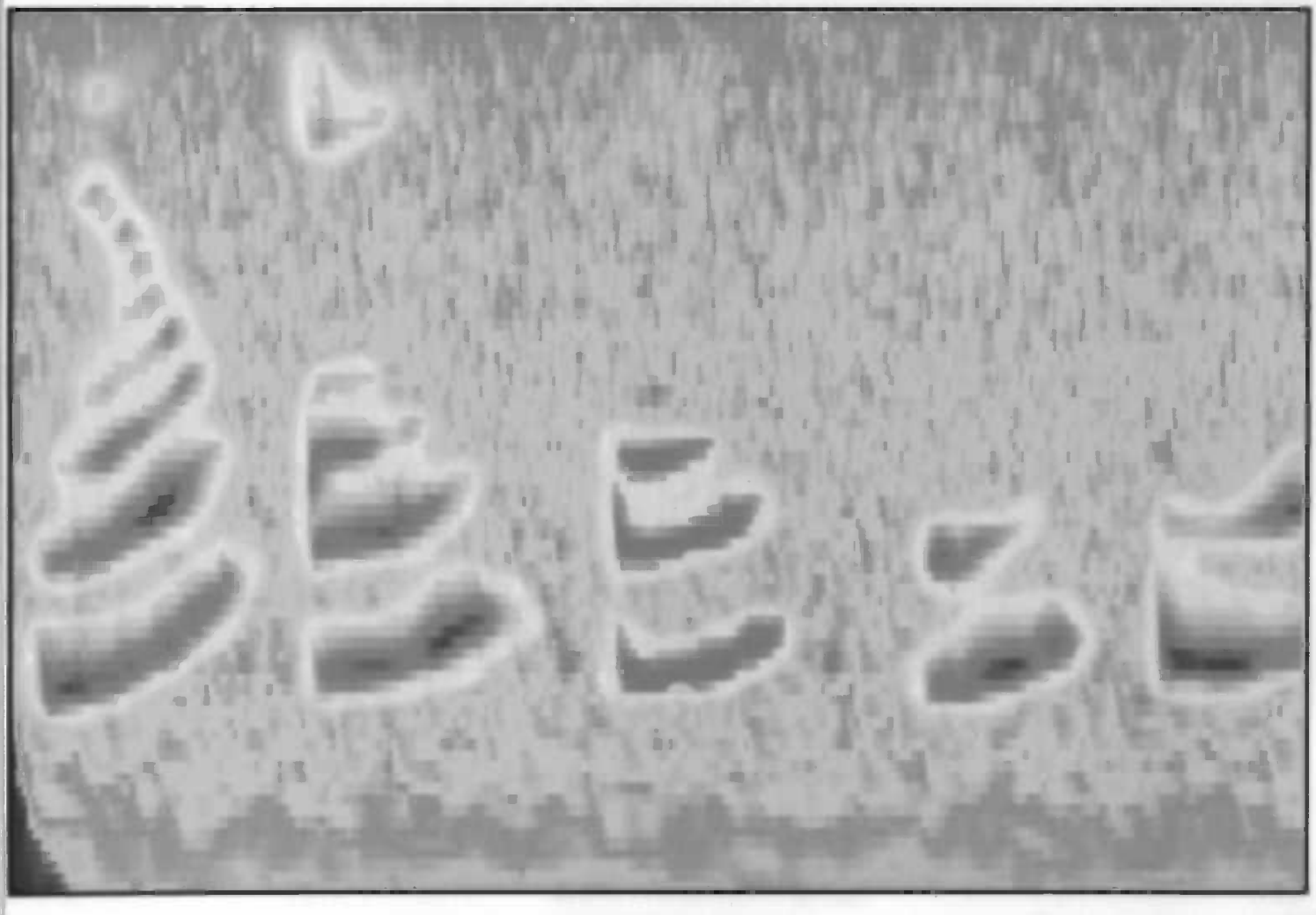
# Classification of signal components in cochleograms



**doctoraalscriptie**



**Pim van Oerle**

**November 2006**

# Classification of signal components in cochleograms

-Scriptie -

(Taal en Spraak)

Pim van Oerle
s1058797

21st November 2006

Interne begeleider: Dr. Tjeerd Andringa
Interne begeleider: Ir. Dirkjan Krijnders
Externe begeleider: Dr. Ir. Peter van Hengel
Stagebedrijf: Sound Intelligence

**Kunstmatige Intelligentie**
**Rijksuniversiteit Groningen**

**Abstract**

Distinguishing between the sounds in our environment is quite trivial for humans. We can easily distinguish between sounds that originated from different sources, and routinely isolate single sources. Since the '50s, the perceptual segregation of sound into components has been object of study, but a definitive method to accomplish this, seemingly so very simple, feat has not been found.

To facilitate the analysis of sound - to accomplish, for example, speech recognition or instrument separation - some form of base separation will have to be performed. The basic components of sound can be broadly separated into three categories: pulses, sines, and noise-like components. Detecting these base components of sound would allow for later application of algorithms to combine them into greater, more complicated components defining the structure of sound.

A method to accomplish the detection of such base components was investigated, using Sound Intelligence's Cochlear Analysis system and existing algorithms written by Tjeerd Andringa and Dirkjan Krijnders. A simple system was made which tries to detect components in cochleograms, and resynthesises a signal based on the detected components. An analysis of the performance of the system was made by assessing the results of the resynthesis of several testsignals.

# Contents

# 1 Introduction

## 1.1 Introduction

Distinguishing between the sounds in our environment is quite trivial for humans. We can easily distinguish between sounds that originated from different sources, and routinely isolate single sources - Colin Cherry noted the ability of human listeners to attend to a single speaker in a mixture of many voices in 1953, and called this the "cocktail party problem" (Brown94). Since Cherry's observation, the perceptual segregation of sound into components has been object of study, but a definitive method to accomplish this, seemingly so very simple, feat has not been found.

To facilitate the analysis of sound - to accomplish, for example, speech recognition or instrument separation - some form of base separation will have to be performed. The basic components of sound can be broadly separated into three categories: pulses, which are compact in time and broad in frequency-range, sines, which are well defined in frequency and more continuous in time, and lastly noise-like components, which lie in between sines and pulses, both in a time- and in a frequency-sense. Detecting these base components of sound would allow for later application of algorithms to combine them into greater, more complicated components of sound.

This thesis will investigate a method for the detection of such base components as described above. For simplicity's sake, the 'noisier' components are neglected in the remainder of this thesis, and focus will lie on the detection of relatively well defined pulses and sines.

### 1.1.1 An earlier incarnation

At the start of the research for this thesis, the initial goal was to develop a system, based on the existing model of the cochlea and the energy matrix it produces, which would differentiate between pulses, noise, and sines. In the course of research, the first incarnation of the system, which will be described in detail later, was found to be too much work to build correctly. It was based on broadly the same assumptions and algorithms as the current system, but turned out to be too simplified (or just plainly wrong at some points) to be useful.

As it turned out, tackling all the problems that would arise - developing the algorithms, adding a layer to the cochlear analysis system, - would take too much time for a master's thesis, so the decision was made to lighten the workload by focusing on implementing and testing.

### 1.1.2 Shifting focus

For simplicity's sake, the actual development of the system was to be done in Matlab - using c or C++ would hinder rapid development of differing approaches (although execution the algorithms themselves would be more efficient). The cochlear model (and assorted software to process the data) was written by Sound Intelligence in C++. This presented a bit of problem - communication between the C++- and Matlab-parts had to be established.

As a trade for the time Sound Intelligence invested in adapting the new version of their cochlear analysis system in such a way that good matlab-interaction could be grafted onto it, I did an internship, developing a Graphical User Interface for their SIgard Sound Detection Platform.

The remainder of this thesis will consider the creation and testing of a framework to use Tjeerd Andringa's peak- and basic component-finding code and DirkJan Krijnders's Matlab-Cochleamodel link (the Academic Worker), to distinguish pulses and sines in sound. If my description of the reasons for - and exact workings of - some of the algorithms seems somewhat lacking, it probably stems from my not creating them and therefore not grasping all their finer points wholly.

## 1.2 Research Question

The research question we will focus on is twofold: How to detect sines and pulses in sound, and whether the system we develop on these grounds has acceptable results.

## 1.3 scientific / social relevance

Feature recognition in sound is often based on lower-level, signal-analysis based, concepts. A workable dissection of sound into base components could facilitate description in terms of higher level concepts.

# 2 (Theoretical) Background

The following chapter contains an analysis of prior work done regarding detection of pulses and sinusoids in audio. Research into this specific question often stems from a more generalized version of the same question - how to detect certain elements in a stream of sound. For this work, we need to find theories and models that try to (or offer a framework wherein to) find pulses and sinusoids in audio. The framework we would like to choose should be extensible to include noisy components, and fairly modular to allow for updating or extension of the various steps that will be taken within the model. Our preferred system should be robust, always work, be adaptable for new contexts, and not make any assumptions about it's input.

A fitting place to start searching is with Bregman's 1990 book 'Auditory Scene Analysis' (Bregman90). Based on carefully executed psychoacoustic experiments, Bregman constructs a theory which tries to explain the organisation and processing of sound. The actual perceptual process can be described with a two-stage model, according to Bregman. At the first stage, acoustic input is decomposed into *segments*, cued by harmonicity and onset synchronisation (along with modulation and spatial location). These segments are then fed into the second stage, where they will be segregated (and combined) by source into *streams*. Differentiation between sources is achieved by looking at attributes like amplitude, pitch or timbre. Further, higher level, processing will make use of these streams as a basis. The general idea of separation of segments and combination into streams seems a useful one, but a strict two-stage model such as Bregman proposes strikes as overly simplified, especially when moving from the testing ground of clean sinusoids and controlled bursts of white noise to the real world with it's broad range of noisy components.

Before Bregman published his book, early work in the field was being done. Weintraub, for example, proposes the first computational model of auditory organisation in 1985 (Weintraub85), influenced by the psychoacoustic research of Bregman and others. With his model, Weintraub attempted to separate and reconstruct two simultaneous speakers, with differing average pitches, from one signal. When the first system he developed didn't perform to satisfaction, Weintraub moved on to his 'current' system, which is state-based and uses Markov models to model the two speakers. The states of the models determine the subsequent processing steps - hypotheses about the two speakers and the signal are generated. Because sound processing works with causal events, the generation and use of hypotheses seems the most important piece of information that can

be used from Weintraub's work.

Following Weintraub's efforts, many more attempts at speaker separation have been made. Independent component analysis (ICA) is a statistical and computational technique for revealing hidden factors that underlie sets of random variables, measurements, or signals (Hyvarinen01). The method can be used to separate any mixture of signals, including performing Blind Source Separation, blindly separating mixed signals from different sources. After separation, further processing can be performed on the separated sources. Taking this 'extra step' before a form of segmentation could benefit and simplify source-separation, although assumptions have to be made (the statistics of the signal are invariant, and multiple channels are actually present).

The most direct, complete and ambitious implementations expanding on Bregman's idea's are found in work by Brown (Brown92), Cooke (Cooke91) and Mellinger(Mellinger91). These systems have in common that no strong assumptions are made about the number or the type of sources that are present in the signal. Furthermore, time and frequency are treated as equally important (unlike conventional speech processing, where focus lies on short-term spectral estimates (e.g. Parsons76)). The flow of data is strictly unidirectional, going from concrete to abstract, as can be seen in figure 1. We could call this type of system *Data-driven*, after Ellis (Ellis96). On a conceptual level, all the systems try to duplicate the organisation of the human auditory system: There is a perifery (including a cochlear model), a basic representation of low-level properties, and segregation into streams or groups. Brown and Cooke focus on separating speech from noise, Mellinger set out to separate the participating instruments in ensemble music. The proposed processing stages seem usable, but the strictly bottom-up system seems non-robust, and unforgiving with respect to ambiguities. Furthermore, the use of FFT's requires the implicit assumption that the signal is periodical, which real sound is not, and therefore introduces distortions into the sound, making it difficult to correctly track sources.

A less rigid system than the previously encountered data-driven ones was proposed by Ellis (Ellis96). To contrast his system with the unidirectional systems of Brown and Cooke, he calls his approach *prediction driven* - a model of the world is created, with which predictions can be generated and checked (by looking at energy and periodicity, for example). Ellis differentiates between three classes of elements: noises, tonal elements and transients (or 'clicks'). Prediction driven processing allows for performing inference on incomplete data, handling ambiguity by allowing multiple hypotheses at once, and revision of hypotheses by backtracking. Incor-

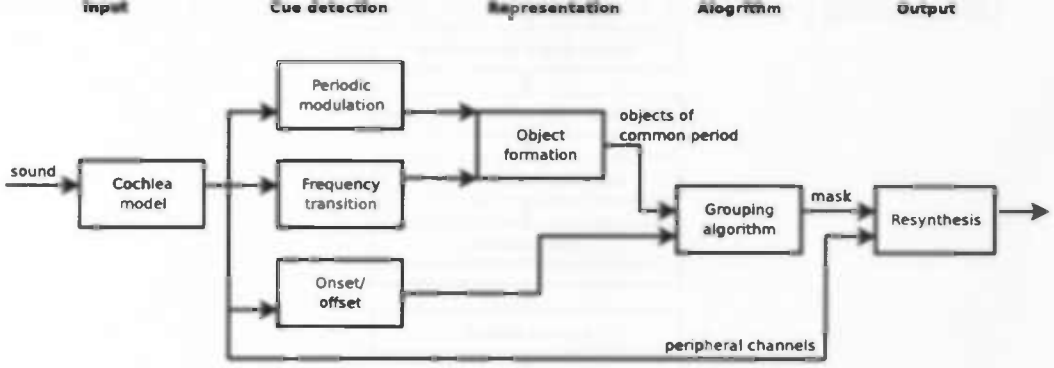



Figure 1: Diagram of a typical Brown-style (data-driven) sound analysis system (based on Ellis(96) and on Brown(92)). The system consists of three distinct processing stages, and input and output stages. Flow is strictly from left to right, from concrete processing to more abstract 'reasoning'.

porating elements from this approach would be useful for us - the use of predictions, or hypotheses was mentioned before. The shift from a strict per-frame approach, in which only the current frame is of importance, to a more flexible view of time is something that should also be incorporated - looking back at the past is important (including adapting future processing to current hypotheses).

