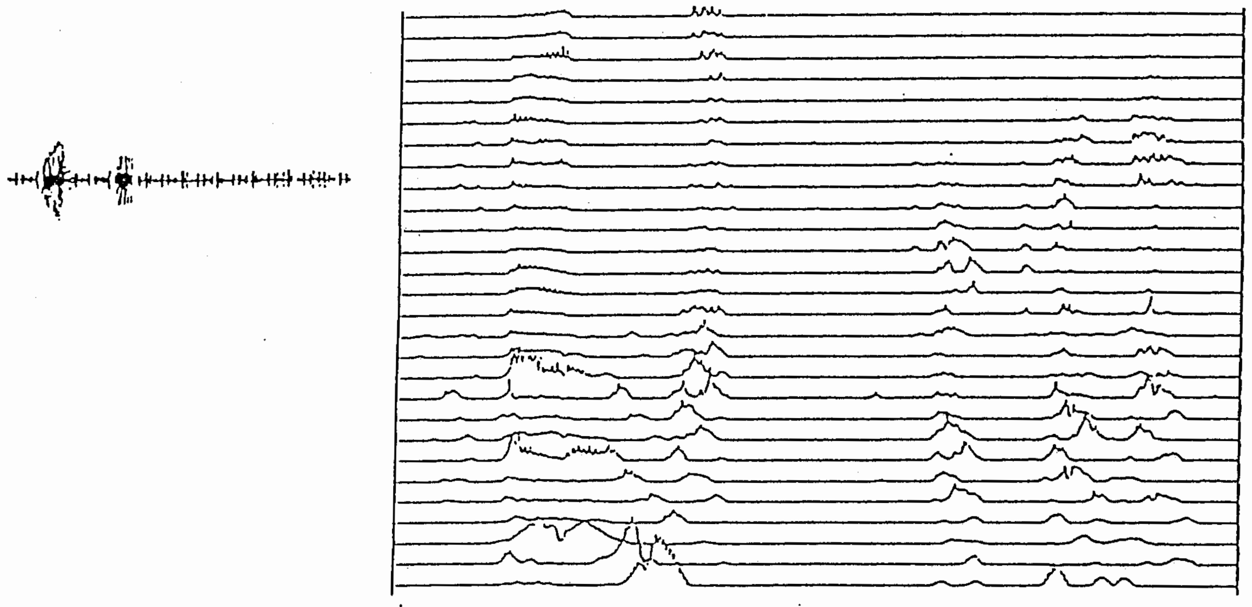




Figure 4.7 : Synchronicity spectrums. in order : previous model output given clear input; previous model output given noisy conditions; model with corrector output given noisy conditions. Half of the channel are represented. The highest channel is on the top for the three diagrams.

7. Conclusion

First, we carefully studied Seneff's inner hair cell functional model. We clarified the problem when using the model with an adaptive Q cochlear filter bank. That is, each stage of the model is largely influenced by the input level, since all stages except the lowpass filter are nonlinear circuits. Moreover, the original model parameters chosen for Seneff's cochlear filter bank, were not always suitable for other cochlear filter banks, i.e. in our case the adaptive AQ cochlear filter bank. Thus, we changed several parameter values of her previous model such as in the halfwave rectifier and the AGC in order to get the same results as her.

Second, we modified the GSD circuit to obtain a stable output because: (1) the sensitivity of the original GSD circuits was too high, and (2) it often provided unexpected output particularly for higher frequency channels and for the DC component of each channel. We solved these problems by: (1) adding small fluctuations to the synchrony detector input signal and (2) using a frequency dependent appropriate integration period for the GSD. These modifications made the GSD output stable and reliable. In addition to that, we removed the LPF and AGC stages from the original model because, we found that those stages affected the GSD operation only slightly. This modification made the model simpler and we saved considerablecomputation time.

Finally, a new noise reduction algorithm using a Dirac impulse train was proposed. The GSD works well for clean speech. However, it works well only in the lower frequency channels for noisy speech. When this Dirac correction algorithm was used in the GSD, it worked well in any channel for noisy speech. In all the extended GSD output representations, we saw the Dirac correction algorithm seemed to effectively suppress the noise. However, it is necessary to make quantitative study, such as a speech recognition experiment using this algorithm, to give a proper evaluation.

ACKNOWLEDGEMENTS

This work constitutes the internship report of Eric Tardif, who would like to express his gratitude to all the people in the ATR Auditory and Visual Perception Research Laboratories, who have helped him during this internship and contributed to making a very friendly work atmosphere.

He would first like to thank Dr. Koohei Habara, ATR vice-president, for his consideration in arranging the internship.

He would like to thank Dr. Eiji Yodogawa president of the ATR Auditory and Visual Perception Research Laboratories, who made my internship at ATR possible.

He is also very grateful to Dr. Yoh'ichi Tokhura, head of the Auditory Perception department, who gave me friendly scientific advice. This internship would have been impossible without Dr. Tohkura's dynamic and efficient department leadership, and his energetic promotion of international scientific cooperation.

He wish to express sincere thanks to Dr. Tatsuya Hirahara for the effective way in which he supervised my work, his continual support and very helpful ideas and advice during the length of my stay at A.T.R. Working with him was an interesting, challenging, and fun experience.

He is indebted to many in the Auditory and Speech Perception Department at ATR, for their help in the various stages of this research, particularly Mr. Seiichi Tenpaku for his very useful programming advice.

