13

Automatic mixing

E. Perez-Gonzalez and J. D. Reiss

13.1 Introduction

Audio mixing is the process of applying a series of signal-processing operations to multiple audio sources and then combining them together to create a sound mixture. In this chapter we will explore signal-processing algorithms which aim to automate this process. The first automatic mixing implementations for live sound can be traced to [Dug75, Dug89]. Here the basic principles of automatic level adjustment for automatic mixing were proposed. This system is able to maintain constant gain structure regardless of the number of active microphone inputs. Decisions on time-domain gain compensation were based on RMS measurements extracted from the signal. A similar system, but based on a mechanical principle called direction-sensitive gating for automatic microphone mixing was developed by [JT84]. Other implementations of automatic microphone mixing such as [Pet78, SBI00] have been implemented, but most of these designs concentrate on level microphone management for speech applications only. Such automatic microphone mixing implementations make use of low-level features, in order to take mixing decisions. In recent years, techniques have been described related to non-real-time automatic-mixing processing algorithms. In [TR09b] an automatic monitor mixing method for deriving fader levels using optimisation is described. In [Kol08] and [BR09] a method that reconstructs mixing parameter values of each channel through analysing a target mixture was proposed. In [Ree00] a method which uses nearest-neighbour optimisation techniques to attempt to recreate expert mixing is derived. [TR09a] presented a method for automatically setting noise-gate parameters for drum recording using optimisation. Finally, some work on perception and automatic detection of frequencies which require equalisation compensation has been performed by [BLS08, BJ09].

In this chapter we will look at current automatic-mixing techniques for real-time digital signal processing that are capable of panning, correcting polarity and time offset problems, enhancing channels, compensating gain and fader level, and a simple self-equalisation algorithm. We will concentrate on automatic-mixing designs for live music mixing as opposed to speech-oriented

DAFX: Digital Audio Effects, Second Edition. Edited by Udo Zölzer.
 ISBN: 978-0-470-66599-2

microphone systems. Automatic mixing has applications to live sound and music performance, remote mixing, recording and post-production as well as real-time mixing for interactive scenes and games. We will refer to these automatic-mixing digital effect tools as AM-DAFX.

13.2 AM-DAFX

From the point of view of signal flow, a DAFX is a device that takes an un-processed input signal and outputs a processed signal. In most cases the user can control the signal-processing behaviour by manipulating a number of parameters through a graphical user interface. The aim of the user is to manipulate the signal-processing parameters in order to produce the desired transformation of the input signals. Figure 13.1 shows the standard implementation of a DAFX, where $x(n)$ is the input source and $y(n)$ is the output resulting from the signal processing.

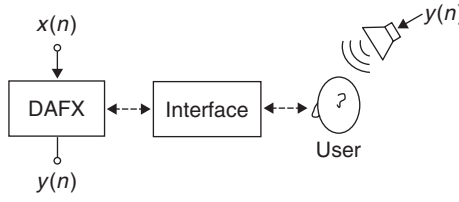

Figure 13.1 Diagram of a DAFX and user.

In an automatic-mixing context we aim to aid or replace the task normally performed by the user. In order to achieve this, some important design objectives should be performed by the AM-DAFX:

(1) The system should comply with all the technical constraints of a mixture, such as avoiding distortion and maintaining adequate dynamic range.

(2) The design should simplify complex mixing tasks while performing at a standard similar to that of an expert user.

(3) For sound-reinforcement applications the system must remain free of undesired acoustic feedback artefacts.

In the case of a system used for live performance mixing, the automatic system must avoid undesired acoustic feedback artefacts at all cost. For this reason several stability solutions have been developed, for example gain shearing [Dug75, Dug89] and self-normalisation techniques [PGR08a, PGR08b]. In most cases these techniques try to prevent acoustic feedback by ensuring a maximum electronic transfer function gain no bigger than unity. This ensures that regardless of the changes in signal-processing parameters the system remains stable.

A diagram depicting a generic AM-DAFX can be seen in Figure 13.2, where $x_e(n)$ is an external source. Our aim is to emulate the user's control parameters. An AM-DAFx is formed of two main sections: the signal-processing section and the side-chain processing portion. The signal-processing algorithm is a standard DAFX processing device and can include a user interface if the AM-DAFX is meant to give visual feedback for its actions. The analysis decision section of the automatic-mixing algorithm is what we will refer as the side-chain processing. The analysis decision-making portion of the automatic mixing tool takes audio from one or more channels together with optional external inputs and outputs the derived control data. The controlling data drives the control parameters back to the signal-processing algorithm.

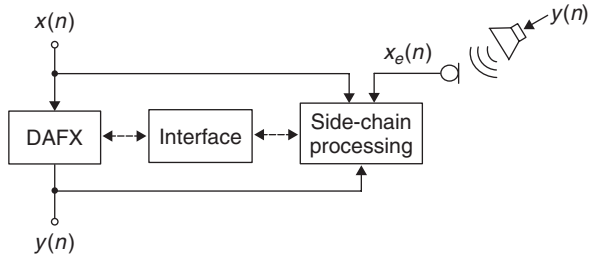

Figure 13.2 Diagram of an AM-DAFX.

The AM-DAFX described herein aims to take objective technical decisions. This is useful for improving the audio engineer's work flow and allowing him to achieve a well-balanced mix in a shorter period of time. The AM-DAFX described in this chapter are not designed to take into account any uncommon mixing practices or to be able to take subjective mixing decisions. In order to optimise the design of AM-DAFX the use of common mixing practices can be used as constraints. Given that the task normally performed by an expert user also involves perceptual considerations, perceptual rules can improve the performance of the algorithms. When combining several basic AM-DAFX tools to emulate the signal flow of a standard mixer, we can achieve a mixture in which the signal-processing flow is comparable to the one performed in a standard mixing situation.

One of the simplest AM-DAFX is the automatic gain control. In a standard digital audio mixer the head amplifier gain control is a simple multiplier which is in charge of ensuring correct analogue to digital conversion. Typically the head amplifier gain is used to scale the signal such that two technical properties are achieved:

(1) The input signal should be scaled in order not to have distortion.

(2) The signal must make optimal use of the dynamic range available.