On a more fundamental level, Griffiths (Griffiths04), describes what he calls Auditory Objects, the fundamental elements of the auditory world. He proposes a conceptual framework for analysis, and indicates some basic properties - such as their separation by perceptual boundaries and their relation to sound sources - that auditory objects should posses to be able to be marked as such. From the conceptual framework, a more general framework for models of auditory-object analysis (see figure 2) can be derived. While this model is overly general for the task at hand, it could be adapted for our use. An interesting idea which Griffiths touches on is the 'reordering' of the flow of processing, depending on the task at hand. This means stages that are not useful to the current hypotheses can be skipped, or earlier stages can be reused when needed. Combined with the use of hypotheses, this could make for efficient processing.

Deviating a bit from the auditory field, and remembering that Auditory Scene Analysis was inspired on the field of Machine Vision (on Scene Analysis), we look at Marr (Marr82), who stresses the importance of multiple levels of analysis. He distinguished three levels for information-processing tasks: an implementation layer, an algorithmic layer, and a layer consisting of computational theory. We should probably use these

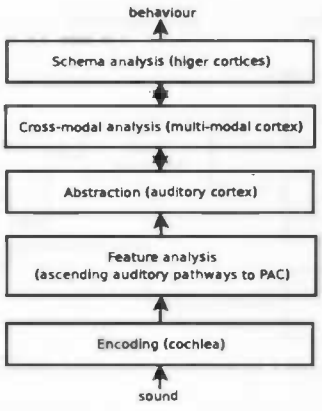



Figure 2: Griffiths' framework for models of auditory object analysis (adapted from Griffiths(04)). Mentioned are the main operational processing stages, and their plausible neurological substrates. Griffiths notes that the precise order of states is probably dependent on task demands.

levels of description to describe our systems and solutions, though that seems the logical thing to do anyway. Machine Vision has developed many techniques pertaining to the task at hand, although not very many seem directly applicable without any modification. The ideas from Machine Vision (and signal processing in general) are, of course, invaluable for the type of work we are attempting.

Conceiving of a model or system to use is one part of the task at hand, but the importance of assessing it's performance should not be taken lightly. assessment is actually quite difficult, because the models are trying to duplicate the operation of an internal perceptual process, while we cannot directly access that process' output to compare it with our systems (Ellis96). If we could create a perfect measure to compare our model with, we would use that measure, and forego all the trouble that comes with making a new model. Assessment could be done by mixing two signals, and seeing how well those can be reproduced. This creates another problem: how to measure resemblance between the originals and the reproductions. A good option would be subjective listening tasks by human observers to obtain measure of resemblance between original and resynthesis, but that takes too much resources for our limited scope. Using the techniques we currently are, comparing source and resynthesis can be done by calculating an inproduct between the two (Andringa02). For the assessment of natural signals, comparing a masked source and masked resynthesis prevents distortion by irreproducible - and perhaps, irrelevant - background noise, although this could easily remove relevant information by accident

and calls for a tricky question: how to acquire and assess good masks?

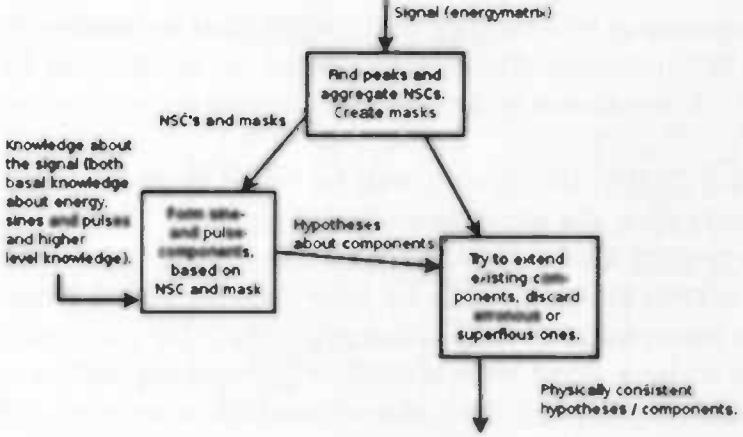



Figure 3: Conceptual representation of the system we will use. Shown is one processing 'step', which has a typical length of 500 ms in our application.

# 3 Algorithms

## 3.1 general principles

In the previous chapter, we searched for a set of ideas and concepts to adapt for our goals. In the following chapter we shall construct a system and set of algorithms that will allow us to detect pulses and sinusoids in sound. We start by deriving a set of general principles the system we will develop should adhere to.

The first principle the system should be based on is the principle of robustness: the system should always work - when the human system works as well. It would not be prudent to require the system to function where a human could not. How could we check the results?. For our system to 'always works', like it's human counterpart, it can not rely on a preference of optimal conditions; using it should not require the user to adapt, rather, the system should adapt to the user and his current environment.

Besides being robust, the system should also take heed of the current context it is being used in. Therefore, the second principle will be that the system has to be context-based. While a strict bottom-up process, like Brown and Cooke proposed, could be made to perform well in certain situations, for a system that is to be used in the real world, uncertain situations cannot be constricted to a few predefined uses. Top-down processing has to be used to influence the processes the system uses, and to adapt to the current context. Adhering to this principle, a system could be created

in which processing will be inhibited unless cues indicating that processing should occur present themselves. When the system has to be used in a new context, top-down processes can shift focus, achieving a modular design.

Third and finally, the system will be based on the principle that uncertainty pertaining the input necessitates working with hypotheses. Hypotheses represent knowledge about the world, and (again) the context in which the system is being used. As Ellis (Ellis96) states, working *without* hypotheses raises quite a few problems, of which the problem of obscured data seems to be a good representative [1]. Working with a system that generates and tries to prove hypotheses, enables us to solve difficult crossings and circumnavigate missing pieces of information. It also opens up the way to backtracing in the signal - for example, to find a sine's phase-information, only when needed to solve an uncertainty.

Keeping in mind these three principles and looking back at the theories explored in the last chapter, a conceptual description of the system we will use can be constructed. To facilitate eventual real-time use of the system, and to give the system a steady base of input, all the signal fed to the system will be divided into slices (consisting of a hundred frames, or with all settings standard, 500 ms). By looking at certain aspects (energy present at a certain time and frequency) of a slice of the input, hypotheses about the components present in those frames can be formed. When progressing in time to the next slice of input, hypotheses that were formed earlier can be checked. If a hypothesis about a component turns out to be correct (for instance, a pulse was found in the last slice, and no new information debunked the presence of that pulse), it can be assumed to be correct. Fig 3 illustrates this conceptual overview of the system.

We will use Tjeerd Andringa's CPSP system (Andringa02) for the actual processing of the input, which will give us a logarithmically scaled energy matrix to work with. At the base of the system lies a model of the cochlea, plus a number of signalprocessing steps, provided by Sound Intelligence. Building up from that basis, the processes of the system can be subdivided into three main stages: Preselection, where Narrow Signal Components (NSC) are formed from peaks in the signal, Combination into components, where the newformed NSCs are aggregated into full-fledged components (either sine or pulse), followed by the stage of Higher Level processing where the processes go beyond simple selection reside.

[1] Warren(Warren70) conducted an experiment where a syllable in speech was obscured by a cough-like burst. Subjects could infer what was being said at the time of the burst, but a more interesting result was that subjects had great difficulty pinpointing the exact occurrence of the burst.

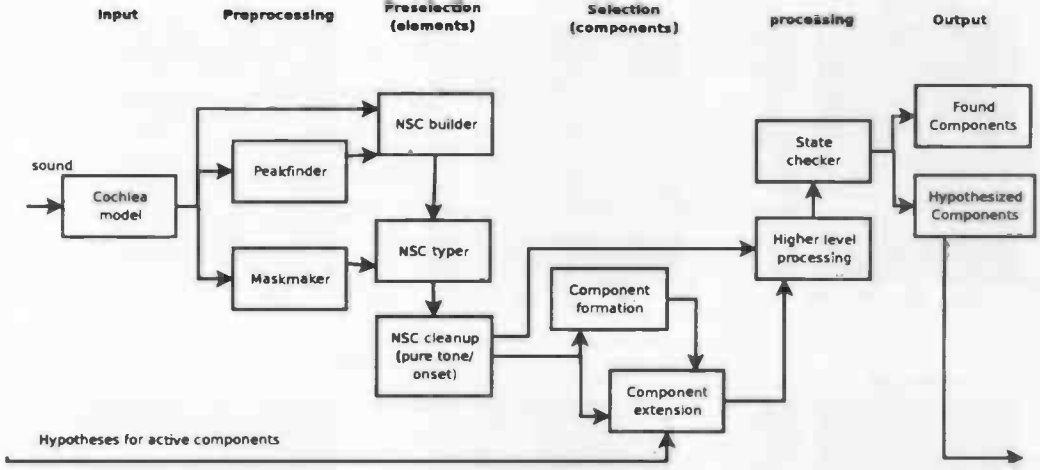



Figure 4: Diagram of the system we will use. Shown is one processing 'step', which has a typical length of 500 ms in our application.

Context-specific processes are found in this stage, as well as processes that 'look back' at the energy-matrix, to see what potential components have fallen outside the mask. Standing somewhat apart from the last three stages is resynthesis, the rebuilding of the signal based on the components that were found. This stage's contribution is two-part: it is used for assessment of the performance of the system as a whole, and it demonstrates the system in use. Figure 4 gives a schematic overview of the stages, their relations and the processing-elements they contain. The following sections delve deeper into the stages and the processes they encompass.

## 3.2 preselection (ridges to NSC's, typing)

Ideally, preselection should take in the signal and yield only the relevant, interesting bits. Practically,we don't know beforehand what is relevant and what not, so we have to make do with a less precise sifting of the signal. Our preselection process will find the peaks[2] in the signal, and then try to combine those peaks into continuous groups, called Narrow Signal Components, as mentioned before. The peakfinding algorithm works as a primitive form of template matching, testing if a pulse or tone would fit in one dimension (it's width) like a matched filter.

At this stage of the system the focus lies not with pruning the available data, but on acquiring as many candidates for component-hood as possible. Everything that has a high signal to noise ratio is interesting, and can

[2]Peaks are segments that have higher energy then their neighbors

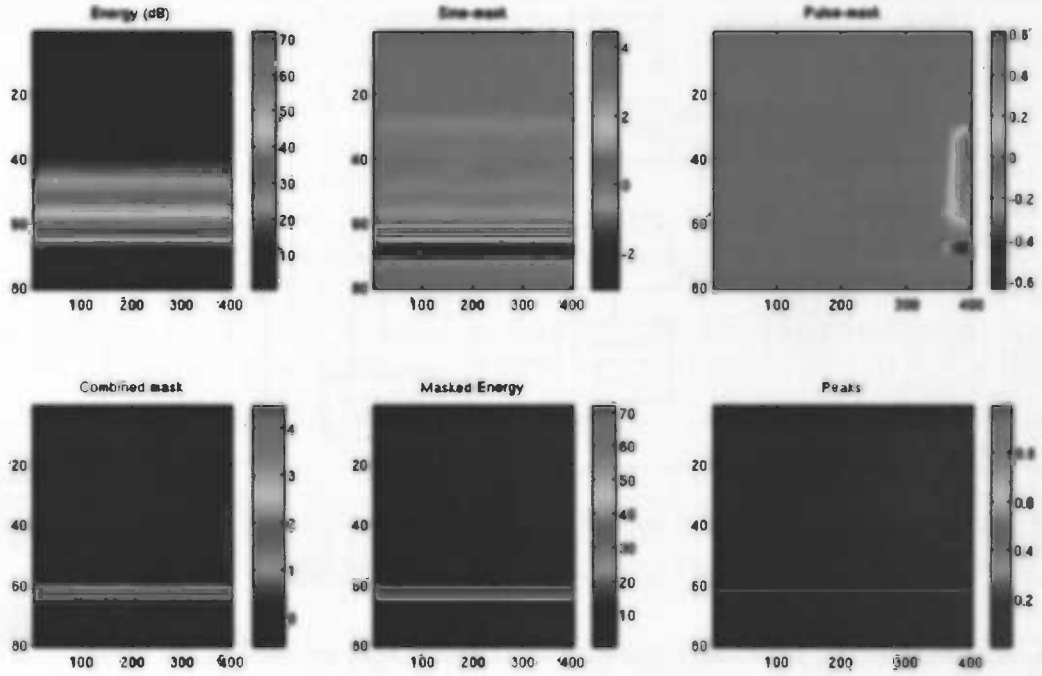



Figure 5: An example of the peak-finding and masking processes. The steps go from the basic energy-matrix, through the forming of the masks, to masking the energy-matrix and finding peaks.

easily be pruned later on. We will, however, make assumptions about the most appropriate type, to ease later processing, but no elements found in this stage will be discarded out of hand. The stage will consist of three modules: the module that finds peaks, the module that combines peaks into NSCs, and the module that assigns tentative types to NSCs.