He would also like to thank the secretaries in the ATR Auditory and Visual Perception Laboratories for their kindness and constant and effective help.

Thanks to Dr. Carl Muller for checking this report and for his fruitful comments.. Finally special thanks to Erik McDermott for his constant presence in many cases during my stay in Japan.

9. References

D. Crane Hewitt (1983). "IHC - TM Connect - Disconnect amd Mechanical Interaction among IHCs, OHCs, and TM".

E. C. Carterette & M. P. Friedman (1978) "Handbook of perception (vol 4). Hearing"

J. Conrad & J. Hemond (1983) *Engineering acoustic and noise control*

T. Hirahara & T. Komakine (1989) "A computational cochlear nonlinear preprocessing model with adaptative Q circuits" Proc. ICASSP'89 Vol.1, pp.495-499

D. Imbert (19??) *Audition*

B. C. J. Moore (1982) *An introduction to the psychology of hearing*

J. O. Pickles (1982) *An introduction to the physiology of hearing*

D. O'Shaughnessy (1987) *Speech communication. Human and Machine*

S. Seneff (1985) "Pitch and Spectral Analysis of Speech Based on an auditory Synchrony Model". Technical Report 504 Massachusetts Institute of Technology. Cambridge, Massachusetts

S. Seneff (1988). "A joint synchrony/mean-rate model of auditory speech processing". Journal of Phonetics, Vol.16, No.1, pp.55-76