This involves setting the gain such that the maximum peak values do not overflow the *analogue to digital converter* (ADC), while ensuring that its maximum peak value is as close as posible to 0 dB FS. Figure 13.3 shows the implementation of such an automatic-gain AM-DAFX device, where the input signal-processing section of the algorithm consists of a simple gain controller and an ADC. The side-chain processing corresponds to a simple peak feature measurement performed on the input. Every time the peak input voltage is bigger than the maximum peak value held by the ADC, the input head amplifier gain gets diminished by a step. An equivalent implementation can

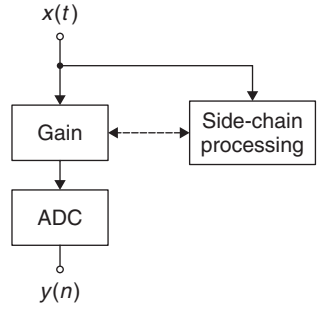

Figure 13.3 Diagram of an automatic gain AM-DAFX.

be achieved by using the overflow flag of the ADC instead of performing a direct measurement over the peak input [PGR09a]. In Figure 13.3 the AM-DAFX side chain extracts a feature from the input signal, taking a decision based on the extracted feature and then outputs control data to perform changes to the desired parameter in the signal-processing portion of the automatic-mixing algorithm. This feature extraction and decision-making process is characteristic of adaptive AM-DAFX.

13.3 Cross-adaptive AM-DAFX

DAFX architectures have been classified by their implementation [Zöl05]: filters, delays, modulators, time-segment processing, time-frequency processing, etc. Similarly, DAFX have also been classified by the perceptual attributes [ABLAV03] which they modify e.g., timbre, delay, pitch, positions or quality. Although these classifications tend to be accurate in many contexts, they are not optimal for understanding the signal-processing control architectures of some more complex effects. More recently, an adaptive digital audio effect (ADAFx) class was proposed [VZA06]. This class uses features extracted from the signals to control the signal-processing process. In terms of their adaptive properties, digital audio effects may be distinguished as follows:

(1) Direct user control: Features are not extracted from input signals so they are non-adaptive. A multi-source extension of this approach is the result of unifying the user interface, for example when linking a stereo equaliser. This provides exactly the same equalisation for the left and right channel using a single user panel. Although the user interface is unified, the output signal processing is independent of the signal content.

(2) Auto-adaptive: Control parameters are based on a feature extracted from the input source. These include, for example, auto tuning, harmonisers, simple single-channel noise gates and compressors.

(3) External-adaptive: The system takes its control processing variable from a different source to the one on which it has been applied. This is the case for ducking effects, side-chain gates and side-chain compressors.

(4) Cross-adaptive effects: Signal processing is the direct result of the analysis of the content of each individual channel with respect to the other channels. The signal processing in such devices is accomplished by inter-source dependency. It is a feedforward cross-adaptive effect if it takes its control variable from the inputs, and it is called a feedback cross-adaptive effect if it takes its control feature from the outputs.

When mixing audio, the user tends to perform signal-processing changes on a given signal source not only because of the source content, but also because there is a simultaneous need to blend it with the content of other sources, so that an overall mix balance is achieved. There is a need to be aware of the relationship between all the sources involved in the audio mixture. Thus, a cross-adaptive effect-processing architecture is ideal for automatic mixing. The general block diagram of a cross-adaptive device is depicted in Figure 13.4. Due to the importance of source inter-relationship in audio mixing for music, we can add another design objective to be performed by the AM-DAFX:

(5) The signal processing of an individual source is the result of the inter-dependent relationships between all involved sources involved.

A cross-adaptive process is characterised by the use of a multi-input multi-output (MIMO) architecture. For the sake of simplicity we will define our MIMO systems in this chapter to have the same number of input and outputs, unless stated. We will identify inputs as $x_m(n)$ and outputs

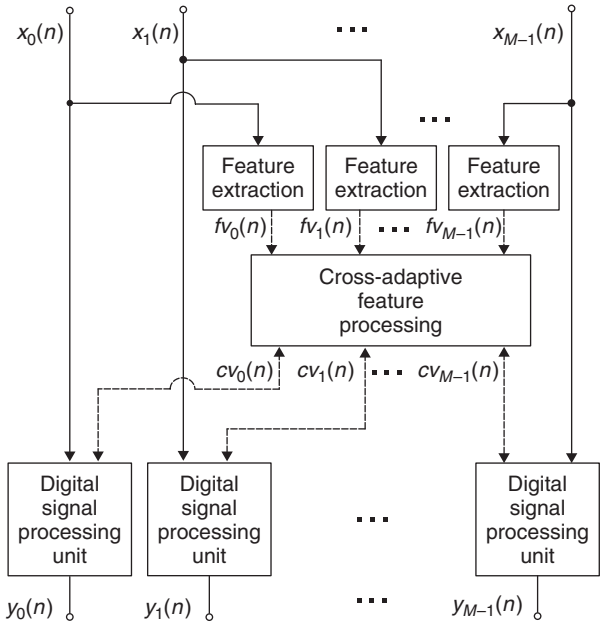

Figure 13.4 General diagram of a cross-adaptive device using side-chain processing.

as $y_m(n)$, where m has a valid range from $0, \ldots, M-1$ given that M is the maximum number of input sources involved in the signal-processing section of the AM-DAFX. External sources are denoted $x_e(n)$, as in Figure 13.2. During this chapter we will use an architecture that does not make use of feedback. Therefore the side-chain processing inputs will be taken only from the input of the signal-processing section of the AM-DAFX. In a cross-adaptive AM-DAFX the side chain consists of two main sections:

(1) A feature extraction processing section.

(2) A cross-adaptive feature processing block.

The feature extraction vector for all sources, obtained from the feature extraction processing section, will be denoted $fv_m(n)$, where n denotes the discrete time index in samples. The control data vectors for all sources, obtained from the cross-adaptive feature processing block, will be denoted as $cv_m(n)$.

13.3.1 Feature extraction for AM-DAFX

The feature extraction processing block is in charge of extracting a series of features per input channel. The ability to extract the features fast and accurately will determine the ability of the system to perform appropriately in real-time. The better the model for extracting a feature, the better the algorithm will perform. For example, if perceptual loudness is the feature to be extracted, the model of loudness chosen to extract the feature will have a direct impact on the performance of the system. According to their feature usage AM-DAFX can be in one of two forms:

(1) Accumulative: This type of AM-DAFX aim to achieve a converging data value which improves in accuracy with time in proportion to the amount and distribution of data received. The system has no need to continuously update the data control stream, which means that

the accumulative AM-DAFX can operate on systems which are performing real-time signal-processing operations, even if the feature extraction process is non-real-time. The main idea behind accumulative AM-DAFX, as implemented herein, is to obtain the probability mass function of the feature under study and use the most probable solution as the driving feature of the system. In other words we derive the mode, which corresponds to the peak value of the accumulated extracted feature.