### 3.2.1 Peaks

Finding the peaks in the signal is essentially done by scanning through it and noting the segments with the highest local energy (and high SNR - quiet components in silence are clearly present, 'high energy' is quite relative). This creates a matrix the size of the energymatrix that was used as input, but with all the points that might be of interest marked as such. To actually build the peak-matrix, we will build a mask along two dimensions: in segment(frequency)-direction and in frame(time)-direction. The resulting pulse- and sine-masks are then used to create a common mask indicating all 'active areas' in the slice of signal being processed. To obtain the final peak-matrix, we first threshold this general mask, and apply it to

the energy-matrix. This masked energy-matrix is then fed to a peakfinding algorithm. Figure 5 gives an overview, and the steps are described in more detail in the following paragraphs.

First, as a preprocessing step before peak-finding can take place, a pulse-mask and a sine-mask are created by correcting the energy-matrix for the broadness of a sine or pulse, which was run through the cochlea at an initialization stage. This gives a measure of the 'fit' of a pulse- or sine-template on the energy-matrix.

**Algorithm 1** - Creating the sine- and pulsemasks

```
(1)   Make the sine-mask: loop over all segments,
      except the first and last, per segment do:
(1.1)    Per segment, return the EdB-value,
         corrected for (calculated earlier) sine-
         broadness of the cochlea at the present
         segment, and divided by the standarddev-
         iation for whitenoise at that segment
         for this cochlea
(2)   Make the pulse mask: loop over all segments,
      except the first and the last
(2.1)    Per segment, return the EdB-value, corrected
         for (calculated earlier) pulsebroadness of the
         cochlea at this segment, and divided by the
         standarddeviation for whitenoise at the
         current segment for this cochlea
```

We then combine these two masks by taking the energy-value of the mask which has the biggest value, do that for each point on the mask, apply that mask to the energy-matrix, and feed that to the peakfinder-algorithm. This algorithm walks over all frames, and marks the segments that are higher in energy then their two immediate neighbors. This gives the local maxima in energy on that frame, and gives us the peaks we need.

### 3.2.2 combining peaks into NSC's

While peaks show us the local spots of interest, we need to make some form of structures, tracing paths of interest through both time and frequency - after all, a sine is extended in time, while a pulse is extended through multiple frequencies. In this module elligeble peaks will be combined into Narrow Signal Components when they can be traced into ridges,

**Algorithm 2 - creating the peak matrix**

```
(1)   Create an empty copy of the masked energy-matrix
      (call it the peak-matrix)
(2)   Loop over all frames in the masked energy-matrix,
      for every frame do:
(2.1)    Find the segments that are higher in energy
         then their immediate neighbours, so as to get
         a local maximum.
(2.2)    Mark the found segments in the peak-matrix
(3)   Return the peak-matrix
```

effectively 'binding' them together. The procedure for this is, simply put, to find startpoints and try to extend them as much as possible, either in time or frequency. When extension is no longer possible, a NSC has been formed.

### 3.2.3 assigning types to NSC's (masking / thresholding)

As the two basic types of components we wish to find are pulses and sines, a tentative assignment of these types can be made at this stage, based on thresholded versions of the sine- and pulse-masks that were constructed earlier.

In the current version of the system, the lower limit for the fit with the sine mask is set to 40 %, and the limit for fit with the pulse mask is set to 75%. The limit for the sine mask is set quite a bit lower, because sine-like components that are partly covered by the the sine mask are more often sines than not. Choosing a high limit would falsely discard more correct sines then it would prevent inappropriate typing of not-sines as sines.

## 3.3 Selection of components

After extracting the NSCs from the signal, we are left with a set of possible pulses and sines. Not all NSCs are actual components. Some are pieces of noisy background that resemble a sine-like component by chance. Some are part of a larger component, and some are just artefacts, introduced by our desire to find as much components as possible to reduce the chance of missing valid components. Figure 6 shows the multitude of NSCs found in a typical signal. Some methods must be used to find out which NSC can actually be marked as components (be they sines or pulses), and which

**Algorithm 3 - combining peaks into NSC's**

```
(1)   Loop over all frames, for every frame do:
(1.1)    If there are active segments, sort them
         in descending order
(1.2)    For every active segment, do:
(1.2.1)     Try to extend the NSC at this segment
            (ExtendNSC is described in the appendices)
(1.2.2)     Add the extended segments to the list of
            active segments
(1.3)    Find connected vertical structures
(1.4)    Loop over the found structures:
(1.4.1)     Use the start-segment of every found verti-
            cally connected structure as a startingpoint
(1.4.2)     Check if the segment isn't already being used
(1.4.2.1)      If not, start a new NSC and add the segment
               to the list of actives (StartNSC is de-
               scribed in the appendices)
(2)    Discard the NSC's that are trivially small
(3)    Set the type of all NSCs to 0 - undefined.
```

**Algorithm 4 - assigning a type to the NSC's**

```
(1)   Take the sine-mask formed earlier, choose a limit
      and threshold to form a binary mask indicating where
      'sine'-areas are.
(2)   Check for sines: loop over all NSCs
(2.1)    Check if the NSC is untyped
(2.1.1)     If so, check how well the element fits in
            the sine mask
(2.1.1.1)      If the fit is good enough, mark as a sine
(3)   Take the pulse-mask formed earlier, choose a limit
      and threshold to form a binary mask indicating where
      'pulse'-areas are.
(4)   Check for pulses: loop over all NSCs
(4.1)    Check if the NSC is untyped
(4.1.1)     If so, check how well the element fits in
            the binary pulse mask
(4.1.1.1)      If the fit is good enough, mark as pulse.
```

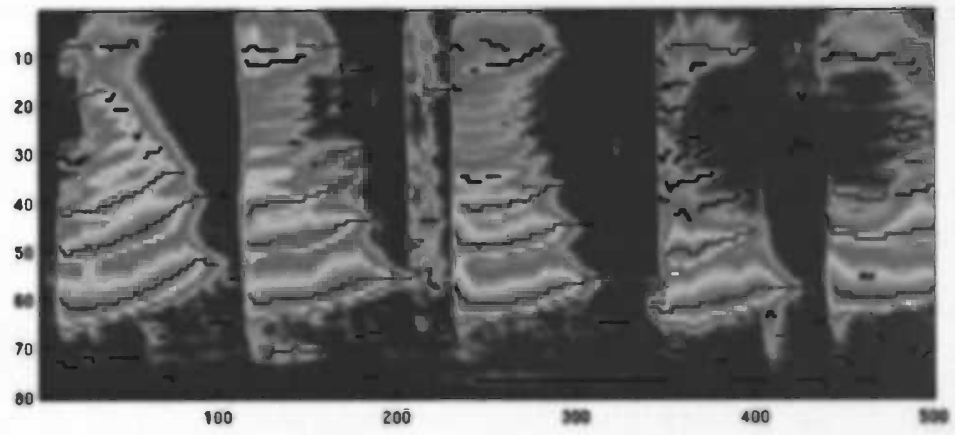



Figure 6: Energy-matrix of the signal [nul een twee drie vier] with the NSCs filled in. Blue lines represent NSCs that were typed as 'sine', white lines were typed as 'pulse' and the black lines are untyped

NSCs are part of other components so they can be combined to form a bigger component. Knowledge of the way sines and pulses behave is used to accomplish these goals.

This stage will deviate somewhat in approach from the former stage. The scope of the stage is not limited to the slice of the signal we feed it, but extends to previous and following slices. To form components out of the NSCs that were acquired, hypotheses about the existence and continuation of sines and the place in time of pulses are formed and checked. This stage also tries to make a more rigorous sifting between the 'used' and 'unused' components - at the end of the stage, we are left with a number of components. The stage will consist of several modules: The forming and extension of sines, the forming and domination of pulses and an added two modules for eliminating obvious unwanted NSCs from the pool: on- and offsets and 'ringing out' reverberation effects of the cochlea. All NSCs that remain unused or uneliminated by any of the modules are marked as 'non-typed'.

### 3.3.1 Using NSCs to build sines

For this module we assume that NSCs which were typed earlier as 'sines' can be used as the basis for sine-components without any further checks. Because the element was shown to fit positively to the sine-mask, we can assume it represents a sine. The module searches through the set of NSCs. When one is found that is typed as sine a new sine-component is created.

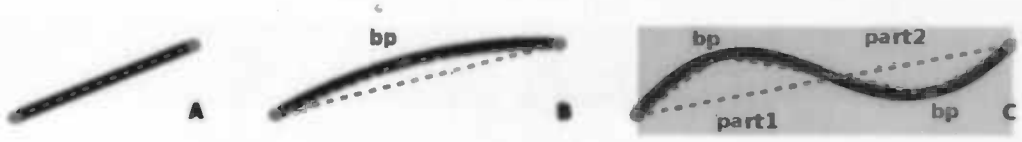



Figure 7: Three examples of sine-type elements. A is a straight line through start and endpoint, as seen when the striped red line is fitted to the black sine-type element. Sine B has a hump, and cannot be represented correctly by it's start- and endpoint (see the green striped line). A 'bendpoint' has to be added (point 'bp', see the red line again). Figure C represents a sine that has to be treated as two parts to be represented correctly. The green line again shows the error obtained by only using the start- and endpoints, the red line shows the better fit of a two-part, start, middle, end - approach. )

NSCs that are already 'in use' by other components are of course ignored in this step.

At this stage defining how a 'component' is actually implemented becomes useful. A pulse-component is implemented as a simple combination of the time when the pulse occurred and the relative amplitude it had. A sine-component is more involved, and has two levels of description. At the lowest level of description lie three arrays: the frames the sine runs through, the segments it occupies at those frames, and the energy it has at that frame. With this data, a precise copy of the original component can be made. At the highest level of description a sine-component consists of a number of points in time and frequency that are important to the definition of the sine. The most important point to consider are the start-and endpoints of the NSCs that make up the component, it would seem. Using only these point will give lots of errors if the NSCs are curved. This implies extra descriptive points should be added at the 'bendpoints' of the curve - defined as the point where the curved line has maximal distance from a straight line through start- and endpoint.

Adjusting our description to make room for these curved NSCs brings to light another related problem. As seen in Figure 7 a NSC can contain multiple 'parts' that each consist of a start-, end- and bendpoint. This should be held in consideration when traversing NSCs to construct sine-type components. Finally, when the component is started, the link between the new component and the NSC that gave birth to it must be administrated.

**Algorithm 5** - building sines

Loop over all NSC's, if the NSC is typed as a sine, and not a part of a component then do:

(1) Create a new component (of type 'Sine')
(2) Find startframe and -segment
(3) Find the 'parts' of the NSC (by looking at the second derivative)
(4) For every part, do:
(4.1) Find if the part has a 'hump', by looking at the maximum deviation from a straight line from start to end
(4.2) Add the startpoint to the component
(4.3) If there's a 'hump', add point where it is maximal (the 'bendpoint', second derivate is zero) to the component
(4.4) Add the endpoint to the component
(5) Calculate the average amplitude, devide by the max amplitude to scale it
(6) Add a copy of the energy, frame- and segment-array of the NSC to the component (for eventual exact reproduction if it might be needed).
(7) Administrate the link between component and NSC-element

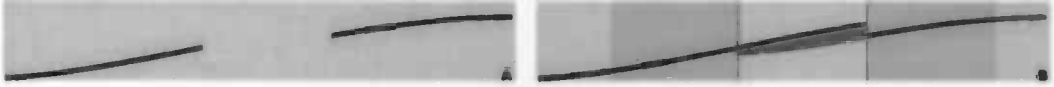


Figure 8: An example of mirroring. Part A shows an interrupted line, perhaps by some noisy interference. Part B shows the process of mirroring. The vertical green lines demarcate the ends of the original lines . The light green area is taken, mirrored vertically and horizontally, and pasted behind it's 'parent'. This creates the lighter blue and red lines between the green vertical demarcations. The purple area between them indicates their distance.

### 3.3.2 extending sines

The next step after creating a new component is seeing if it can be extended. We use a greedy algorithm for this, so that a component tries to extend itself as much as possible. To extend the component, candidates must be found, and the simplest way to find these is to simply probe around the end of the component. The question arises which area should be used to find elements 'in range'. Too large an area results in unneccesary testing, while only considering a small area might miss valid continuations. Some experimenting indicated an optimal target area from the endpoint of the component of -5 to +14 frames and -4 to +4 segments, so we use that for the system.

To check whether the NSCs that are found to be in range of the component are actual continuations of that component, some form of fitness-test has to be performed. Using mirroring to see if the current component and chosen NSC are part of the same sine gives 'distance' between the component and NSC as a measure, see figure 8. This works quite well, and we assume that giving some leeway for error and allowing one segment average distance will assure the correct NSCs are added to their parent-components, while still preventing erroneous extensions.

The type of the NSC being used for extension is not really relevant in this step. As long as the NSC is of type pulse, sine, or untyped, it can be used for extension as long as the mirroring-distance limit holds - why shouldn't it be added if it fits so well.

### 3.3.3 building pulses from NSC's

Having extended all sines to their maximum, we can begin to look at pulses. This module is used to start new pulses, Just as with the modules dealing with sines, the cycle goes Search-Find-Extend-Search. Loop-

**Algorithm 6** - extending sines

```
(1)   Create a range from the endpoint to check for
      candidates
(2)   For all NSC-elements that fall into the range,
      and are not part of this component yet, do the
      following:
(2.1)    Mirror the end of the component, and the start
         of the element under scrutiny, and add to their
         parent.
(2.2)    Check the distance between the component and
         element
(2.3)    If the distance is sufficiently small (maximum
         of one segment on average), add the element to
         the component (following the procedure used above)
(2.4)    Administrate the link between the component and
         the NSC-element that was just added.
(3)   When all elements in range are exhausted, start over
      from the end of the grown component.
```

ing over all NSCs, if one is found to be of type 'pulse' it is used to start a new pulse-component.