V. J. Tobias & D. E. Schubert (1983). *Hearing Research and Theory*, Vol. 2. .

E. Zwicker & R. Feldtkeller (1981) *Psychoacoustique. L'oreille recepteur d'informations*,

```
/*********************************************************************

    Hair cell model according to Optimun Designing
        Cascade/Parallel type Cochlear Filter

                  25 oct 89
                  E.Tardif (ATR Japan)

**********************************************************************

    Modified by   T.Hirahara
                  November 30 1989

**********************************************************************

    Modified by   E.Tardif
                  December 10 1989

*********************************************************************/

# include       <stdio.h>
# include       <sgtty.h>
# include       <errors.h>
# include       <errno.h>
# include       <sys/types.h>
# include       <sys/ioctl.h>
# include       <sys/stat.h>
# include       <fcntl.h>
# include       <ctype.h>
# include       <math.h>

# define        MAXLEN 128            /* Maximum length of char array */

# define        Fs      20            /* sampling frequency [kHz] */
# define        N_CH    64


main( argc, argv )
int  argc;
char *argv[];
{
  FILE    *file1;
  FILE    *file2;
  FILE    *file3;
  FILE    *file4;

  char    *p, *q, *r, *o, *malloc();
  char    freq_file[MAXLEN];
  char    input_file[MAXLEN];
  char    para_file[MAXLEN];
  char    output_file[MAXLEN];
  char    frequency_file[MAXLEN];

  char    fname_freq[MAXLEN];
  char    fname_spec[MAXLEN];
  char    fname_rslt[MAXLEN];

  char    *dir1 = "/usr3/tardif/SENEFF/";
  char    *dir2 = "/CONTIG/";

  char    char_dummy[MAXLEN];

  float   F_list[65];
  float   *spec_array;
  float   *array1, *array2, *array3;

  int     i, j, k, K;
  int     length, size;
  int     chno, chno_check;

  float   F, f;                      /* signal input frequency [kHz] */
  float   Maxh;                      /* Max of Halfwave output */

  float   A, B;                      /* parameters of the first half wave */
  float   R, Re, C, T1, T2;          /* Goldhor's model constants */
  float   mua, mub;                  /* Goldhor's model time constants */
  float   Tlow, Thigh;               /* Dirac's correction parameters */
  float   TAUX1;                     /* low pass parameters */
  float   TAGC, KAGC;                /* AGC parameters values */

  int     GSD_delay;
  int     GSD_ave_length;
  float   GSD_GAIN;                  /* GSD atan(x/gsd_gain) */
```

```
    int      SAVE_INP ;
    int      SAVE_HLF ;
    int      SAVE_DIR ;
    int      SAVE_GLDE;
    int      SAVE_GLDSIG;
    int      SAVE_GLDD;
    int      SAVE_GLD ;
    int      SAVE_LPF ;
    int      SAVE_AGC ;
    int      SAVE_GSD ;
    int      SAVE_OUT ;
    int      CORRECT  ;
    int      REVERSE  ;

    nice (-20);

    /* ----- Read Controle Files ----- */

    if( argc <= 1 )
      {
        printf("\n first usage is as follows;\n");
        printf("-f [frequency_file]\n");
        printf("-i [input_file]\n");
        printf("-o [output_file]\n");
        printf("-p [parameter_file]\n\n");
        printf("-- without Dirac correction\n\n");
        printf("-I save switch for INPUT.DATA\n");
        printf("-h save switch for HALF.DATA\n");
        printf("-d save switch for DIR.DATA\n");
        printf("-s save switch for GLDSIG.DATA\n");
        printf("-g save switch for GLD.DATA\n");
        printf("-l save switch for LPF.DATA\n");
        printf("-a save switch for AGC.DATA\n");
        printf("-G save switch for GSD.DATA\n\n");
        printf("-A save switch for all\n\n");
        exit(0);
      }

    strcpy(freq_file   , "FREQ_LIST");
    strcpy(para_file   , "HAIRCELL.PARA");
    strcpy(input_file  , "foo.inp");
    strcpy(output_file , "%Gsd");         /* foo.out"); */
    strcpy(frequency_file   , "%Frequencies");

    SAVE_INP    = 0;
    SAVE_HLF    = 0;
    SAVE_DIR    = 0;
    SAVE_GLDE   = 0;
    SAVE_GLDSIG = 0;
    SAVE_GLDD   = 0;
    SAVE_GLD    = 0;
    SAVE_LPF    = 0;
    SAVE_AGC    = 0;
    SAVE_GSD    = 0;
    SAVE_OUT    = 0;
    CORRECT     = 1;
    REVERSE     = 0;

    k = 1;
    while (argv[k][0] == '-')
      {
        switch (argv[k++][1])
          {
          case 'f': strcpy (freq_file,   argv[k++]); break;
          case 'p': strcpy (para_file,   argv[k++]); break;
          case 'i': strcpy (input_file,  argv[k++]); break;
          case 'o': strcpy (output_file, argv[k++]); SAVE_OUT = 1; break;
          case '-': CORRECT     = 0; break;
          case 'I': SAVE_INP    = 1; break;
          case 'h': SAVE_HLF    = 1; break;
          case 'd': SAVE_DIR    = 1; break;
          case 'e': SAVE_GLDE   = 1; break;
          case 's': SAVE_GLDSIG = 1; break;
          case 'c': SAVE_GLDD   = 1; break;
          case 'g': SAVE_GLD    = 1; break;
          case 'l': SAVE_LPF    = 1; break;
          case 'a': SAVE_AGC    = 1; break;
          case 'G': SAVE_GSD    = 1; break;
          case 'R': REVERSE     = 1; break;