(2) Dynamic: This type of AM-DAFX makes use of fast extractable features to drive data-control processing parameters in real-time. An example of such a dynamic system can be a system which uses an RMS feature to ride vocals against background music. Another example can be gain-sharing algorithms for controlling microphones such as the one originally implemented in [Dug75]. Dynamic AM-DAFX do not tend to converge to a static value. A compromise between dynamic and accumulative feature extraction can be achieved by using relatively small accumulative windows with weighted averages.

An important consideration to be taken into account during the feature extraction process is noise. The existence of bleed, crosstalk, self-noise and ambient noise will influence the reliability of the feature extraction. Common methods for obtaining more reliable features include averaging, coherence validation and gating. One of the most common methods used for AM-DAFX is adaptive gating, where the gating threshold adapts according to the existing noise. This method was introduced to automatic mixing applications by [Dug75, Dug89]. It requires an input noise source which is representative of the noise in the system. In the case of a live system a microphone outside of the input source capture area is a good representation of ambient noise. Therefore this microphone signal can be used to derive the adaptive threshold needed to validate the feature.

For accumulative AM-DAFX, variance threshold measures can be used to validate the accuracy of the probability mass function peak value. The choice of feature extraction model will influence the convergence time in order to achieve the desired variance. For this to work appropriately in a system that is receiving an unknown input signal, in real-time, some re-scaling operations must be undertaken. First the probability mass function must always be equal to one. Second, if the maximum dynamic range of the feature is unknown the mass probability function must be re-scaled. In such a case, the axis range should be normalised continuously to unity by dividing all received feature magnitudes by the magnitude of the maximum received input value.

An example of the effects of adaptive gating and re-scaling in an accumulative feature extraction block is shown in Figure 13.5. In this example the feature under study is loudness, which has been extracted from a musical test signal. If no re-scaling and no adaptive gating is used to optimise the loudness mass probability function, the resulting most probable feature value is always 0, as shown in Figure 13.5(a). This is because there is a large amount of low-level noise which biases the loudness measurement. A second test with re-scaling and no adaptive gating is shown in Figure 13.5(b). It can be see that although a Gaussian shape corresponding to the actual loudness can be seen, there are still a large number of data points in the lowest bin of the histogram, causing an erroneous null measurement. When adaptive gating is performed without re-scaling, Figure 13.5(c), the number of zero-bin occurrences is dramatically reduced. Finally, a test consisting of both, re-scaling and adaptive gating, is depicted in Figure 13.5(d). It can be seen that the algorithm is able to correctly identify the most probable feature value. This means that that both adaptive re-scaling and gating must be performed in order to achieve accurate extraction of the most probable feature value.

13.3.2 Cross-adaptive feature processing

The cross-adaptive processing section of the AM-DAFX is in charge of determining the inter-dependence of the input features in order to output the appropriate control data. These data controls parameters in the signal-processing section of the AM-DAFX. The obtained control parameters are usually interpolated before being sent to the signal-processing portion of the AM-DAFX. This

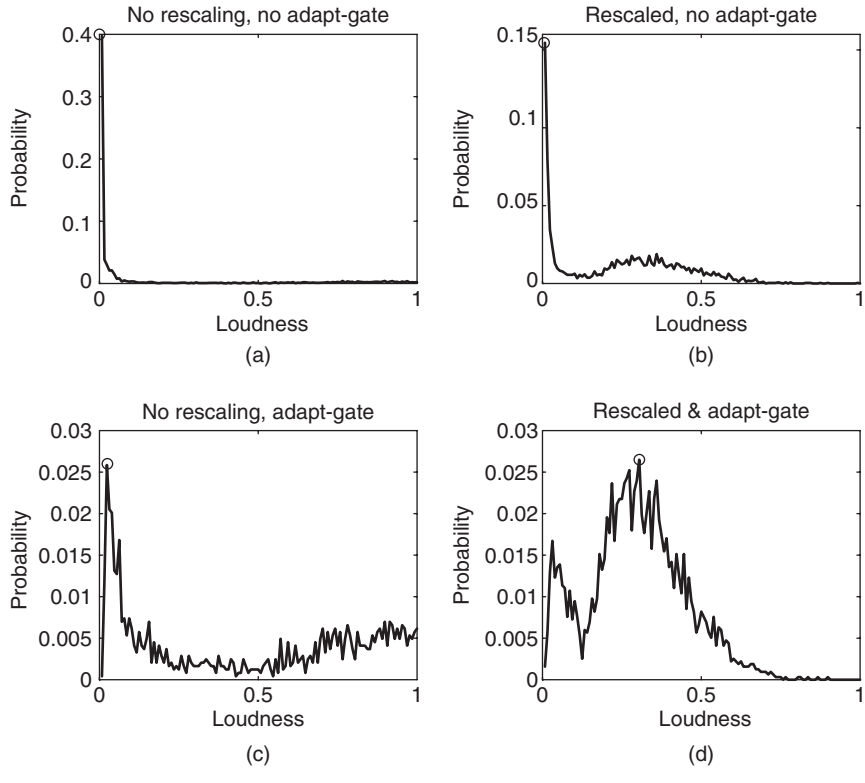

Figure 13.5 Accumulated histograms. The circular marker denotes the resulting accumulated peak loudness value [PGR09a].

can be achieved using a low pass filter that will ensure a smooth interpolation between control data points. The cross-adaptive feature processing can be implemented by a mathematical function that maps the inter-dependence between channels. In many cases constraint rules can be used to narrow the inter-dependency between channels. In order to keep the cross-adaptive processing system stability the overall gain contribution of the resulting control signals can be normalised so that the overall addition of all source control gains is equal to unity. The cross-adaptive function is unique for every design, and has to be individually derived according to the aim of the AM-DAFX.

13.4 AM-DAFX implementations

In this section we will describe the major steps needed to implement five different AM-DAFX. The resulting audio signal processing is the direct result of the analysis of the inter-relationship between input sources. The systems described are intended to be used for real-time signal processing. They can be implemented individually or cascaded together in order to implement an automatic audio mixer suitable for live audio mixing of music.