**Algorithm 7** - building pulses

```
Loop over all NSC's, if the NSC is a pulse, and not
part of a component then do::
(1)   Create a new component, type Pulse
(2)   Find startframe
(3)   Set length of the pulse (standard: one frame)
(4)   Calculate the average amplitude, devide by the
      max amplitude to scale it
(5)   Administrate the link between component and
      NSC-element
```

### 3.3.4 Checking for overlapping pulses

A pulse in the signal can conceivably yield more then one pulse-typed NSC. To prevent these multiple NSCs to form their own overlapping pulse-components, a check for overlap is executed. The area a single pulse in-

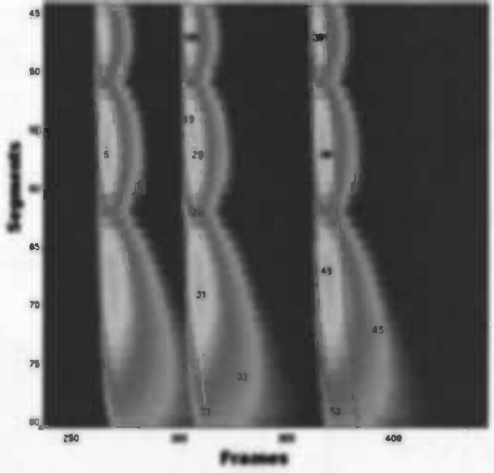


Figure 9: The two red lines demarcate the area dominated by the pulse formed from NSC 34.

fluences is fairly stable, and can be estimated by laying a line from the first frame of the pulse along all segments - this is not as straightforward as taking a purely vertical line, but should take group-delay into account. This creates the leftmost edge of the pulse-area. The other edge is formed by adding the group-delay factor for each segment to that segment, plus an additional shift to allow for the 'width' of the pulse. Call the left edge $El$, the right edge $Er$ and the group-delay factor $gd$, we get the following basic method to find the right edge: $Er_{seg} = El_{seg} + gd_{seg} + A$, where $A$ is a constant (with a value of 4 segments - found before to produce good results, which also seems to hold here).

For the time being a standard shift of four frames gives satisfying results, and seems to be a good-enough estimation of the width of a standard pulse. This should be replaced by a better motivated solution, perhaps finding the width of the standard pulse that was routed through the cochlea at startup and used to define the pulse-mask, created earlier. Figure 9 gives an example of the area dominated by a pulse.

All NSCs that are typed as pulses and fall into it's area are added to the pulse-component. Because these NSCs are linked to their parent component, they will not be used to form components of their own. For possible later use, to emulate the pulse more accurately, for instance, the average central segment of the pulse is estimated by taking the average center segment of all NSCs that are part of the component.

**Algorithm 8** - combining overlapping pulses

```
(1)   Create a line over all segments through the
      frame and center-segment of the pulse and cor-
      rect for groupdelay (gdline)
(2)   Create a second line: copy the first line, add
      the groupdelay-factor, and shift some frames to
      allow for pulses being detected later in the life
      of the pulse
(3)   Check if other NSC's of type pulse fall between
      those two lines, if so:
(3.1)     Change the type to 'part-of-other-pulse'
(3.2)     Administrate the component the NSC-element
          'belongs to'
(3.3)     Adjust the average segment of the pulse
          (the 'center' of the pulse)
```

### 3.3.5 Eliminating onsets and offsets

The following two modules are used to eliminate unwanted NSCs from the search-space. Some anomalous NSCs will emerge, no matter how carefully the NSCs are constructed, and these modules seek to remove the most common occurances: On- and offsets and decaying tones that are an artifact of the processing method, also called 'ringing out'.

This module deals with the pulse-typed NSCs that tend to appear at the onset and offset of a sine-type component. To eliminate these, the naive method of probing around the start- and endpoints for pulses is used. This method gives the desired result - elimation of onset- and offset-pulses - but tends to remove legitimate pulses that happen to occur close to the start- and endpoints of sines. Because false pulses distort the resynthesis of the signal more than unjustly removed pulses are missed, the naive method is good-enough for this application. A better algorithm could perhaps check back in the energy-matrix to see if the suspected pulse really occurs at a onset- or offset-position. The current method implements a simple form of this check by looking at the 'centre' of the pulse, as estimated before.

### 3.3.6 Eliminating 'unnatural' frequency-invariant sines

The other oft-occurring anomaly we try to eliminate at this stage are unnatural frequency-invariant sines. These are mostly caused by *ringing out* of sines in the cochlea, and marked by smoothly decaying energy. As this

**Algorithm 9** eliminating onsets and offsets

```
(1)   At the start and end of sine-type components
      do:
(1.1)    Probe around the start- or end-point, check
         the NSC's there
(1.2)    If there's a pulse centered there, remove it
```

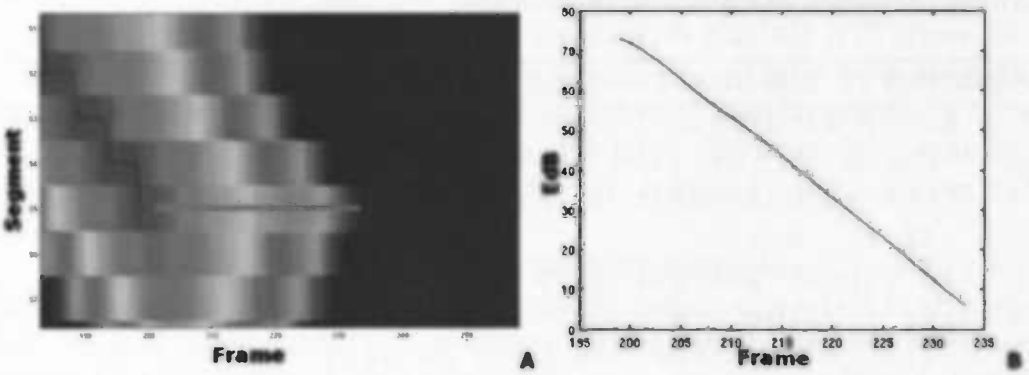



Figure 10: A decaying pure tone - 'ringing out' (A) and its fall in energy (B).

ringing out is an effect of the cochlea, it would be wrong to duplicate these sine-like effects in the resynthesis of the signal.

To remove these anomalies, see if any frequency-invariant sines are present. If so, we can check its type; three different types of frequency-invariant sines can be distinguished: NSCs that are part of a pulse, but fall in the sine mask. NSCs in pulses sometimes show as frequency-invariant elements, if that frequency range is strongly present throughout the life of the pulse. This type can safely be ignored, and will almost never occur if the mask are good enough. The second form is the *ringing out* mentioned earlier, as this is a decaying tone who's energy reduces with time because after the abrupt cut-off of a tone, the time integration of energy (with $e^{(} - \lambda t)$)causes a sloping decline in energy (lasting about 10ms) - it should be ignored in further processing stages and typed as the anomaly it is. The last type we can encounter is the true pure tone, most often seen in artificially generated signals. This should be left alone.

The algorithm will deal with the 'decaying' type of frequency-invariant tonal component (see also figure 10). Checking whether the decay of the energy of the NSC is steady is implemented somewhat naively by con-

structing a decay-array (where $Decay_x = E_x - E_{x+1}$), so it's length is one less than that of the Energy-array of the NSC. If the distance of the decay is big enough - ringing out is quite noticeable when the component that causes it is high in energy, checked simply by looking at $max(E) - min(E)$, and the difference between the mean decay and the mean step needed to decay from $max(E)$ to $min(E)$ is small enough ($dif = abs((max(E) - min(E))/numel(E) - mean(decay))$), the NSC can be marked as 'ringing out'. As an extra check, the standard deviation of the decay-array $std(Decay)$ should be small, to ensure smooth decay.

**Algorithm 10** - eliminating frequency-invariant sines

```
for every sine-type NSC-element, do:
(1)  Check if the sine is a pure tone (stays on
     the same segment (min(seg) == max(seg))).
(2)  If so, do:
(2.1)   Check for steadily decaying energy:
(2.1.1)    Create a decay-array
(2.1.2)    Check the distance between max(E)
           and min(E)
(2.1.3)    Check if the average decay-step
           needed to get from max(E) to min(E)
           is equal to the average real decay
(2.1.4)    Verify the decay is steady by looking
           at the stdev of the decay-array.
(2.2)   If the decay is steady, mark the NSC
        as type 'ringing out'.
```

## 3.4 higher level processing: solving crossings and tracing sines

After the previous stages of processing are done, we are left with a collection of components, both pulse and sine. What should be done with those components will be largely decreed by the application of the system. Higher level modules should be added as they are needed. The great advantage of such a modular system is that it is highly flexible, and that a form of unit-testing[3] can be applied to check the separate modules.

[3]Unit-testing is the method of testing parts of a system in isolation from the rest of the system, thereby assuring the correct operation of that part.

In this section, we will discuss a few more generally applicable higher level processes. As the actual implementation of these processes largely falls outside the scope of this work, description is mostly limited to a theoretical form. Where a real-world implementation of a module was created and tested, this will be mentioned in the text.

The first two modules we cover are used to remedy some rather inescapable fallacies of the system, namely crossings and sines that fall outside the sinemask. We shall look at a way to form hypotheses for the parts of the signal where a sine should probably be, but no NSC was found (crossings) or where gaps where too large. This module will show that it is not necessary for proposed modules to confine themselves to working with the collected NSCs, but that they should also be able to order a 're-scanning' of the signal, as it were. We shall also scan the remaining untyped NSC's for potential sines, that were missed by the masking and extending processes.

The last module in this section proposes an example-module for a hypothetical application of the system, namely speech extraction from a signal. Several other modules to perform higher-level processing were developed and tested, such as a method to eliminate errant 'clicks' where a part of a speech signal should be - the slope of some of the formants was so steep they got into the pulse mask. With initial testing, it appeared the assigned tasks of these modules could also be performed by some smart tweaking of the thresholds involved, so the modules were discarded as superfluous.

### 3.4.1 Resolving crossing problems

When sufficient energy is present under the gap between two sines that could be connected, they should be connected. If such gaps occur, it quite probably means that the threshold-values used to create the peak-matrices are set too high, and should be lowered for this slice of the signal to detect all occurring sines. A rescan might be in order, resulting in a new set of NSCs. When overlap is eliminated, only the newly detected bits of the NSCs will remain, and if the extension-modules are run again the NSCs that were added and are part of a component will be added to that component. The difficult part of this module is determining when a rescan is prudent, because doing so will require extra processing time. A preliminary version of this module performed quite good, but slowed processing down and made assessment of the performance of the other parts of the system more difficult, so is left out of the final system.

This does not solve the problem where a choice has to be made be-

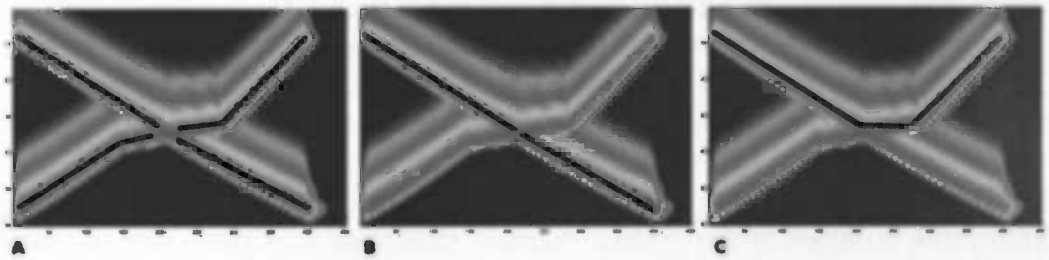



Figure 11: Part A shows an uncertain situation. Four sine-type components meet at one point. Assuming the source consisted of two components, possibility B (crossing) is just as likely as C (touch and retreat). More information is needed to resolve this situation

tween crossing sines or 'bending' sines. Other information should be used here. A possible solution would be to look at the phase-shift information to resolve a choice, or to trace based on most-likely situations - try to trace a line through all possibilities and take the one that seems to have the best fit with the actual energy-matrix. Figure 11 illustrates this problem. There is no implementation of this module, because it would fall outside the scope of this work. Also, there is the problem of human-solvability. Some of these crossing-problems are not easy to solve, even for the human observer. It would be overly ambitious to address this difficult problem as a small aside to the workings of the system.

### 3.4.2 Finding sines by looking at smoothness of energy

After pulses and sines are formed from typed NSCs, a number of untyped NSCs remain. Some of these untyped NSCs could well be sines that fall outside of the sine-mask, for instance, low-energy harmonics will probably not be found in the sine mask, unless they are removed quite a few segments from the higher energy harmonics. Even though these NSCs are not present in the sine-mask, they are still sines, and should be treated as such by the system.