          case 'A': SAVE_INP=SAVE_HLF=SAVE_DIR=SAVE_GLD=SAVE_GLDE=SAVE_GLDSIG=SAVE_GLDD=SA
->  VE_GLD=SAVE_GSD = 1; b
  reak;
          default: break;
          }
    }

  /*----------- Read Parameter File -------------*/

  if( (file1 = fopen(para_file,"r") ) != NULL )
    {
      fscanf( file1,"%s %s %s %f", char_dummy,char_dummy,char_dummy,&A);
      fscanf( file1,"%s %s %s %f", char_dummy,char_dummy,char_dummy,&B);
      fscanf( file1,"%s %s %s %f", char_dummy,char_dummy,char_dummy,&T1);
      fscanf( file1,"%s %s %s %f", char_dummy,char_dummy,char_dummy,&T2);
      fscanf( file1,"%s %s %s %f", char_dummy,char_dummy,char_dummy,&C);
      fscanf( file1,"%s %s %s %f", char_dummy,char_dummy,char_dummy,&Tlow);
      fscanf( file1,"%s %s %s %f", char_dummy,char_dummy,char_dummy,&Thigh);
      fscanf( file1,"%s %s %s %f", char_dummy,char_dummy,char_dummy,&TAUX1);
      fscanf( file1,"%s %s %s %f", char_dummy,char_dummy,char_dummy,&TAGC);
      fscanf( file1,"%s %s %s %f", char_dummy,char_dummy,char_dummy,&KAGC);
      fscanf( file1,"%s %s %s %f", char_dummy,char_dummy,char_dummy,&GSD_GAIN);
      fscanf( file1,"%s %s %s %d", char_dummy,char_dummy,char_dummy,&chno_check);

      fclose( file1 );
    }

  else
    {
      printf(" Can not open %s\n",para_file);
      exit(0);
    }


  /*------------ Read Frequency List ------------*/

  strcpy( fname_freq, dir1 );
  strcat( fname_freq, freq_file );

  if( (file1 = fopen(fname_freq,"r") ) != NULL )
    {
      while( !feof(file1) )
        {
          fscanf( file1,"%d %f", &chno,&F);
          F_list[chno] = F;
        }

      fclose( file1 );
    }

  else
    {
      printf(" Can not open %s\n",fname_freq);
      exit(0);
    }

  /*----Read *.spec.# file and decide the array size -----*/

  strcpy( fname_spec, dir2 );
  strcat( fname_spec, input_file );

  if( (file1 = fopen(fname_spec,"r") ) != NULL )
    {
      get_file_length( fname_spec, sizeof(float), &length );
      length = length / N_CH;

      printf("\nnumber of channels = %d\n",N_CH );
      printf("length of datas    = %d\n",length);

      fclose(file1);
    }

  else
    {
      printf(" Can't open %s\n",fname_spec);
      exit(0);
    }


  /* ---------------- Memory allocation ---------------*/

  if( (  p = malloc( N_CH * length * sizeof(float) )) != NULL )
    {
      spec_array = (float *)p;

      if( (file2 = fopen( fname_spec, "r" ) ) != NULL )
        fread( (char *)spec_array, sizeof(float), (length*N_CH), file2 );
        else
          {
            printf("\n open error for %s \n", fname_spec );
            exit(0);
          }
      }

    else
      {
        printf("\nCan't allocate the memory for spec_array\n");
        exit(0);
      }

    if( (q = malloc( length * sizeof(float) )) != NULL )
      {
        array1 = (float *)q;
      }

    else
      {
        printf("\nCan't allocate the memory for array1\n");
        exit(0);
      }

    if( (r = malloc( length * sizeof(float) )) != NULL )
      {
        array2 = (float *)r;
      }

    else
      {
        printf("\nCan't allocate the memory for array2\n");
        exit(0);
      }

    if( (o = malloc( length * sizeof(float) )) != NULL )
      {
        array3 = (float *)o;
      }

    else
      {
        printf("\nCan't allocate the memory for array3\n");
        exit(0);
      }


    /* ------------------  CALCULATION  ------------------- */

    l_goldhor_input ( &R, &Re, &mua, &mub, C, T1, T2 );

    for( chno=0; chno<N_CH; chno++ )
      {
        F    = F_list[chno]/1000.0;

        if(chno>=56 && CORRECT==1)
          {
            Tlow = 0.08 ;
            K = (int)((float)chno/10. + 4. ) ;
            GSD_GAIN = GSD_GAIN / 3. ;
          }

        if((chno>=52 && chno<56) && CORRECT==1)
          {
            Tlow = 0.09 ;
            K = (int)((float)chno/10. + 3. ) ;
            GSD_GAIN = GSD_GAIN / 3. ;
          }

        if((chno>=10 && chno<52) && CORRECT==1)
          {
            K = (int)((float)chno/10.) ;
          }

        if(chno< 10 && CORRECT==1)
          {
            K = (int)((float)chno/10. + 1.) ;
            GSD_GAIN = GSD_GAIN * 2. ;
          }

        if(F_list[chno] == 0.0)
          {
            goto LOOP1;
```

```
/*          printf(" Check the Frequencies List !!! Frequency of %d ch is 0 \n", chno);
            exit(0);
*/
            }
/*
        if( chno != chno_check ) goto LOOP1;
*/
        printf("\nNow calculating for chno = %d,  F = %f  [kHz]\n", chno, F);

        Load_data( spec_array ,array1, length, N_CH, chno );
        if( SAVE_INP == 1 ) save_array( array1, length, "INPUT.DATA" );
        printf("\ninput has been loaded\n");

        if(REVERSE == 1)
          {
            reverse_array( array1, array2, length);
            move_array( array2, array1, length);
          }

        Halfwave_rectifier( array1, array2, length, A, B, &Maxh );
        if(SAVE_HLF == 1) save_array( array2, length, "HALF.DATA");
        printf("halfwave output has been calculated\n");

        if( CORRECT == 0 )
          {