13.4.1 Source enhancer

In audio mixing a common process is to enhance a source by making it more prominent. A way to achieve enhancement is by incrementing the gain of a source $x_\mu(n)$ with respect of the other

input sources. Performing such an action could introduce acoustic feedback or distortion if the gain needed is too large. For this reason the preferred action is to lower the gain of all sources, except for the one in need of enhancement. Although performing such an action will result in a stable enhancement of the desired source, it is not an optimal solution. This is because the sources that are not spectrally related to $x_\mu(n)$ are also attenuated. In other words, if we aim to enhance a piccolo flute there should be no need to decrease a bass guitar because its spectral content shares little or no relationship with the piccolo. This type of complex frequency-dependent enhancement is familiar to audio engineers and it is what we aim to reproduce.

With this in mind we can design an AM-DAFX source enhancer whose aim is to unmask a source by applying an attenuation to the rest of the sources, relative to their spectral content. Such an enhancer should comply with the following properties:

(1) The enhancer must be able to identify the spectral inter-dependency between channels so that it only attenuates each source in proportion to its spectral relationship to $x_\mu(n)$.

(2) The gain of $x_\mu(n)$ must remain unchanged.

The signal-processing section of such a device is an attenuation multiplier per input channel, and is given by $y_m(n) = cv_m(n) \cdot x_m(n)$. Where the output $y_m(n)$ is the result of multipling the input signal $x_m(n)$ by the scaling factor $cv_m(n)$. Many other types of enhancements can be achieved by creative modifications of this architecture. For example, the same architecture presented here as a gain enhancer can be used to enhance signals by altering the amount of reverberation added to each channel in proportion to its spectral content. More on such an implementation and detailed references can be found in [PGR08c]. The side-chain processing architecture of such an enhancer will be detailed next.

Feature extraction implementation

The feature extraction algorithm we aim to design takes a digital audio input, $x_m(n)$, and outputs a data signal, $fv_m(n)$, representative of the spectral content of the input source. In order to ensure a clean acquisition we must use a data-validating algorithm. In this particular implementation we make use of an adaptive-gate algorithm. Therefore we will need the aid of an external input, $x_e(n)$, in order to derive the adaptive-gate threshold. This external input is usually taken from a measurement microphone placed in an area that is representative of the ambient noise. Such a feature extraction system is represented by $fv_m(n) = f(x_m(n), x_e(n))$. Given that $fv_m(n)$ must provide us an indication of the spectral content of $x_m(n)$, a system based on spectral decomposition is appropriate, Figure 13.6.

Therefore we can decompose the input $x_m(n)$ after it has been processed by the adaptive gate, $xg_m(n)$ and process it using a filter bank. The filter bank is comprised of K filters with a transfer function $h_k(n)$, where k has a valid range from $0, \ldots, K-1$. In order to give equal opportunity for input sources to be classified as unique the filter bank can be designed to have $K = M$, where M corresponds to the maximum number of sources involved in the AM-DAFX and has a valid range from $0, \ldots, M-1$. This also avoids having many sources clustered in the same spectral area. Once $xg_m(n)$ is decomposed into spectral bands we calculate the *root mean square* (RMS) of each k band, $\text{RMS}(h_{k,m})$. Then we identify the RMS element that has the maximum RMS magnitude, $\max(\text{RMS}(h_{0\ldots k,m}))$, and we store it in order to derive the probability mass function. We produce a histogram using the probability mass function $k = \max(\text{RMS}(h_{0\ldots k,m}(n)))$. The x axis of the histogram represents frequency bins, from $0, \ldots, K-1$, and the y axis corresponds to the probability of the signal $x_m(n)$, to be characterised as having spectral content dominated by a given frequency bin. Given that we are calculating a probability function we must normalise all elements inside the probability mass function so that the overall addition of probabilities per bin is equal to one. This is achieved by continuously dividing the number of occurrences per bin by the total number of elements received. Finally, the maximum peak of the probability mass

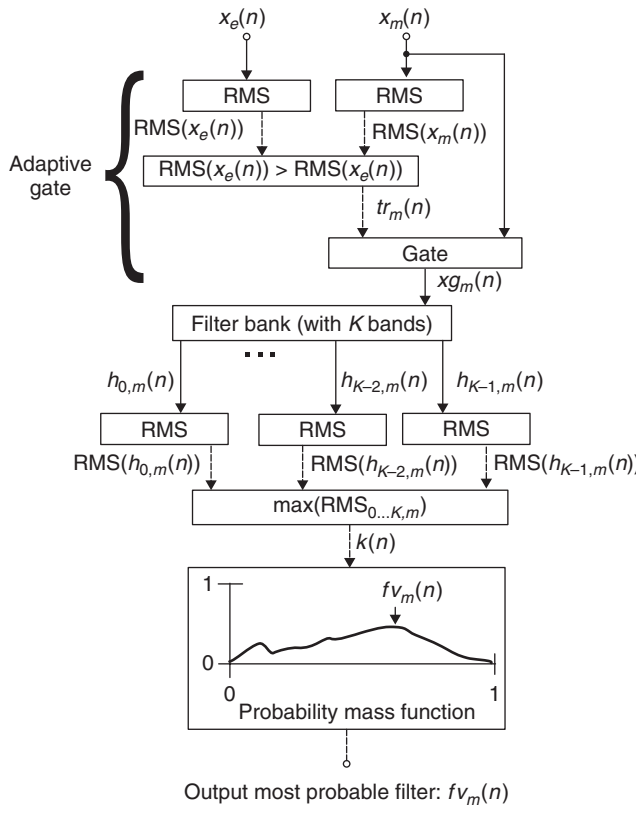

Figure 13.6 Block diagram of a feature extraction algorithm for spectral decomposition of a channel.

function must be found. This peak corresponds to the most probable spectral decomposition band $fv_m(n)$ that characterises the $x_m(n)$ under study. Under the feature extraction schema stated, $fv_m(n)$ corresponds to the most probable filter identification value and has a valid range from $0, \ldots, K-1$. Such a feature extraction process must be implemented for all M sources.

Cross-adaptive feature processing

Now that we have extracted a feature that represents the spectral content of the input signal $x_m(n)$, a function that maps the inter-relationship between the spectral content of the sources must be found. Given a source to be enhanced, $x_\mu(n)$, we must find a continuous function whose minima are located at $fv_\mu(n) = k$. This function should increase for filter values away from $fv_\mu(n)$. The final implementation should make use of the following parameters:

- User parameters
  - μ: This is the main parameter that states which of the sources is the one to be enhanced.
  - G: Is the maximum attenuation. Any source that shares the same spectral classification as $x_\mu(n)$ will have an attenuation equal to G.
  - Q: Is the amount of interaction the enhancement has with non-spectrally related sources.