To find these unjustly untyped NSCs, we look at the length of the NSC and smoothness of the energy of the NSC. The NSC should span enough frames - some testing resulted in a minimum of 10 frames. NSCs that were shorter than this size rarely could be marked as a sine. Furthermore, the energy under the NSC should have a smooth fluctuation. Assessing smoothness of fluctuation is not as trivial a process as it might seem at first sight, so we try to approach it by looking at the deviation of the energy from a smoothed version of itself, $mean(abs(E - smooth(E, 3)))$. Fig-

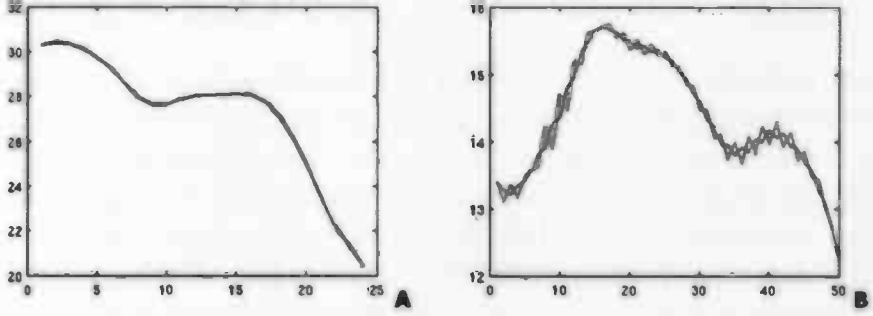



Figure 12: The red lines represent the energy of the NSCs shown, the blue line is a smoothed version (moving average, 3 points). The NSC in A is untyped, but is actually a low-energy formant (of [drie] in [nul een twee drie vier]. It's smoothness is 0.069. Part B shows the energy of a noisy component that should remain untyped. It's smoothness is 0.148.

ure 12 gives an example. The thick red line is the energy of the NSC, the smaller blue line the smooth variant, smoothed with Matlab's Curve Fitting Toolbox, which applies a three-point moving average. The threshold for smoothness is again difficult to ascertain on other grounds than experimental ones, so we use common sense and pick the value that seems to produce the best results.

### 3.4.3 Looking at harmonics to find sines

This proposal for a module is intended to give an example of what a higher-level module could work like. Assuming an application where harmonics have to be found, an f0-estimate (Sound Intelligence has an estimator built into the system)can be used to find the base-frequency. Using that, two different paths can be taken to find as many harmonic elements in the signal as possible. The simple solution is to check if any remaining NSCs of type 0 can be found at harmonic places, and to convert them into sine-components. The more complicated solution, that will result in more harmonics being found as components, involves processing parts of the signal again, but with the sin-mask extended to include the harmonic segments.

# 4 Testing & Results

By implementing the algorithms that were developed in the last chapter, a working system for the detection of pulses and sines in sound was built. The questions that remain are how well it actually functions, what it can and cannot recognize, and what further improvements could and should be made. In this chapter, the system is tested on both artificially created and real-world signals. A system is developed to resynthesize the detected components into a signal, so that comparisons can be made, both by comparing the actual energy present in the signal and it's resynthesized counterpart and by doing a comparative listening-test.

## 4.1 Methods of testing

As mentioned in Chapter 2, finding a correct measure to assess the performance of the system is quite a difficult undertaking in itself. We shall use two relatively simple methods. One consists simply of listening to the original signals and their resynthesized counterparts, and describing the fallacies that are encountered. This also encompasses scrutinizing the visual representations (the energy-matrices) of both the original and resynthesized signal for obvious differences. The second method involves calculating the average difference in energy of the cochleogram between the original and resynthesized signals.

It becomes apparent that having a good method of resynthesis is quite important if we are to perform any kind of meaningful assessment. The components that are found by the system are used to create snippets of signal representing those components, and then merged together to, hopefully, create a facsimile of the original signal.

### 4.1.1 assessment by listening to the reconstructed signal

The most basic way of assessing the performance of the system is to simply listen to the original and reconstructed signals. This quickly gives a basic idea of performance, and comes in especially handy when working with natural signals, where the full content of the signal is not known in advance, and plenty of noise and other types of interference which can distort measurements are present.

There are some caveats when using this procedure. It can't really be used to acquire hard, reproducible data (but can still be used to give an indication of performance). Furthermore, the listener is introduced as an extra factor, which means that multiple subjects should undergo multiple

listening trials to rule out interindividual variations, if any usable data is to be collected.

Setting up a listening experiments entails making some choices. Ideally, a double-blind test would be performed, presenting a subject with both the original and resynthesized signal, who would then have to indicate which is the original, and which the resynthesis. Practically, problems arise. When dealing with artificial signals, how would a subject differentiate between original and resynthesis, when there is no information indicating originality? The test becomes meaningless when trying to differentiate. To fix this, another approach could be used with regards to artificial signals: Asking the subject whether the signals are the same, and varying between presenting [original, original] and [original, resynthesis].

When trying to double-blind-test natural signals, the natural signal will always be found immediately, owing to the simple set-up of the resynthesis and it's inability to deal with noise, an integral part of any natural signal. A better method would be to ask subjects to identify important parts in the signal, and ask whether they hear those important elements in the resynthesis - these are the elements we are interested in, after all.

For our simple purposes, the tests performed will be kept minimal, without using any complicated procedures. Only a few subjects (two) will be used, and the set-up of the experiment will be kept simple, as described further down. Furthermore, a visual comparison of the original and resynthesized signals will be used, complimentary to the auditive method of assessment.

### 4.1.2 Assessment by percentage of energy explained

While listening to and looking at the original and reconstructed signals might give a good indication of quality, some way of quantization of the results is needed to make objective claims about the quality of the system - and to effectively measure the effect of changes made to try and improve quality.

A simple, but quite effective, method immediately springs to mind. As we have the energy matrix of both the original and reconstructed signals, we can compare the two and derive a measurement for quality. To accomplish this, we take the cochleograms of the original and resynthesized signals, subtract them, and look at the (absolute) remaining energy. The mean of the resulting absolute energy will be used as a final measure $mean(mean(abs(difMatrix)))$. Figure 13 illustrates the process.

Some problems remain, foremost of which is the problem of noise, which won't be recognized by the system, and thus won't be present in

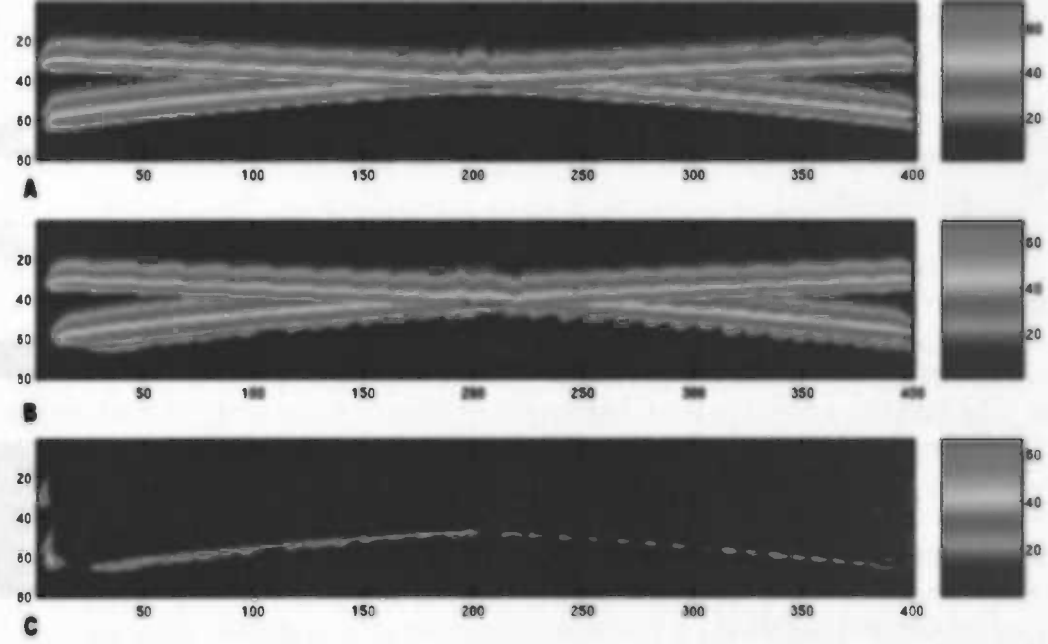



Figure 13: Part A shows the original signal, part B the resynthesis (see further along the text for a description of the method of resynthesis). Part C shows the difference between the two.

reconstructions. This introduces errors in the measurement. Not recognizing noise is a desired trait, but the presence of noise in the original signal skews the result of the assessment. To fix this problem, a mask could be overlayed, exposing only the relevant parts of the original signal for comparison. This eliminates unwanted noise, but will bring it's own problems: What to mask, how do you automate the masking process, and is masking not a subtle form of cheating - especially when using the exact same signal components to make a mask as were used to create the resynthesis?

### 4.1.3 description of the setup of the test-program

Keeping in mind the preceding points, we can formulate a program for a basic test of the performance of the system. Using a battery of test signals, perform the following steps:

1. Synthesise the signal that is to be tested at 44.1 kHz, and read the generated wave-file into matlab.
2. Run the system, acquiring a list of components and a reference cochleogram.
3. Resynthesize a new signal, based on the newfound components.

4. Save the resynthesis as a wave file.

5. Run the resynthesis through the cochlea.

6. Compare the old and new energy-matrices, computing their mean difference.

At this point, differing paths will have to be taken depending on the nature of the signal.

For artificial signals:

7 Listen to a few trials of different combinations of original and resynthesis (O,O and O,R), and see if original and resynthesis can be differentiated.

8 Finally, make a visual comparison.

For natural signals:

7 Listen to the original signal, noting the important elements.

8 Listen for the occurance of these significant elements in the resynthesized signal.

9 Finally, make a visual comparison.

The method used to test whether original and resynthesis can be differentiated is ABX testing[4], where the listener has to identify an unknown sample X as being A or B, with the original samples A and B available for reference. This trial is repeated multiple times (25 per signal-pair, in our case), resulting in a score (percentage correct) and a $\rho$ value indicating statistical significance of the resulting score.

### 4.1.4 description of the method of resynthesis

As mentioned at the onset of this chapter, and touched upon implicitly throughout it, having a good method of resynthesis lies at the heart of good assessment. Without it, comparison between what we found and what is in the signal would be limited to reading through the components and mentally pasting them onto the original signal to see if they fit[5].

[4]The software used was PCABX (http://www.pcabx.com/).

[5]Actually, a method to do just this was created for debugging purposes - see plotSig.m in the Appendices

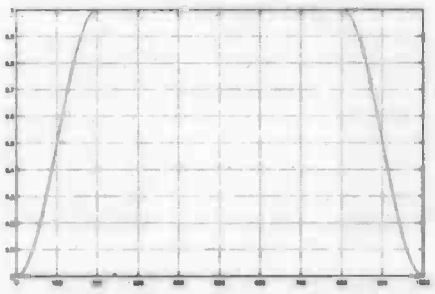


Figure 14: An example plot of the $cos^2$ tapering-window. The width of the taper is set to 20% at both ends for demonstration-purposes. Normally, it would be 2% in our application.

Actually developing a good method of resynthesis took up some more time than expected. Questions arose regarding finding out the correct amplitudes with which to recreate components - a conversion from dB to a percentual scale was neccesary to adjust the amplitude of the components, with the question of finding the exact comparative 'loudness' still open - and the best way to recreate pulses and sines.

At the start of the resynthesis process the type of the component to be resynthesized is checked. If it is a pulse, a small snippet of signal is created, containing the pulse. We use a standard square pulse at the moment, but quite different types of pulses, with differing attributes, are possible. The hurdle that remains for any kind of differentiation between pulses is defining and detecting whether a pulse is of a certain type.

If the component is a sine, the method of recreation is somewhat more involved. There is a choice of two levels of complexity; an exact reproduction of the component with the information taken from the component's NSC's[6], and a - much more compact in terms of data present - approximation by using a logarithmic or linear sweep through key points of the component[7]. The amplitude of the component is ascertained by, in the case of exact replication, using a per-frame energylevel which was taken from the component's NSC's, while the approximation method uses a mean energylevel, found at the time of detection/creation of the component.

To avoid introducing artefacts like onsets and offsets, the snippets of signal that are created by the above two processes are tapered at both sides, using a $cos^2$-window, as illustrated in figure 14. Then the snip-

[6]implemented in generalsweep.m

[7]implemented in logsweep.m and linsweep.m

pets are mixed together to form the main signal, thereby creating the final resynthesis.

A complete listing of the resynthesis-code can be found in the Appendices.

## 4.2 Signals & testing

A body of appropriate signals, both natural and artificial, will be used in the execution of the tests. Artificial signals are used because they can easily indicate specific stronger and weaker points of the system. Caution is necessary, because the exclusive use of artificial signals in testing gives no real indication of the usefulness and performance of the system in real-world situations - while a real, usable system is a desirable outcome. To give an indication of the system's performance on real world signals, a range of 'natural' signals is also queued up for testing.

The signals will now be presented with a short description, and a cochleogram[8].

### 4.2.1 Artificial signals

The following signals were generated to test the system. Noise was added to signals 9-12 with Adobe Audition. Fig 15 shows the cochleograms of the signals. The corresponding descriptions of the signals can be found below.

1. A 200 Hz sine, lasting 2 seconds.
2. Two sines, at 200 and 220 Hz, combined to result in a modulated tone, lasting two seconds.
3. Two crossing sines, one linear sweep from 200 Hz up to 1000 hz, the other sweeping down from 1000 to 200 Hz. The two cross after one second at 600 Hz.
4. The same signal as (3) above, but the amplitude of the downward sinesweep is set to a tenth of that of the upsweep.
5. Multiple Sinesweeps following each other, varying between 200 and 2000 Hz - totaling 2 seconds.
6. Two strings of sinesweeps, crossing multiple times. Just like (5) the frequencies range from 200 to 2000 Hz.