            Goldhor_Model( array2, array1, length, Re, C, mua, mub );
            if(SAVE_GLD == 1) save_array( array1, length, "GLD.DATA");
            printf("Goldhor output has been calculated\n");
            goto LOOP2;
          }

        Dirac_generator( array2, array3, length, Thigh, Tlow, K, Maxh );
        if(SAVE_DIR == 1) save_array( array3, length, "DIR.DATA");
        printf("Dirac's items has been charged\n");

        for( i=0; i<length; i++ )
          array2[i] = array2[i] + array3[i];

        if(SAVE_GLDE == 1) save_array(array2, length, "GLDE.DATA");

        Goldhor_Model( array2, array1, length, Re, C, mua, mub );
        if(SAVE_GLDSIG == 1) save_array( array1, length, "GLDSIG.DATA");

        for( i=0; i<length; i++ )
          {
            array1[i] = array1[i] - 1.5 * array3[i];
            if( array1[i] < 0. ) array1[i] = 0.0;
          }

        printf("\nGoldhor output with Dirac correction has been calculated\n");
        if(SAVE_GLDD == 1) save_array( array1, length, "GLDD.DATA" );


LOOP2:

        GSD_delay = (int)(1.0/F*Fs);
        GSD_ave_length = (int)(20.0/F*2.0);

        printf("\nGSD_delay for the GSD calculation (T) = %d  [points]\n", GSD_delay);

        GSD( array1, array2, length, GSD_delay, GSD_ave_length, GSD_GAIN);
        if(SAVE_GSD == 1) save_array( array2, length, "GSD.DATA");
        printf("GSD output has been calculated \n");

        if( SAVE_OUT == 1 )
          {
            if( (file3=fopen(output_file,"a")) != NULL && (file4=fopen(frequency_file,"a")
->) != NULL )
                {
                  fprintf(file4, "%f\n", F);
                  size = fwrite( (char *)array2, sizeof(float), length, file3 );
                  if( size != length )
                    printf("\nwarning !! size '%s' values : error",output_file);
                  if( fclose(file3) == NULL && fclose(file4) == NULL )
                    printf("\nOK, array has been saved\n");
                  else printf("\ncan not close the file, the saving has failure ...\n");
                }

                else printf("\n can not open the file '%s' or '%s'",output_file,frequency_f
->ile);
          }

LOOP1:
        printf(".");

      }

    free(r);
    free(q);
    free(p);
    free(o);

    return(0);

  }
```

```
/***************************************************************

      Calculation at every step of the Hair Cell model
 Version according to the Optimun Designing Cascade/Parallel
                   Type Cochlear Filter

                     26 Oct 89
                     E.TARDIF (ATR Japan)

****************************************************************

    This program is provided to run with :

              - r_main.c ( main program )
              - r_load.c

***************************************************************/

#include <stdio.h>
#include <math.h>

# define  T   50e-6           /* sampling period  [s] */
# define  PAI 3.1415926535

/*
---------------------------------------------------------------

  Load 1 channel data from N-channel spec_array

---------------------------------------------------------------
*/

Load_data( in_array,out_array ,length,max_ch, chno)
float   *in_array;
float   *out_array;
int     length;
int     max_ch;
int     chno;
{
  register i;

  for( i=0; i < length ; i++ )
    out_array[i] =in_array[i * max_ch + chno];

  return(0);
}

/*
---------------------------------------------------------------

  Save 1 channel data to N-channel spec_array

---------------------------------------------------------------
*/

Save_data( in_array,out_array ,length,max_ch, chno)
float   *in_array;
float   *out_array;
int     length;
int     max_ch;
int     chno;
{
  register i;

  for( i=0; i < length ; i++ )
    out_array[i * max_ch + chno] = in_array[i];

  return(0);
}


/*
----------------------------------------------------

  Half Wave Rectifier

----------------------------------------------------
*/

Halfwave_rectifier( in_array, out_array, length, A, B, Maxh )
float   *in_array;
float   *out_array;
int     length;
float   A;
float   B;
float   *Maxh;
{
  int i;

  *Maxh = 0. ;

  for( i=0; i < length; i++ )
    {
      if( in_array[i] >= 0. )
        out_array[i] = 1 + A * (float)atan((double) (B * in_array[i]/32767.) ) ;

      else
        out_array[i] = exp((double)(A * B * in_array[i]/32767.));

      if( *Maxh<out_array[i] ) *Maxh = out_array[i];
    }

  return(0);
}


/*
-----------------------------------------------------------

  Dirac generator

-----------------------------------------------------------
*/

Dirac_generator( in_array, out_array, length, Thigh, Tlow, K, Maxh )
float  *in_array;
float  *out_array;
int    length;
float  Thigh;
float  Tlow;
int    K;
float  Maxh;
{
  register i;

  int      j, k, n, N, J;
  int      J1[50], J2[50];

  float    *sum;

  n = (int)(Thigh/T);
  N = (int)(Tlow /T);
  J = length/N;              /* number of Dirac items */

  for( j=1; j<=J; j++ )
    {
      J1[j] = length/J * (j-1);
      J2[j] = J1[j] + n;
      sum[j] = 0. ;