- Non-user parameters
  - M: This corresponds to the total number of sources involved in cross-adaptive processing.
  - m: This tells the algorithm which source is being processed, thus getting the proper attenuation level which corresponds to a given source.

An ideal candidate to map the spectral inter-relation between channels is an inverted Gaussian. Such a function is a continuous smooth function and its minima can be made to be located at $fv_\mu(n)$. Therefore we can derive

$$fg_m(n) = \frac{1}{Q\sqrt{2\pi}} \mathrm{e}^{\frac{-(fr_m(n)-\mu(n))^2}{2Q^2}}, \tag{13.1}$$

where $fg_m(n)$ is a Gaussian function, $fr_m(n)$ is the frequency bin and $\mu(n)$ is the position on the frequency axis where the maximum inflection of the Gaussian function is located. In order achieve maximum attenuation at $fv_\mu(n)$ we normalise and invert Equation 13.1. The user controllable G will also be included as a multiplier, so that it scales the inverted Gaussian mapping function. $a_m(n)$ is the inter-source-dependent mapping function given by

$$a_m(n) = \left| \left(G \left(\frac{fg_m(n)}{\frac{1}{Q\sqrt{2\pi}}} \right) \right) - 1 \right|. \tag{13.2}$$

For each source $x_m(n)$ we must find the assigned attenuation value $a_m(n)$. Since $K = M$ we must normalise $fv_m(n)$ with respect to M in order to obtain the correct value for $fr_m(n)$. Such normalisation is given by

$$fr_m(n) = \left(\frac{2}{M-1} \left(fv_m(n) - 1 \right) \right) - 1. \tag{13.3}$$

We require that the minima of the mapping function must be centred at $fv_\mu(n) = k$, and the algorithm has K filters with M channels. So we must normalise $fv_\mu(n))$ with respect to M,

$$\mu(n) = \left(\frac{2}{M-1} \left(fv_\mu(n) - 1 \right) \right) - 1. \tag{13.4}$$

Recall that our objective is to enhance $x_\mu(n)$ with respect to the rest of sources. So we must maintain the gain of $x_\mu(n)$ unchanged. This is expressed by

$$cv_m(n) = \begin{cases} 1 & m = \mu \\ a_m(n) & m \neq \mu. \end{cases} \tag{13.5}$$

The cross-adaptive implementation of such an AM-DAFX is depicted in Figure 13.7. Since our approach only applies gain attenuations, it introduces no phase distortion.

The final mixture is now

$$mix(n) = \sum_{m=0}^{M-1} cv_m(n) \cdot x_m(n), \tag{13.6}$$

where $mix(n)$ is the overall mix after applying the cross-adaptive processing. The control attenuation for source m is given by $cv_m(n)$, where $cv_m(n) = 1$ for $m = \mu$. The attenuation control parameter $cv_m(n)$ varies with respect to its spectral content relationship Q, with a maximum attenuation equal to G for all sources in the same spectral category as $x_\mu(n)$.

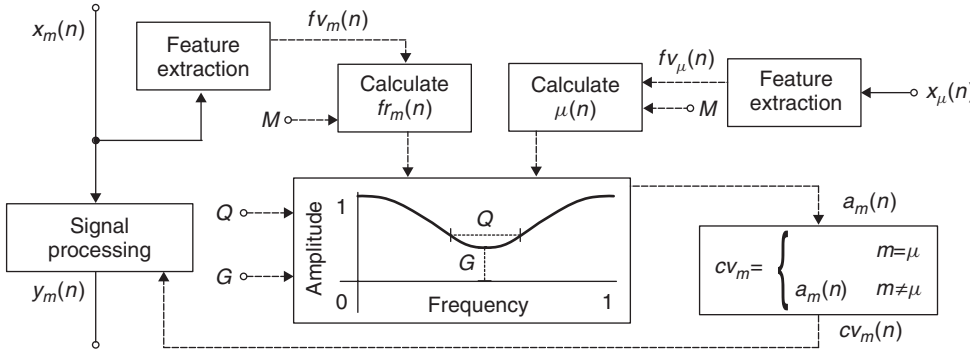

Figure 13.7 Block diagram of an enhancer.

13.4.2 Panner

Stereo panning aims to transform a set of monaural signals into a two-channel signal in a pseudo-stereo field. Many methods and panning laws have been proposed, one of the most common being the sine cosine panning law. In stereo panning the ratio at which the source power has been spread between the left and the right channels determines its position. The AM-DAFX panner aims to create a satisfactory stereo mix from multi-channel audio. The proposed implementation does not take into account knowledge of physical source locations or make use of any type of contextual or visual aids. The implementation down-mixes M input sources and converts them into a two-channel mix, $y_L(n)$ and $y_R(n)$. The algorithm attempts to minimise spectral masking by allocating related source spectra to different panning space positions. The AM-DAFX also makes used of constrained rules, psychoacoustic principles and priority criteria to determine the panning positions. For this panning implementation the source inputs $x_m(n)$ have a valid range from $0, \ldots, M-1$ and the filter bank has a total of K filters $0, \ldots, K-1$. In order to achieve this we can apply the following panning rules:

(1) Psychoacoustic principle: Do not pan a source if its energy is concentrated in a very low-frequency bin. This is because sources with very low-frequency content cannot be perceived as being panned. Not panning these sources will also give the advantage of splitting low-frequency sources evenly between the left and right channels.

(2) Maintain left to right relative balance: Sources with the same spectral categorisation should be spread evenly amongst the available panning space. In order to give all input sources the same opportunity of being classified as having a different spectral content from each other, we will make the amount of filters in the feature extraction block equal to the number of inputs, $K = M$.

(3) Channel priority: This is the subjective importance given by the user to each input source. The higher the priority given by the user the higher the likelihood that the source will remain un-panned.

(4) Constrained rules: It is accepted common practice not to wide pan the sources. For this reason the user should be able to specify the maximum width of the panning space.

Further information can be found in [PGR07, PGR10]. The feature extraction of the proposed AM-DAFX panner is based on accumulative spectral decomposition, in a similar way to the automatic enhancer shown in Figure 13.6. Therefore we can proceed directly to a description of the cross-adaptive feature processing of the AM-DAFX panner.