[8] Audio versions of the signals are available on http://pim01.nl/classification/

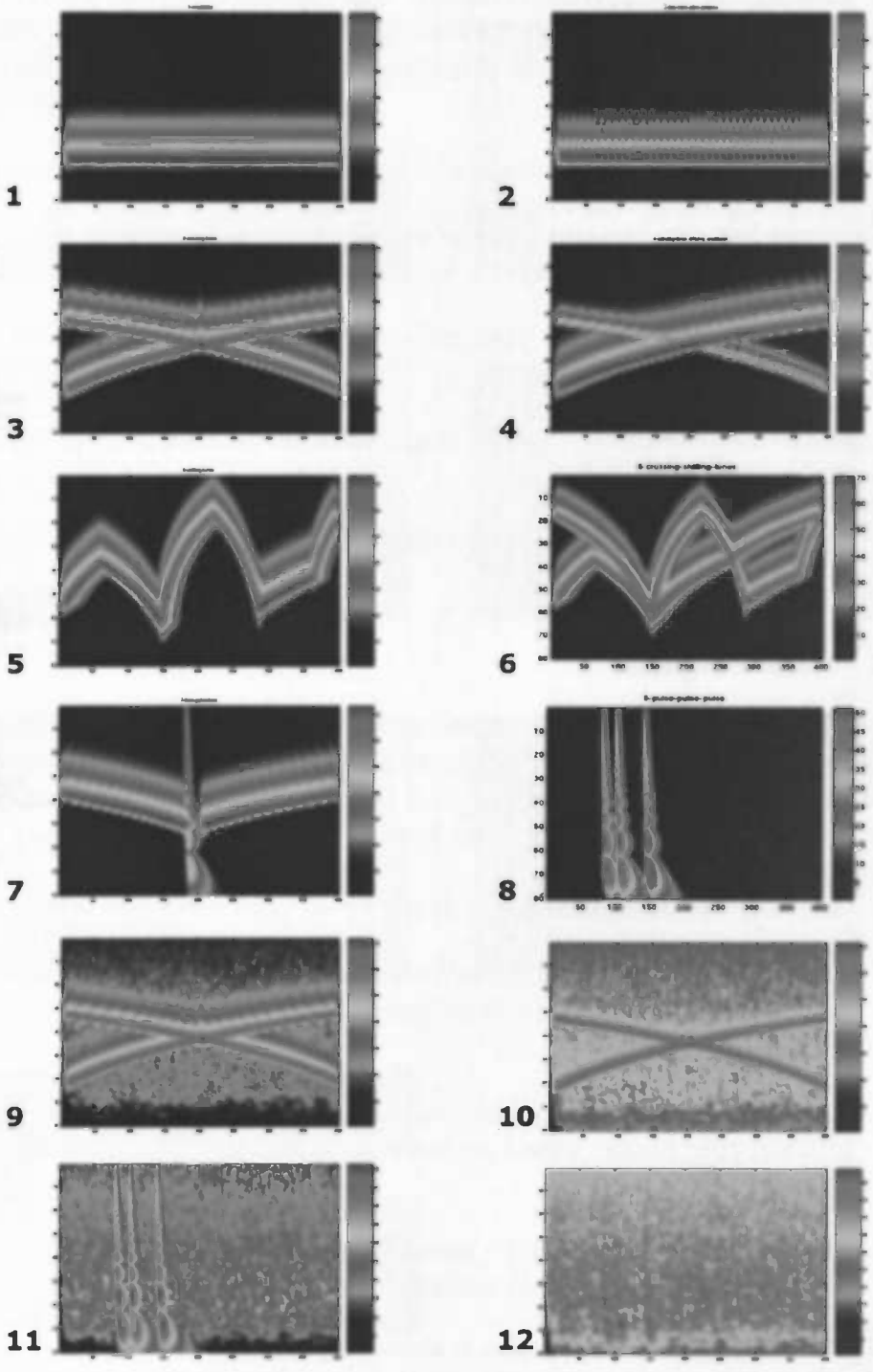



Figure 15: The artificial signals generated for testing the system.

7. This signal contains a sinesweep, descending from 800 to 400 Hz in 0.9 seconds, immediately followed by a pulse. At the 1-second mark, a second sinesweep starts, this time climbing back up from 400 to 800 Hz in one second.

8. Three pulses, one at 0.4, one at 0.5 and one at 0.7 seconds.

9. Signal (6), mixed with pink noise[9] with an RMS energy level 33dB below that of the pulses.

10. Signal (6), mixed with pink noise with an RMS energy level 14dB below that of the pulses.

11. Signal (8), mixed with pink noise with an RMS energy level 33dB below that of the pulses.

12. Signal (8), mixed with pink noise with an RMS energy level 14dB below that of the pulses..

### 4.2.2 Natural signals

The following natural signals were used to test the system. They were picked to represent a range of possible situations, from babble with some information, to speech, to slower and quicker music. Again, audio versions of the signals are available on http://pim01.nl/classification/.

1. A woman pronunciating the dutch words nul een twee drie vier.

2. A woman pronunciating the dutch words nul een twee drie vier, mixed with pink noise with an intensity of 40 (see also the preceding section, signal (9)).

3. An excerpt from Liquid Tension Experiment - Paradigm Shift. Rapid and varying music, based on keyboards, guitar, Chapman Stick and drums.

4. An excerpt from Loituma - Ievan Polkka. An a-capella track, featuring both male and female voices.

[9]The cochlear place-frequency map is almost logarithmic. Therefore a pink noise, which has an even energy density per octave, will produce an almost flat average excitation in the cochlea. As the Cochleograms we generate (and human hearing, for that matter) use a logarithmic (Db) scale, this type of noise seemed the most appropriate to choose to 'pollute' the signal with.

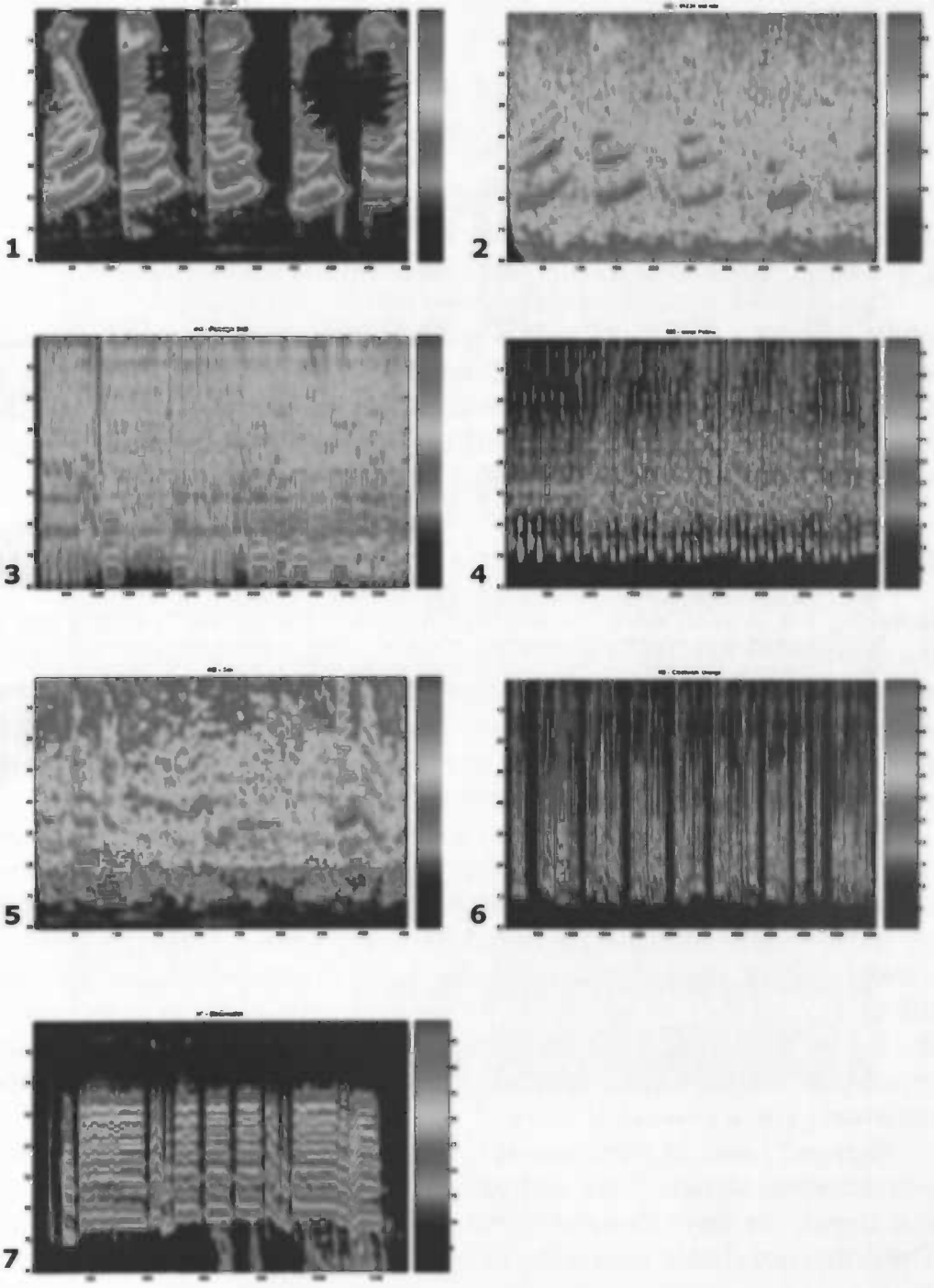



Figure 16: The natural signals used to test the system.

5. Babble and background noises, with some clearly audible saxophone notes woven through.

6. An excerpt from a reading of Anthony Burgess' A Clockwork Orange.

7. A person imitating a horn.

## 4.3 Results

While preparing to run the tests, some hitherto undiscovered small faults and bugs in the system had to be fixed. The bigger problems that came to light will be discussed in the Recommendations section of chapter 5. The following section describes the result of running the previously discussed sets of signals through the system.

### 4.3.1 Results - Artificial signals.

Running the system with the artificially generated signals yields a list of the mean differences between original and reconstructed signal, as shown in table 1. Signals 5 (multiple sweeps) and 12 (clicks in high-level noise) show a remarkably high difference in energy, while signal 6 (multiple sweeps, crossing) also seems to have quite a high difference in energy, signifying problems with correct recognition of the components.

The ABX testing results showed a steady 100% differentiation rate between the original and resynthesized signals. It should be noted that resynthesized signals 9-12 had the appropriate amount of pink noise mixed in, so that a fair comparison could be made - the resynthesis leaves a 'cleaned' signal, after all. When askeds, subjects would indicate they were differentiating the two signals on subtle differences, such as short gaps in the signal at crossings and small differences in frequency. The ABX test results are not shown here as a figure, because they all scored 100% differentiation (with a p-value of 0.05).

Figures 17 and 18 show visual comparisons between the original and resynthesized signals. Each comparison consists of three parts - the original signal, the resynthesized signal and the difference between the two. The difference shown here is the same energy-matrix as was used in table 1.

Signal 1 has been reproduced quite faithfully. Noticeable differences are the slightly higher frequency of the resynthesis - a difference of 6 Hz, still noticeable, as was shown in the ABX test - and the slightly belated

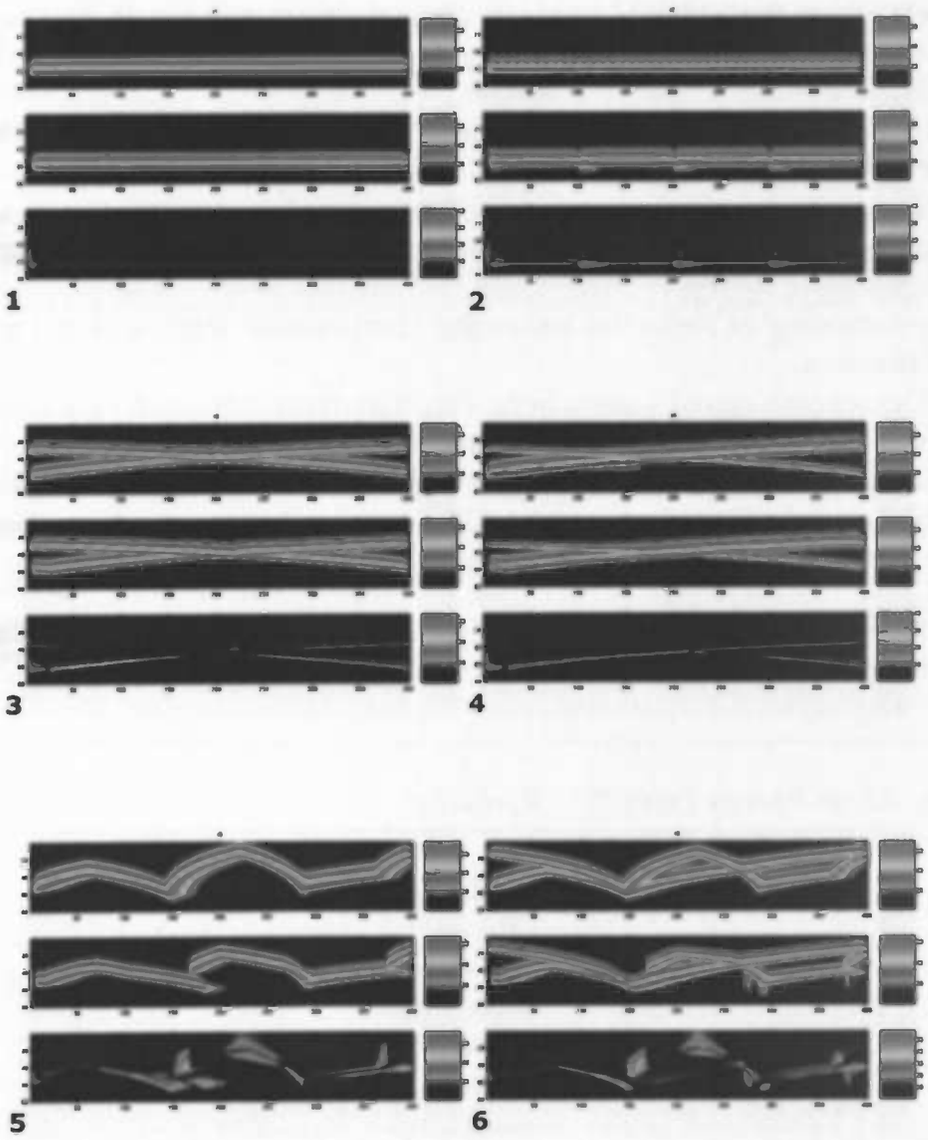



Figure 17: The first six artificial signals and their resynthesis. The figures shown consist of three parts - the original, the resynthesis, and the difference between the two. Colourbars besides the images indicate the amount of energy represented by the colours.

start of the resynthesized sine[10]. This slight shift in starting position will be seen in other signals.