      for( k=J1[j]; k<=(J2[j]+250); k++ )
        sum[j] += in_array[k];
    }

  j = 1;
  for( i=0; i< length; i++ )
    {
      if( i>=J1[j] && i<=J2[j] && sum[j] != ((int)(n+251) * 1.) )
        {
          out_array[i] = (float)K * Maxh / (sum[j]/((float)(n+250)) ) ;
          if( i == J2[j] )
            {
              j++;
              printf("sum[%d] = %f\n", j, sum[j]);
            }
        }

      else out_array[i] = 0. ;
    }

}

/*
-----------------------------------------------------------

  Goldhor Model

-----------------------------------------------------------
```

```
   NOTATION :  R, P, C ...      composants of this model
               Re ...           R and P equivalent resistance
               mua and mub ... times constant

               T ...            slamping period
               Vc1, Vc2 ...     tensions at the capacity at instants [n-1] and [n]
               Id ...           courant thougth the diod

------------------------------------------------------------
*/

Goldhor_Model( in_array, out_array, length, Re, C, mua, mub )
float   *in_array;
float   *out_array;
int     length;
float   Re;
float   C;
float   mua;
float   mub;
{
  register  i;

  float Vc1, Vc2, Id;

  /* initialization */
  Vc1 = Vc2 = Id = 0;

  for( i=0; i<length; i++ )
    in_array[i] = in_array[i] -1.0;

  /* calculation */
  for( i=0; i<length; i++ )
    {
      /* test if the diod is on */
      if( in_array[i] > Vc2 )
        {
          /* capacity tension calculation at the instant [n] */
          Vc2 = 1/(1/T+mua+mub) * (mua*in_array[i] + Vc1/T);

          /* courant calculation at the instant [n] */
          Id  = Vc2*(1/Re+C/T) - (C/T*Vc1);

          /* ageing */
          Vc1 = Vc2;

          /* record the output in an array */
          out_array[i] = Id ;
        }

      if( in_array[i] <= Vc2 )
        {
          Vc2 = 1/(1+(mub*T)) * Vc1;
          Id  = 0. ;
          Vc1 = Vc2;

          /* record the output in an array */
          out_array[i] = Id ;
        }

    }
  return(0);
}


/*
------------------------------------------------------------------------------

   LOWPASS FILTER CALCULATION :

   INPUT  :  tab_entry[i] (can be choised)
   OUTPUT :  tab_lowpass_output[i]

------------------------------------------------------------------------------
*/


Low_pass_filter(in_array,out_array,length, TAUX )
float   *in_array;
float   *out_array;
int     length;
float   TAUX;
{
  register     i;
  First_order_LPF(in_array,out_array,length,TAUX);

  return(0);
}

/*
-----------------------------------------------------------------------------

   Automatic Gain Controler

-----------------------------------------------------------------------------
*/

AGC(in_array,out_array, length, TAGC, KAGC)
float   *in_array;
float   *out_array;
int     length;
float   TAGC;
float   KAGC;
{
  register      i;
  float         average;
  char          *p,*malloc();
  float         *temp_array;

  if( (p = malloc( length * sizeof(float) )) == NULL )
    {
      printf("\nCan't allocate the memory for temp_array in 'AGC'\n");
      exit(0);
    }
  else
    {
      temp_array = (float *)p;
    }

  First_order_LPF(in_array,temp_array,length,TAGC);
/*                save_array(temp_array,length,"AGC.LPF");
*/
  for( i=0; i<length; i++ )
    {
/*        average = in_array[i];
*/
        average = temp_array[i];

        out_array[i] = in_array[i]/( 1.0 + KAGC * average );
    }

    free(p);
    return(0);
}


/*
--------------------------------------------------------------

   GSD CALCULATION :

   INPUT  :  tab_entry[i] and tab_entry2[i]
   OUTPUT :  tab_GSD_output[i]

--------------------------------------------------------------
*/

GSD(in_array1, out_array, length, GSD_delay, GSD_ave_length, GSD_gain)
float   *in_array1;
float   *out_array;
int     length;
int     GSD_delay;
int     GSD_ave_length;
float   GSD_gain;
{
   register     i, j;
   int          maxi;
   int          size;

   float        x, y, z;
   float        maxy, maxz;

   char         *p1,*p2,*p3,*p4,*p5,*malloc();
   float        *sum,*sum_average;
   float        *dif,*dif_average;
   float        *in_array2;
```

```
    if( (p1 = malloc( (length+GSD_ave_length) * sizeof(float) )) == NULL )
      {
        printf("\nsorry, I can't allocate the memory for temp_array\n");
        exit(0);
      }
    else
      {
        sum = (float *)p1;
      }

    if( (p2 = malloc( (length+GSD_ave_length) * sizeof(float) )) == NULL )
      {
        printf("\nsorry, I can't allocate the memory for temp_array\n");
        exit(0);
      }
    else
      {
        dif = (float *)p2;
      }

    if( (p3 = malloc( (length+GSD_ave_length) * sizeof(float) )) == NULL )
      {
        printf("\nsorry, I can't allocate the memory for temp_array\n");
        exit(0);
      }
    else
      (
        sum_average = (float *)p3;
      }

    if( (p4 = malloc( (length+GSD_ave_length) * sizeof(float) )) == NULL )
      {
        printf("\nsorry, I can't allocate the memory for temp_array\n");
        exit(0);