Cross-adaptive feature processing

Given that we have classified all $x_m(n)$ inputs according to their spectral content, a set of cross-adaptive processing functions may be established in order to determine the relationships between the spectral content of each source and its panning position. Since the algorithm aims to improve intelligibility by separating sources that have similar spectral content, while maintaining stereo balance, we must design an algorithm that makes use of the following parameters:

- User parameters
  - U_m: The user priority ordering of the sources established by the user, that determines which source has more subjective importance.
  - W: The panning width scales the separation between panned sources in order to set the maximum width of the panning space.
- Non-user parameters
  - k: The filter category is used to determine if the spectral content of a source has enough high-frequency content in order for it to be panned
  - R_m: The total repetitions per classification is the number of sources in the same feature category.

In order to assign a panning position per source we must be able to identify the total number of sources in the same feature category as $x_m(n)$, denoted as $R_m(n)$, and the relationship between the user priority, $U_m(n)$, and its spectral classification $fv_m(n)$, denoted as $P_m(n)$. We can then calculate the panning position of a source based on the obtained parameters $R_m(n)$ and $P_m(n)$.

Equation 13.7 is used to obtain the total number of classification repetitions, R_m, due to other signals having the same k classification, given the initial condition $R_0 = 0$.

$$R_m = \sum_{j=1}^{M} R_{j-1} + \begin{cases} 1 & fv_m(n) = fv_{j-1}(n) \\ 0 & fv_m(n) \neq fv_{j-1}(n) \end{cases} . \tag{13.7}$$

Now we proceed to calculate the relationship between the user-assigned priority of a source $U_m(n)$ and its spectral classification $fv_m(n)$, we will refer to this relationship as P_m. The user-assigned priority U_m has a unique value from $0, \ldots, M-1$, the smaller the magnitude of U_m, the higher the priority. The assigned priority due to being a member of the same spectral classification, P_m, has a valid range from 1 to its corresponding R_m. The lower the value taken by P_m the lower the probability of the source of being widely panned. P_m is calculated by

$$P_m = \quad |\{U_i : fv_i(n) = fv_m(n)\} \cap \{U_i : U_i \leq U_m\}| \quad \text{for} \quad i = \{0...M-1\}, \tag{13.8}$$

where the modulus of the intersection of the two sets, $\{U_i : fv_i(n) = fv_m(n)\}$ and $\{U_i : U_i \leq U_m\}$ gives us the rank position, which corresponds to the value taken by P_m.

Given R_m and P_m, we can relate them in order to obtain the panning control parameter with

$$cv_m(n) = \begin{cases} 1/2 & R_m = 1 \\ W + \left[(1-2W)\,\frac{R_m - P_m - 1}{2(R_m - 1)}\right] & P_m + R_m \; is \; odd \\ W + \left[(1-2W)\,\frac{R_m + P_m - 2}{2(R_m - 1)}\right] & P_m + R_m \; is \; even, R_m \neq 1, \end{cases} \tag{13.9}$$

and by evaluating R_m and P_m the assigned panning position can be derived. The panning position $cv_m(n)$ has a valid control range from 0 to 1, where 0 means fully panned left, 0.5 means centred

and 1 means panned fully right. The panning width limit, W, can go from wide panning $W = 0$ to mono $W = 0.5$.

Finally, based on the principle that we should not pan a source if its spectral category is too low, we set $cv_m(n)$ to be centred if the spectral category of the input source $fv_m(n)$ is less than a psychoacoustically established threshold tr_{ps}, (set to 200 Hz in [PGR07, PGR10]). This can be implemented by using

$$cv_m(n) = \begin{cases} 1/2 & fv_m(n) \leq tr_{ps} \\ cv_m(n) & fv_m(n) > tr_{ps} \end{cases} . \tag{13.10}$$

Given that we intend to mix all input sources by combining them in their respective stereo summing buss, the final mixture is given by

$$y_L(n) = \sum_{m=0}^{M-1} \sin(cv_m(n)\pi/2) \cdot x_m(n) \tag{13.11}$$

$$y_R(n) = \sum_{m=0}^{M-1} \cos(cv_m(n)\pi/2) \cdot x_m(n). \tag{13.12}$$

Where $y_L(n)$ and $y_R(n)$ correspond to the automatically panned stereo output of the mixing device, $cv_m(n)$ is the panning factor and $x_m(n)$ represents the input signals. Such an AM-DAFX panner implementation has been depicted in Figure 13.8.

13.4.3 Faders

In order to achieve a balanced audio mixture, careful scaling of the input signals must be achieved. The most common interface for achieving such a scaling is sliding potentiometers, commonly known as *faders*. Automatic fader mixers are probably the oldest form of automatic mixing, although most designs are only suitable for speech applications and are not intended for musical use. In [Dug75, Dug89] a dynamic automatic mixing implementation for music is presented. This implementation uses RMS measurements for determining the relationship between levels in order to decide the balance between microphones. One of the problems with automatic mixing implementations that use only low-level features, is that they do not take into account any perceptual attributes of the signal. Therefore a balanced RMS mixture might not be perceptually balanced. In the implementation presented here we aim to obtain a perceptual balance of all input signals. We assume that a mixture in which per-channel loudness tends to the overall average loudness is a well-balanced mixture with optimal inter-channel intelligibility.

We will use accumulative loudness measures in order to determine the perceptual level of the input signals. The main idea behind the proposed implementation is to give each channel the same probability of being heard. That is, every signal must have the same chance of masking each other. The AM-DAFX fader implementation must adapt its gain according to the loudness relationship of the individual input signals and the overall average loudness of the mix. The end result should be to scale all input signals such that they all have equal perceptual loudness. In order to achieve this we apply the following criteria:

(1) Equal loudness probability: By scaling all input signals such that they tend to a common average probability, minimal perceptual masking can be achieved.

(2) Minimum gain changes: The algorithm should minimise gain level changes required in order to avoid the inclusion of excessive gains, this can be achieved by using the overall average loudness of the mix as a reference, given that it is a natural mid starting point.

(3) Fader limit control: There must be a mechanism for limiting the amount of maximum gain applied to the input signals. This avoids unnaturally high gain values from being introduced.

(4) Maintain system stability: The overall contribution of the control gains $cv_m(n)$ should not introduce distortion or acoustic feedback artefacts.