The second signal shows a difference in modulation and three added onsets where there should be none, while the third and fourth signals suffer from the same shift in frequency as the first and second, and exhibit the added strange problem of an incorrect starting dip in the upsweeping component. Both signals 3 and 4 display an audible 'gap' at the crossing, where one of the sweeps dominates.

Signals 5 and 6, containing multiple sweeps, encounter more problems; whole pieces of the original signal are missing (the start of the strong upsweep beginning at 750 ms - frame 150), and there are a few errors with the positioning of some the remaining components' start- and endpoints and maxima.

The seventh signal suffers from a shift in discontinuation - the original 'gap' in the signal occurred at one second, the resynthesized one is positioned half a second later. A bigger problem, though, can be also found at the one-second mark, where a pulse was situated in the original signal. The system ignores this pulse, and in it's stead finds two small sines at har-

[10]The resynthesized signals tend to start about 10ms later than their original counterparts.

| Nr | Mean Energy Diff (dB) | Remarks |
|---|---|---|
| 1 | 0.963227 | Small shift in frequency. |
| 2 | 2.59771 | Difference in modulation, added onsets. |
| 3 | 1.95218 | Dip at start upsweep, 'gap' at crossing. |
| 4 | 2.23015 | Same as 3. |
| 5 | 7.06635 | Missing pieces, positioning errors. |
| 6 | 5.31015 | Same as 5. |
| 7 | 2.56776 | Pulse ignored, discontinuation shifted. |
| 8 | 3.68931 | Pulse classified incorrectly. |
| 9 | 2.01485 | Good reproduction, some superfluous elements. |
| 10 | 3.00574 | Same as 9. |
| 11 | 3.79404 | Pulse-recognition fails. |
| 12 | 9.32402 | Same as 11, many unjustly classified tones. |

Table 1: Result of the comparison between the twelve artificial signals and their resynthesis.

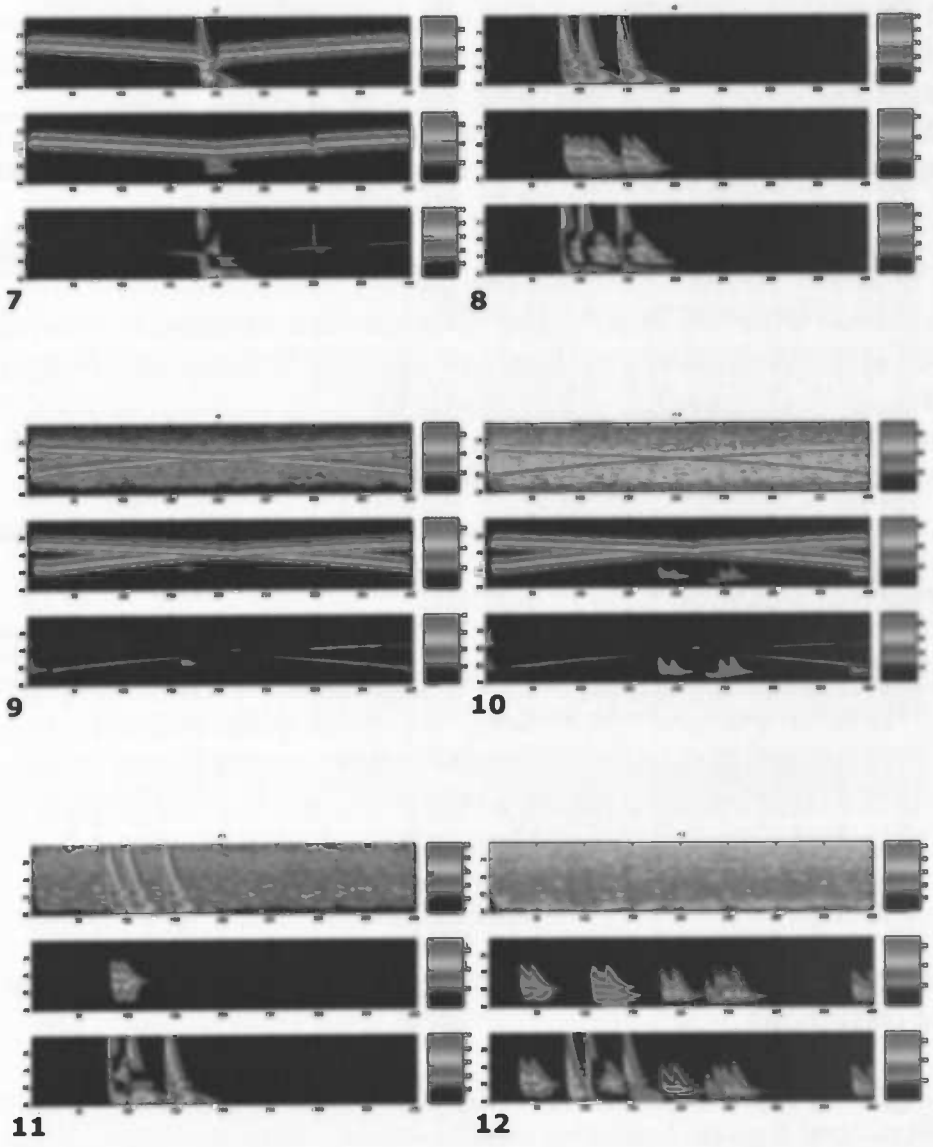



Figure 18: The second set of six artificial signals and their resynthesis.

monic places in the pulse. Signal 8 provides a clearer view at this problem with pulses. Only the ground frequency and one or two of the strongest harmonics are recreated, in stead of a correct pulse.

Signals nine to twelve aim to test the system with noisy signals. 9 and 10 show that the system works quite well in noisy circumstances, but adds more superfluous small components as more noise is added. Signals eleven and twelve show breakdown of the system with pulses in noise, with the system marking random bits of noise as small sines in the noisier signal twelve.

### 4.3.2 Results - Natural signals.

Calculating the mean energy difference between resynthesis and original is not as good an indicator of success of replication with natural signals as it is with artificial ones. Still, to give an impression of the parts of the signals resynthesized by the system, table 2 shows the mean energy differences for the seven natural signals the system was tested with.

| Nr | Mean Energy Diff (dB) | Remarks |
|---|---|---|
| 1 | 10.1657 | Problems with length tonal components. |
| 2 | 26.8732 | Same as 1, missing some higher formants. |
| 3 | 25.8211 | Sinusoids quite good, missing drums. |
| 4 | 9.10789 | Many missing formants, some clicks introduced. |
| 5 | 16.5894 | Most important tonal components found. |
| 6 | 8.97265 | Reproduction quite inaudible. |
| 7 | 9.36895 | Some pulses introduced unwarranted. |

Table 2: Result of the comparison between the seven natural signals and their resynthesis.

Figure 19 show visual comparisons between the original and resynthesized natural signals. Studying these comparisons and, more importantly, listening to the original and resynthesized signals gives an indication of the level of success the system achieves in isolating the major pulse- and tonal components from the signals.

Signals one and two are reproduced quite faithfully, except some of the higher, less clear formants escape detection (especially with [nul] in both signals). Also noticeable in signal one are the incorrect length of the first formant in [twee] and the error made in connecting the first formants of

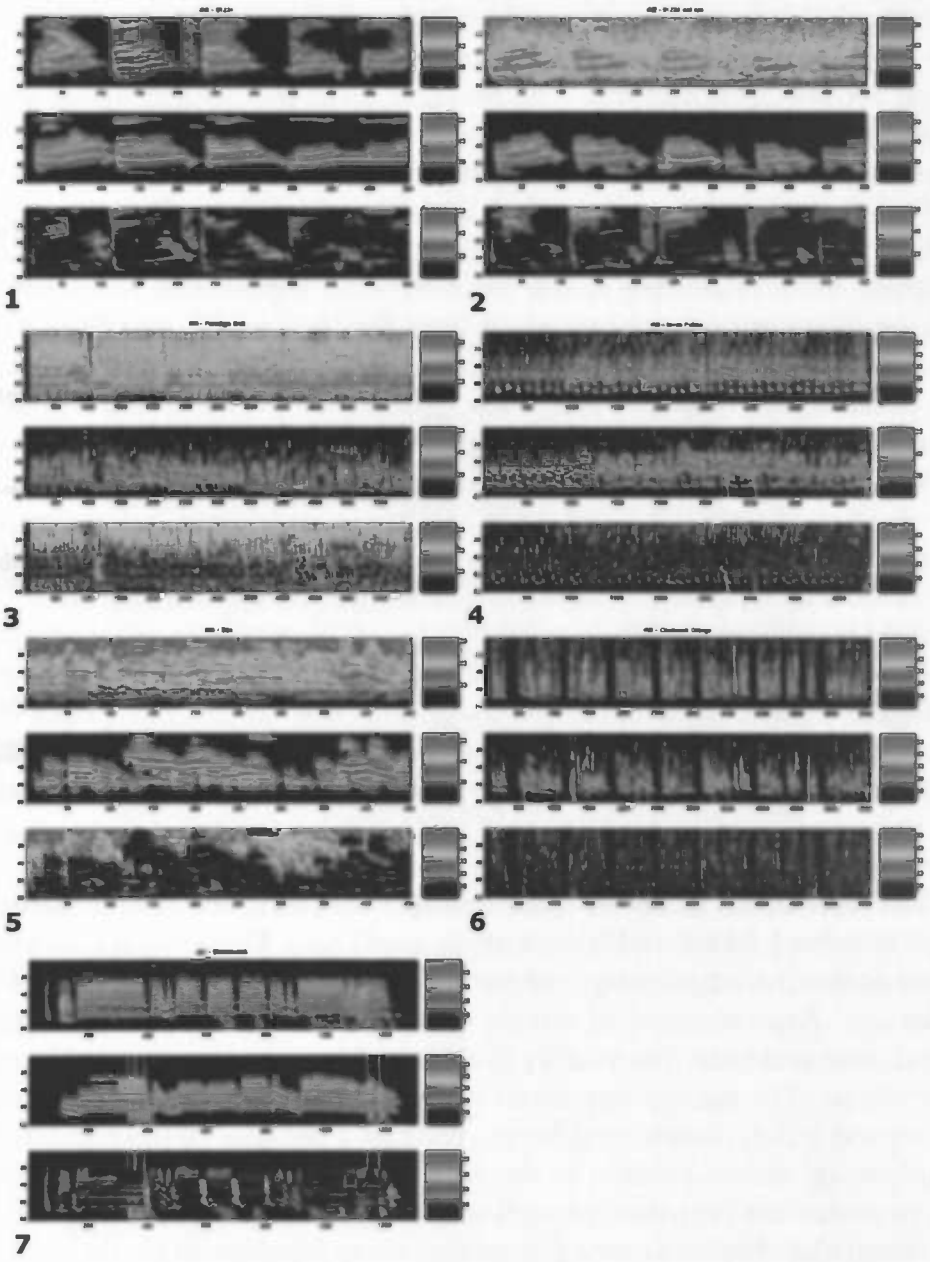



Figure 19: The seven natural signals and their resynthesis. See the figure representing the Artificial Signals for an explanation.

[drie] and [vier]. The noisy second signal shows better results regarding the length of the components. Apart from a high, warbling sound, the resynthesis of signal one is quite legible and the words are understandable. The 'bridged' formants between [drie] and [vier] are audible, and sound somewhat unnatural. The resynthesis of the second signal sounds more 'artificial', which would be consistent with the reduced number of reproduced formants visible in figure 19.

Paradigm Shift (Signal number three) has high overall energy, and as such only the most salient parts of the signal are recognized as tonal components. When listening to the reconstructed signal, this translates in a strange sounding piece of music, where the bass and guitar lines are reproduced and recognizable, but sound nothing like the actual instruments. The drum lines are somewhat ignored by the system, but can still be recognized here and there as short tonal components.

Vocal music (Ievan Polkka, signal four), is reproduced more faithfully. Figure 19 shows that quite a few of the formants present in the signal are reproduced, even some of the more salient ones of higher frequency. Some errors are introduced by finding pulses where there should be none, and - as could be seen in signal one - there is some excess prolonging of components, which causes connecting of formants that should not take place.

Signal 5 shows reproduction of some of the voices (the voices in the last part of the signal are quite clearly audible in reproduction) and of the melody line the saxophone plays. Most obvious important tonal elements are reproduced, but some are forgotten (most noticeably the upper formants of the first two words in the signal).

The resynthesis of Signal 6, the excerpt from Burgess' A Clockwork Orange, is completely inaudible when listened too. The flow of narration is there, as are the tonal components, but the actual words are very hard to make out. Reproduction of female voices seems to fare better, as seen in signals one and four. The warbly high sound is also quite noticeable in this resynthesis. The same phenomena can be observed in signal seven, where the reproduction does sound better, probably because of the nature of the 'trumpeting' sound present in the signal. The lack of higher formants in places makes the reynthesis sound somewhat lower than the original. Signal seven also displays some pulses that were introduced by the system at seemingly arbitrary places in the signal.

As an additional test of the system, signal two was cut into three pieces. which were run through the system. The resulting resyntheses were then combined, and the resulting energy matrix was compared with the original resynthesis of signal two. Figure 20 shows that the results the system generates vary with the chosen startingpoint.