      }
    else
      {
        dif_average = (float *)p4;
      }

    if( (p5 = malloc( (length) * sizeof(float) )) == NULL )
      {
        printf("\nsorry, I can't allocate the memory for temp_array\n");
        exit(0);
      }
    else
      {
        in_array2 = (float *)p5;
      }


     srand(100);

    for(i=0; i<length; i++)
          in_array2[i] = in_array1[i] + (float)(rand()%32767)/1e6;

    for(i=0; i<length+GSD_ave_length; i++)
    {
          sum_average[i] = 0.0;
          dif_average[i] = 0.0;
    }

    /* comparison GSD_delay is outside of the array */
    for(i=0; i<GSD_delay; i++)
      {
        sum[i] = in_array1[i];
        dif[i] = in_array2[i];
      }

    /* comparison + and - with the signal deleted */
    for(i= GSD_delay; i<length; i++)
      {
        sum[i] =  (in_array1[i] + in_array1[i-GSD_delay]);
        dif[i] =  (in_array2[i] - in_array2[i-GSD_delay]);
      }

    for(i= length; i<length+GSD_ave_length; i++)
      {
        if(i-GSD_delay < length)
        {
           sum[i] =  in_array1[i-GSD_delay];
           dif[i] = -in_array2[i-GSD_delay];
        }
        else
          {
            sum[i] = 0.0;
            dif[i] = 1e-10;
          }
      }

    /*****************************************************
      second part of the GSD calculation : integration
    *****************************************************/

     /* the constant integration is outside of the array */
     sum_average[0] = dif_average[0] = 0. ;

     for(i=0; i<GSD_ave_length; i++)
       {
         sum_average[0] += sum[i];
         dif_average[0] += dif[i];
       }

     /* instantaneous integration with GSD_ave_length  */
     for(i=1; i<length; i++)
       {
         sum_average[i] = sum_average[i-1] - sum[i-1] + sum[i+GSD_ave_length-1];
         dif_average[i] = dif_average[i-1] - (float)(fabs((double)(dif[i-1]))
                          + (float)(fabs((double)(dif[i+GSD_ave_length-1])));
       }


     /*****************************************************
       last step of the GSD : sum / difference
     *****************************************************/

     out_array[0] = 0. ;
     for(i=1; i<length; i++)
       {
         out_array[i] = (float)fabs((double)(sum_average[i] / dif_average[i]));
       }

     for(i=1; i<length; i++)
       {
         out_array[i] = (float)(2000.0/PAI)*(atan((double)(out_array[i]/GSD_gain)));
       }

     free(p5);
     free(p4);
     free(p3);
     free(p2);
     free(p1);

     return(0);
  }

 /*
 -------------------------------------------------
   lowpass filter function
 -------------------------------------------------
 */

 First_order_LPF( in_array,out_array,length,TO )
 float   *in_array;
 float   *out_array;
 int     length;
 float   TO;
 {
   register      i;

   out_array[0] = 0.0;

   for(i=1; i<length; i++)
     {
       out_array[i] = (in_array[i] + out_array[i-1]*TO/T)/(1.0 + TO/T);
     }
 }

 /*
 ------------------------
         Save Array
 ------------------------
 */

 save_array(array,length,file_name)
 float   *array;
 int     length;
```

```
char    *file_name;
{
  FILE  *file1;
  int   size;

  file1 = fopen(file_name,"w");
  size  = fwrite(array, sizeof(float), length, file1 );
  fclose(file1);

  return(0);
}


/*
------------------------
  Move Array
------------------------
*/

move_array(in_array,out_array,length)
float   *in_array;
float   *out_array;
int     length;
{
  register i;

  for( i=0; i < length ; i++ )
    out_array[i] = in_array[i];

  return(0);
}


/*
------------------------
  Reverse Array
------------------------
*/

reverse_array(in_array,out_array,length)
float   *in_array;
float   *out_array;
int     length;
{
  register i;

  for( i=0; i < length ; i++ )
    out_array[i] = in_array[length - i - 1];

  return(0);
}
```

```
/********************************************************

               hair cell model parameters
        version according to the Optimun designing
           Cascade/parallel type Cochlear Filter

                   25 Oct 89
                   E.TARDIF (ATR Japan)

********************************************************

    This subroutine is provided to run with :

         - hair6.c
         - calcul_hair6.c

*********************************************************/

#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>


/* --- Goldhor's model parameters calculation --- */

float l_goldhor_input( R, Re, mua, mub, C, T1, T2 )
float *R, *Re, *mua, *mub,
      C, T1, T2;
{
  /* calculation of the time constants */
  *mub = 1/ T2;
  *mua = 1/ T1 - *mub;

  *Re  = T2 / C;
  *R   = *Re /( (*Re * C)/ T1 - 1);

  return(0);
}



/*
----------------------------------
        get_file_length
----------------------------------
 */

get_file_length ( file, size, length )
char *file;
int  size;
int  *length;

{
  struct stat buf;

  if( ( stat( file, &buf ) ) == 0 )
   {
    *length = buf.st_size / size;
    return(0);
   }
  else
   {
    printf("Something strange in 'get_file_length'\n");
    exit(0);
   }

}
```