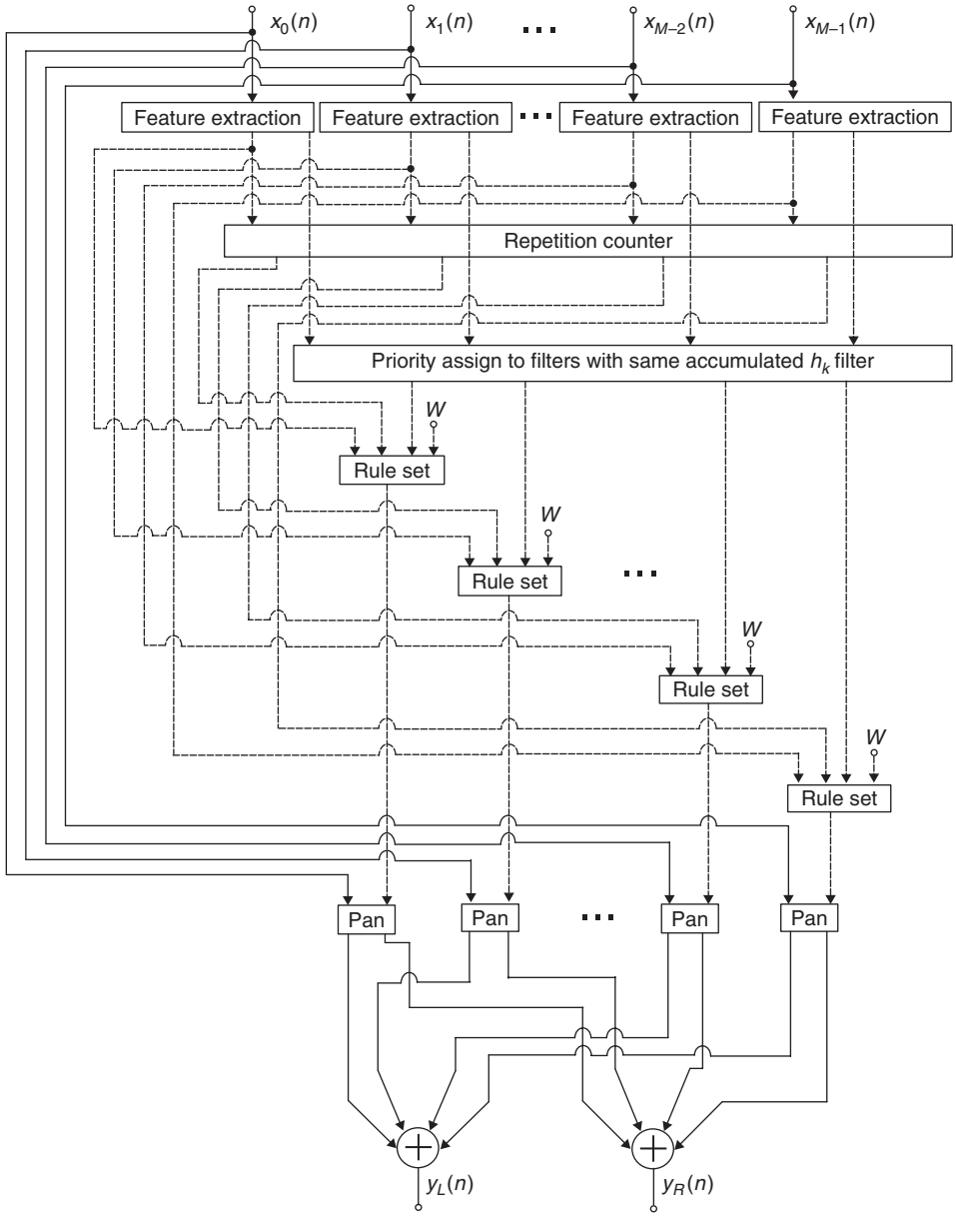

Figure 13.8 Block diagram of an automatic mixing panner [PGR07, PGR10].

The signal-processing section of such a device is an attenuation multiplier per input channel, and is given by $y_m(n) = cv_m(n) \cdot x_m(n)$, where the output $y_m(n)$ is the result of multiplying the input signal $x_m(n)$ by the scaling factor $cv_m(n)$. Further information can be found in [PGR09a]. The side-chain architecture of such perceptually driven automatic faders will be presented next.

Feature extraction implementation

Our feature extraction block will make use of adaptive gating in order to avoid unwanted noise in the measured signals. The feature extraction method requires the input signals $x_m(n)$, together with an external input $x_e(n)$ in order to derive the adaptive gating function. Such a feature extraction method has a function prototype given by $fv_m(n) = f(x_m(n), x_e(n))$.

Loudness is a perceptual attribute of sound. Therefore in order to be able to measure it we require a psychoacoustic model. The model proposed consists of perceptually weighting the input signal $x_n(n)$. Using a set of bi-quadratic filters whose coefficients are calculated so that the transfer function of the filters approximates loudness curves. Given that loudness curves change depending on the sound pressure level, a set of filter coefficients corresponding to different sound pressure levels can be calculated. All calculated coefficients are then stored in a look-up table. A measurement microphone $x_e(n)$, the same used for the adaptive gating, can be used to calculate the sound pressure $SP(n)$ and retrieve the correct set of filter coefficients from the look-up table. With the aim of obtaining a clean measurement, adaptive gating can be implemented. The loudness perceptual weighting of an input signal $xl_m(n)$ is given by

$$xl_m(n) = \frac{1}{S} \sum_{i=1}^{S} (xg_m(n) \star w(SP(n)))_i, \tag{13.13}$$

where S is an averaging constant that can be used to derive longer-term loudness measurements, as opposed to instantaneous loudness, $SP(n)$ is the sound pressure level derived from the external input and $w(SP(n))$ is the weighting filter loudness curves convolved with the gated input $xg_m(n)$.

Once we have a perceptual representation of the input given by $xl_m(n)$ we can proceed to accumulate it in order to calculate its probability mass function. Given that we are calculating a probability we must normalise the occurrences to sum to unity. Due to the fact that we do not know the perceptual dynamic range limits of the signal we must ensure that in the case where we receive a new $xl_{0\ldots M-1}(n)$ which is greater than a previous sample, we must normalise to that maximum value for all M channels. Then we proceed to find the peak of the probability mass function. This is representative of the most probable loudness value of the input under study, denoted as $fv_m(n)$. The psychoacoustic model proposed here is depicted in Figure 13.9.

Cross-adaptive feature processing

Cross-adaptive feature processing consists of mapping the perceptual loudness of each channel to its amplitude level so that, by manipulating its amplitude, we can achieve the desired loudness level. Given that we are aiming to achieve an average loudness value $l(n)$, we must increase the loudness of the channels below this average and decrease the channels above this average. The average loudness $l(n)$ is obtained as the arithmetic mean of $fv_m(n)$ for all channels.