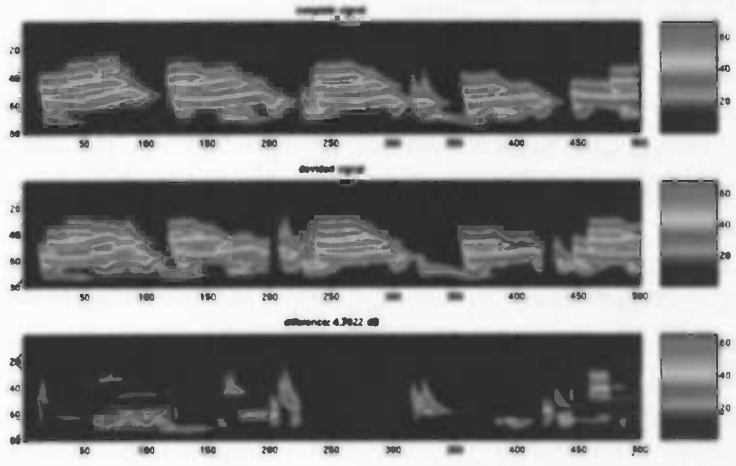


Figure 20: Comparison of the resynthesis of signal two and a version of signal two that was cut into three parts and recombined after resynthesis. The lower plot shows the difference between the two - with a mean difference of 4.78 dB.

# 5 Conclusions, recommendations, discussion

Now that the system is implemented and tested, we can discuss how well it actually performs it's intended tasks, give an indication of what works and what goes wrong, stipulate some reasons for those failures, and note which improvements could be made to improve performance.

## 5.1 Discussion

As can be seen from the results of the tests discussed in the previous chapter, the system performs it's tasks reasonably well when looked at at a very basic level (and as far as tonal components are concerned). However, while the system identifies major tonal components in a mostly correct way, there are many shortcomings to be found on closer examination.

The foremost failure of the system lies in it's inability to detect pulses correctly. Pulse-typed NSCs are sometimes found, but mostly pulses tend to be classified as a short tonal component with some more components above that in harmonic places. This unjust classification happens because the higher level processes that look at sine-like unclassified Narrow Signal Components classifies these sine-like components that lie outside of the sine mask as sines. Most times this classification would be correct, but in the case of these pulses, it is not. Figure 21 shows which NSCs were found in signal 8. The NSCs are as yet untyped, but should have been typed as pulses, as they lie in a pulse.

The errors with pulse-finding could have been fixed by improving the masks-creation stage, or taking pulse masks into account when searching for tonal components amongst the untyped NSCs. One other factor that could play a role in the troubles the system has with finding pulses could be the tendency towards a preferrence for sinusoidal componentens introduced by the methods that were used to generate the energy matrix. Energy tends to be 'smeared' in time, so pulse-like components quickly seem to be more tonal than they actually are.

Concentrating on the performance of the system concerning the detection (and reproduction) of tonal components, a few remarks can be made. The overall perfomance seems somewhat adequate, although there is a lot of room for improvement. When listening to the results of the test with natural signals, the major tonal components are present, although speech, for example in signal six, is not always reproduced in a meaningful manner[11].

[11] Although no understandable speech is resynthesized in signal six, the flow and in-

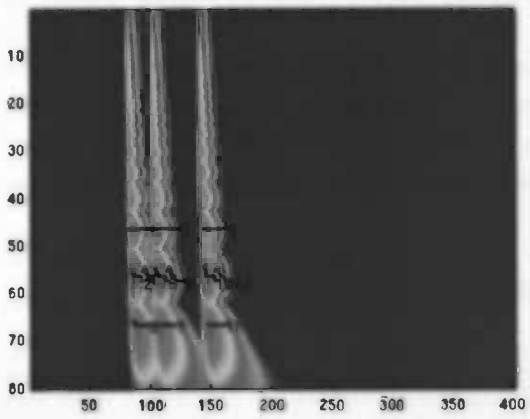



Figure 21: NSCs that were found in Artificial Signal 8 - three pulses.

One of the more noticeable faults present in the resyntheses is the shift in tone. Best noticeable in Artificial Signal 1 (the 200Hz sine), the process of locating the actual peaks is not accurate enough, causing the reproduction to be based on the segment the peak was found on, no the actual frequency. As a peak at 200 Hz lies in the segment that has an upper limit of 206 Hz, the peak of the signal is set at 206 Hz, an obvious inaccuracy. A better method to calculate the actual location of the peak would be to take the peak-segment and the two segments surrounding it, and perform (polynomial) interpolation to get a more precise approximation of the location of the peak.

Another common problem pertains to the usage of masks to determine what areas of the signal are candidates for NSC-forming. With the currently used values for the thresholds, some easily visible components are ignored. To alleviate this problem, the criteria could be loosened (which indeed would result in more inclusion), but this will be to the detriment of performance in noisy signals, because more unwanted noise will be undeservedly marked as being worthy of attention.

The resynthesis of Artificial Signal number two shows a problem with the continuation algorithms. The modulated sine of signal number two causes the system to perceive the Narrow Signal Components that were created during the scanning of the signal as separate components (each as wide as the window). This can be explained by examining the continuation algorithm (see Section 3.3.2): As it tries to extend components, it mirrors the tail (or head) of the component, and the head (or tail) of the

flections of the original are present and understandable.

NSC. When these are mirrored in signal two, the two start out displaced a bit, so that this and the modulating behavior of the nsc create enough distance to be discarded as a possible continuation. This behavior could be fixed by changing the type of mirroring (only mirroring over the vertical axis would help in this situation) of the segment that is to be mirrored, but that would only shift the problem to other signal. It would be better to use a method for determining distance between the two extended elements that takes this possible twining of modulated signals into account.

Both the missing components and unjustly extended components of signals 5 and 6 can be ascribed to steps that went wrong while building and typing the NSCs. The missing bits appear when the incline of the sweep is too steep to be included in the sinemask, and so the NSC goes untyped. Because the angle is so steep, the later stage retyping step also misses these bits of signal. The overlong components that can be seen in signals 5 and 6 and some of the natural signals are caused by NSCs following dying components for too long - see figure 22, this could be alleviated by forcing the NSC-builder to explicitly check if such errors are made.

The complete breakdown of the system while dealing with signal twelve (pulses in -13dB pink noise) is not surprising. Even for the human observer, it is difficult to pick out the exact location of the pulses without knowing where they are supposed to be. That the system will pick out random bits to add to the signal calls for a better noise recognition and elimination stage.

The added higher level post-processing steps help diminish the number of errors, but cannot truly compensate for non-optimal NSC-forming or wrong thresholds in mask creation. If masking and initial NSC forming are good enough, the role of higher level post-processing could be confined to a minimum, and processing would be faster.

Cutting up the signal showed some difference in resynthesis between the whole signal and the cut-up one. This may be caused by a differrence in phase or result from some variation in the mask-making and peakfinding process. Lastly, as expected, the system is quite good with distinguishing tonal components in a noisy signal. Both the artificial and natural tests show only small rises in mean energy difference when noise is added.

## 5.2 Conclusions

A hypothesis-driven approach toward the separation of signal components seems to give somewhat acceptable results. Recognition of pulse-

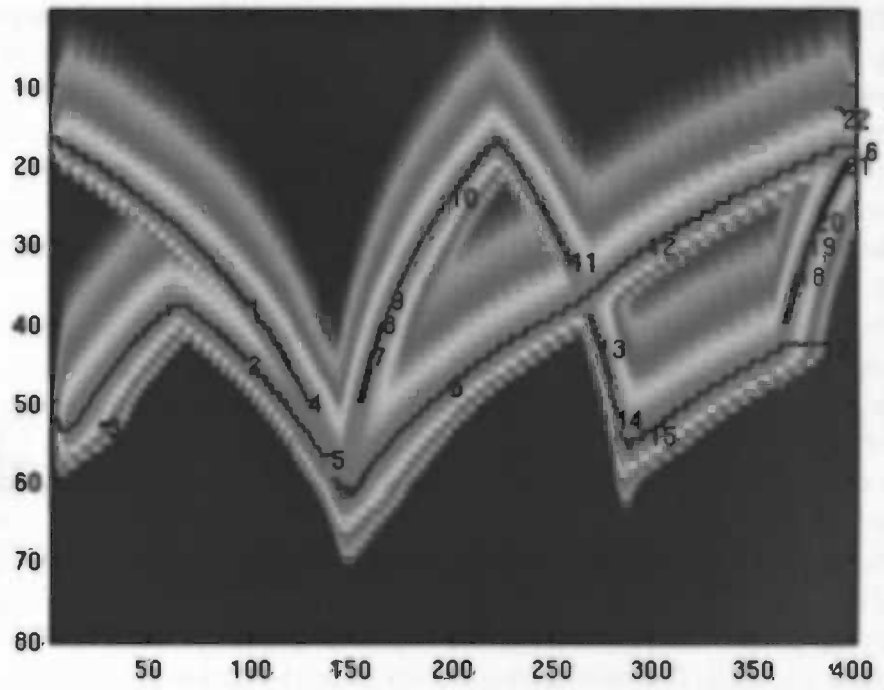



Figure 22: Artificial Signal 6 displays some problems with typing and creating NSCs

and tone-like components on a basic level can be achieved, although fine-tuning to allow for all (or even most) special cases proved to be nigh impossible. The system is quite insensitive to noise, especially when dealing with tonal components.

Adding higher level post-processing steps gives better results then just using the basic techniques - using post processing tuned for specific tasks could be a helpful addition to a component-recognising system. These higher level processes cannot truly make up for failures of the underlying system, and hence should be used only as an enhancement where neccesary, not as as an optimization step.

The system is far from perfect, there is a lot of room for improvement, especially in the lower level processes. Too much effort was spent on trying to create a good tonal component formation subsystem, to the detriment of the pulse-formation parts of the system. The system as it stands now is very slow, especially if some form of realtime processing is desirable, and it may be more advisable to try other methods to find components, maybe based more heavily on creating accurate masks.

## 5.3 Recommendations

Fixing the pulse recognition algorithms seems the most obvious step to take if future research would be conducted with this (or a similar) system. Together with tightening up the other lower level algorithms, and improving masking, this could improve results greatly. A more exact peak-position determinating method, as was mentioned in the Discussion, would be an example of an improvement of the mid-level algorithms, as would the fix that was proposed for the continuation algorithm.

Better amplitude-finding is also neccesary. The current resynthesis algorithms do not hold varying volume into account [12], which would help greatly. As the amplitude information is available, it could be used to create a sliding amplitude scale for the components. As a further improvement, a better solution to finding the width of a standard pulse at all segments (for use in the algorithm found in 3.3.4) would help with the correct identification of pulses.

Finding a way to remove the high warbling tones that occur in reproduction of natural signals would help greatly with resynthesis. The war-

[12] Varying amplitude is taken into account when using the exact-reproduction method of resynthesis, which uses the exact segments, frames and energy-values that were found when building the component. This method introduces unwanted artifacts in reproduction because it does not produce smooth sweeps, but staggers from segment to segment.

bling sounds that are produced attract undue focus, and make the rest of the signal sound more unnatural.

Depending on the application - better higher level processing, for instance rescanning of bits of signal with lower thresholds to find missing pieces of the signal. Once evidence for harmonics have been found, part of the signal where further harmonics are expected to be found could be rescanned with a lower threshold setting. Using phase-information to identify components would also be a big help in solving crossings.

Values for the thresholds of the various masks and algorithms should be based on theory or be dynamically found, based on the surrounding signal or information unveiled during processing, not just based on observation and common sense as they are now.

# 6 Bibliography

## References


[1] *Matlab website.* http://www.mathworks.com/products/matlab.

[2] Andringa, T.C. (2002). Continuity Preserving Signal Processing *Groningen*

[3] Bregman, A. S. (1990). Auditory Scene Analysis: The perceptual organisation of sound *Cambridge, MA: MIT Press. 9*

[4] Brown, G. J. (1992). Computational auditory scene analysis: A representational approach.*Ph.D. thesis CS-92-22, CS dept., Univ. of Sheffield.*

[5] Brown, G. J. and Cooke, M. (1994). Computational Auditory Scene Analysis. *Computer Speech and Language, vol. 8, no. 4, pp. 297-336.*

[6] Duifhuis, H., Hoogstraten, H.W., van Netten, S.M., Diependaal, R.J. and Bialek, W. (1985). Modelling the cochlear partition with coupled Van der Pol oscilators, *Cochlear Mechanisms: Structure, Function and Models, edited by J.W. Wilson and D.T. Kemp (Plenum, New York), pp 395-404.*

[7] D. P.W. Ellis, Prediction-Driven Computational Auditory Scene Analysis, *Ph.D. thesis, MIT Media Laboratory, Cambridge, Massachusetts, 1996.*

[8] Hyvarinen, A., Karhunen, J., Oja, E. (2001). Independent Component Analysis *John Wiley, New York.*

[9] Griffiths, T.D. and Warren, J.D. (2004). What is an auditory object? *Nat. Rev. Neurosci. 5, 887892.*

[10] Marr, D. (1982). Vision*Freeman.*

[11] Mellinger, D. K. (1991). Event formation and separation in musical sound *Ph.D. thesis, CCRMA, Stanford Univ.*

[12] Shapiro, L.G. and Stockman, G.C. (2001). Computer Vision, *Prentice-Hall.*

[13] Weintraub, M. (1985). Theory and computational model of auditory monaural sound separation, *Ph.D. thesis, Dept. of EE, Stanford University.*

# A Signals and Code

The original and regenerated signals and relevant matlab code can be found at http://pim01.nl/classification

# List of Figures