HAIRCELL.PARA   printed on 02/02/1990 at 12:30:32 from hm03

```
HALF_WAVE     A     : 10.0
HALF_WAVE     B     :  3.0
Goldhor       T1    : 0.015
Goldhor       T2    : 0.120
Goldhor       C     : 0.001
Dirac_gene   Tlow   : 0.1
Dirac_gene   Thigh  : 0.0025
LPF          TAUX1  : 0.0003
AGC          TAGC   : 0.0003
AGC          KAGC   : 1000.0
GSD          SLOPE  : 10.0
chno_check   chno   : 28
```

```
/* ---------------- NEW GRAPHIC SUBROUTINES LIBRARIES ----------------- */
/*                                   S.Tenpaku( JAPAN )                 */
/*                                   11.Aug.1989                        */
/*                    subroutines add by E.TARDIF( JAPAN )              */
/*                                   31.Aug.1989                        */
# include       <stdio.h>

# define        TRUE            1
# define        FALSE           0
# define        HOME_VIEW       2

#define RED      2
#define ORANGE   3
#define YELLOW   4
#define GREEN    5
#define BLUE     6
#define WHITE    7

/*
 *
----------------------------
        g_init
----------------------------
 */
void
g_init()
   {    mgiasngp( 0, 0 );       /* assign graphics */
   }

/*
 *
----------------------------
        g_end
----------------------------
 */
void
g_end()
   {
        mgisyncrb( -1 );        /* wait */
        mgideagp();             /* deassign graphics and end    */
   }

/*
 *
----------------------------
        g_plane_clear
----------------------------
 */
void
g_plane_clear( plane )
int     plane;
   {    mgiclearpln( plane, -1, 0 );
   }

/*
 *
------------------------------
        g_pen_style
------------------------------
 */
void
g_pen_style( thick, term, mode, mask )
int     thick;
int     term;
int     mode;
int     mask;
   {    mgiwidth( thick, term );
        mgidash( mode, mask );
   }

/*
 *
------------------------------
        g_color_set
------------------------------
 */
void
g_color_set( index, color )
int  index;     /* number of the color map register     */
char *color;    /* CNS mean string */
   {    mgicm( index, mgfcns( color ) );
   }
```

```
/*
 *
------------------------------
        g_color_select
------------------------------
 */
void
g_color_select( index )
int     index;
   {    mgihue( index );
   }

/*
 *
------------------------------
        g_mgrp
------------------------------
 */
void
g_mgrp()
{
  mgrp( 0.0,0.0 );
}

/*
 *
------------------------------
        g_rline
------------------------------
 */
void
g_rline( x0, y0, x1, y1 )
float   x0, y0;
float   x1, y1;
   {    mgrtl( x0, y0, x1, y1 );
   }

/*
 *
------------------------------
        g_iline
------------------------------
 */
void
g_iline( x0, y0, x1, y1 )
int     x0, y0;
int     x1, y1;
   {    mgitl( x0, y0, x1, y1 );
   }

/*
 *
------------------------------
        g_rlines
------------------------------
 */
void
g_rlines( point, n )
float   *point;
int     n;
   {
register int    i;
        mgrp( 0.0, point[0] );
        for ( i = 0 ; i < n ; i++ )
                mgrl1( (float)i, point[i] );
/*      for ( i = 0 ; i < n-1 ; i++ )
                mgrtl( (float)i, point[i], (float)(i+1), point[i+1] );*/
    }

/*
 *
------------------------------
        g_ilines
------------------------------
 */
void
g_ilines( point, n )
int     *point;
int     n;
   {
register int    i;
        for ( i = 0 ; i < n-1 ; i++ )
                mgitl( i, point[i], (i+1), point[i+1] );
    }

 /*
  *
 ------------------------------
        g_view_define
 ------------------------------
  */
 void
 g_view_define( view, xl, yb, xr, yt, Rxl, Ryb, Rxr, Ryt )
 int     view;
 int      xl,  yb,  xr,  yt;
 float   Rxl, Ryb, Rxr, Ryt;
    {
        mgidefw( view );
        mgipw( view, HOME_VIEW, xl, yb, xr, yt );
        mgrvcoor( view, Rxl, Ryb, Rxr, Ryt );
    }

 /*
  *
 ------------------------------
        g_view_select
 ------------------------------
  */
 void
 g_view_select( view )
 int     view;
    {   mgiv( view );
    }

 /*
  *
 ------------------------------
        g_view_end
 ------------------------------
  */
 void
 g_view_end()
    {   mgiv( HOME_VIEW );
    }

 /*
  *
 ------------------------------
        g_view_delete
 ------------------------------
  */
 void
 g_view_delete( view )
 int     view;
    {   mgidelw( view );
    }

 /*
  *
 ------------------------------
        g_axis_draw
 ------------------------------
  */
 void
 g_axis_draw( grid, x_min, x_max, x_step, y_min, y_max, y_step )
 int     grid;
 float   x_min, x_max, x_step, y_min, y_max, y_step;
   {
        float   xp, yp;
        double  xd, yd;
        float   xa, ya;
        int     thick   = 1;
        int     term    = 0;
        int     mode    = 0;
        int     mask    = 0x0008;

        xd = x_step;
        yd = y_step;
        xa = ( x_max - x_min ) / 100.0;
        ya = ( y_max - y_min ) / 100.0;
        if ( xa < ya )
                xa = ya;
        else
                ya = xa;
        xp = x_min + xd;
        yp = y_min + yd;
        if ( grid < 0 );
        else
        if ( grid > 0 )
           {    g_pen_style( thick, term, mode, mask );
                while( xp < x_max )
                   {    mgrl( xp, y_min, xp, y_max );
                        xp += xd;
                   }
                while( yp < y_max )
                   {    mgrl( x_min, yp, x_max, yp );
                        yp += yd;
                   }
           }
        else
           {    while( xp < x_max )
                   {    mgrl( xp, y_min, xp, y_min+ya );
                        mgrl( xp, y_max, xp, y_max-ya );
                        xp += xd;
                   }
                while( yp < y_max )
                   {    mgrl( x_min, yp, x_min+xa, yp );
                        mgrl( x_max, yp, x_max-xa, yp );
                        yp += yd;
                   }
           }
        /* draw the box */
        g_pen_style( 2, term, mode, 0xff );
        mgrtl( x_min, y_min, x_min, y_max );
        mgrtl( x_max, y_min, x_max, y_max );
        mgrtl( x_min, y_min, x_max, y_min );
        mgrtl( x_min, y_max, x_max, y_max );
  }

/*
 *
------------------------------
        g_font_set
------------------------------
 */
void
g_font_set( index, font )
int     index;
char    *font;
   {    mgifetchgf( index, font );
   }

/*
 *
------------------------------
        g_font_select
------------------------------
 */
void
g_font_select( font )
int     font;
   {    mgigf( font );
   }

/*
 *
------------------------------
        g_print
------------------------------
 */
void
g_print( x, y, string )
int     x, y;
char    *string;
   {    mgigfs( x, y, 0, string );
   }

/*
 *
------------------------------
        g_printf
------------------------------
 */
void
g_printf( x, y, format, arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9, arg10 )
int     x, y;
char    *format;
  {     char    buffer[80];
        sprintf( buffer, format, arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9, a
rg10 );
```

```
        mgigfs( x, y, 0, buffer );
  }

/*
 *
------------------------------
        g_text_affich
------------------------------
 */
void
g_text_affich( view,font,colour,x,y,string )
int  x,y,view,font,colour;
char *string;
{
  g_view_select( view );
  g_font_select( font );
  g_color_select( colour );
  g_print( x,y,string );
  g_view_end();
}

/*
 *
------------------------------
        g_window_set
------------------------------
 */
void
g_window_set(n, x1, y1, x2, y2, xn1, yn1, xn2, yn2)
int   n,x1,y1,x2,y2;
float xn1,yn1,xn2,yn2;
  {
    g_view_end();
    g_view_define(n, x1, y1, x2, y2, xn1, yn1, xn2, yn2);
    g_color_select(7);
    g_iline(x1, y1, x1, y2);
    g_iline(x1, y2, x2, y2);
    g_iline(x2, y2, x2, y1);
    g_iline(x2, y1, x1, y1);
  }

/*
 *
------------------------------
           g_mgrlsc
------------------------------
 */
void
g_mgrlsc( n,x,y,crtl )
int    n, crtl;
float  x,y;
{
  mgrlsc( n, x, y, crtl );
}


/*
 *
-------------------------------
             g_mgrc
-------------------------------
 */
void
g_mgrc( x,y,radius )
float  x,y,radius;
{
  mgrc( x,y,radius );
}


/*
-----------------------------------
          g_graduat()
-----------------------------------
 */
void g_graduat( MAX ) /* axis graduations */
float MAX;
{
  float h;
  for (h=0; h<=MAX; h+=5)
    {
      g_rline(h, -0.05, h, 0.05);
    }
}
```

```
/*
 *
--------------------------------
--------------------------------
 */
/*void*/

/* end-of-file ( graphic.c ) */
```