Since we want a system in which we have a multiplier $cva_m(n)$ such that we scale the input $x_m(n)$ in order to achieve a desired average loudness $l(n)$ its function can be approximated by $Hl_m(n) = l(n)/(cva_m(n)x_m(n))$. Therefore we derive the desired gain parameter as follows:

$$cva_m(n) = \frac{l(n)}{Hl_m(n)x_m(n)}, \tag{13.14}$$

where $cva_m(n)$ is the control gain variable in order to achieve the target average loudness $l(n)$ and $Hl_m(n)$ is the function of the desired system. Given that our feature extraction block has

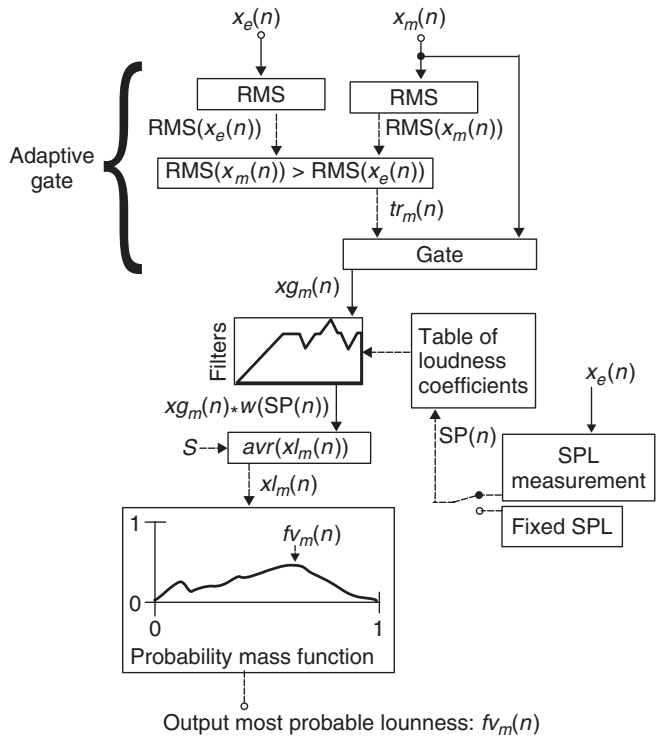

Figure 13.9 Block diagram of an automatic gain fader system.

a function $Hl_m(n) = fv_m(n)/x_m(n)$ we can say that the term $Hl_m(n)x_m(n)$ in Equation 13.14 is equal to $fv_m(n)$. Therefore $cva_m(n) = l(n)/fv_m(n)$. In most cases $cva_m(n)$ will represent a physical fader which has limited maximum gain. In practical applications $cva_m(n)$ has physical limitations. Gain rescaling must be performed in order not to exceed the system limits. This can be achieved by rescaling all channels to have an added gain contribution equal to unity, were unity is the the upper limit of the mixing system. This also ensures that the $cva_m(n)$ values stay below their physical limits and that all the faders perform an attenuation function instead of incrementing gain. This ensures the system does not introduce any undesired distortion or acoustic feedback. The normalisation process can be performed by

$$cv_m(n) = \frac{cva_m(n)}{\sum_{m=0}^{M-1} cva_m(n)}, \tag{13.15}$$

where $cv_m(n)$ is the normalised control gain value that drives the signal-processing section of the AM-DAFX in order to achieve equi-probable average loudness over all channels in the mix. The final mixture is given by Equation 13.6, where $mix(n)$ is the overall mix after applying the cross-adaptive processing. The control attenuation for every source m is given by $cv_m(n)$. The overall system block diagram of the automatic-mixing fader implementation is been shown in Figure 13.10; most cross-adaptive effects that use adaptive gating fit a similar block diagram.

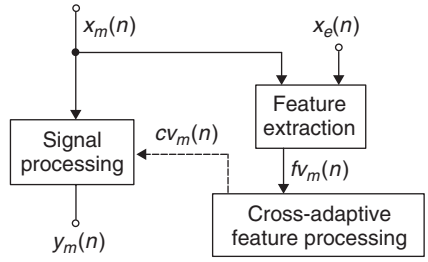

Figure 13.10 Overall block diagram of an automatic gain fader system.

M-file 13.1 (Automatic-Mixing-Framework.m)

```
function Automatic_Mixing_Framework()
% Author: E. Perez-Gonzalez, J. Reiss
% function Automatic_Mixing_Framework()

%---AUDIO INPUT for 8 Mono Files, where x{m} is the input to channel m.
[x{1},Fs]=wavread('x1.wav'); %Read file
% ...
[x{8},Fs]=wavread('x8.wav'); %Read file

%---RECORD FILE BEFORE AUTOMIXING
monoInputSum = 0;
for m=1:length(x) %Mono summing buss
    monoInputSum=monoInputSum + x{1};
end
monoInputSum=monoInputSum *.125; %Mono summing buss scaling
monoInputSumStereo=(repmat(monoInputSum*(1/sqrt(2)),1,2));%Split to Stereo
wavwrite(monoInputSumStereo,Fs,'preAutoMixSum.wav');

%---SIDE CHAIN
tr=0.002;  %%Fixed Threshold
[cv]=LoudnessSideChain_at_Fs44100(x,tr); %Side Chain

%---PROCESSING
[yL,yR]=LoudnessProcessing(x,cv); %Fader Gain

%---RECORD AUDIO OUTPUT
wavwrite([yL yR],Fs,'postAutoMixSum.wav'); %Record file after automixing

%================================================================
function [cv]=LoudnessSideChain_at_Fs44100(x,tr)
%% LOUDNESS SIDE CHAIN FUNCTION %%%
cv=[0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5]; %Initial value

%--Noise removal
for m = 1:length(x)
    xg{m}=x{m}(x{m}>tr); %Gate
end

%---Obtain feature
for m=1:length(x)
```

```
    [xg{m}]=Loudness95dB_at_Fs44100(xg{m});
    clear peakA;
end

%---Accumulative feature processing
for m=1:length(x)
    max_vector(m)= max(xg{m});
end
[max_xg,argmax_xg]=max(max_vector);

for m=1:length(x)
    xg{m}=xg{m}/max_xg; %normalize
end

figure(1); %Figure showing accumulated loudness values per channel
for m=1:length(x)
    subplot(2,4,m)
    [maxhist,maxhist_arg]=max(hist(xg{m}));%Calc. max and maxarg of hist
    [num,histout]=hist(xg{m});%Calculate histogram
    bar(histout,num)%Plot histogram
    axis([0 1 0 maxhist+1])
    hold on;
    %Calculate most probable loudness per channel
    fv(m)=(maxhist_arg*(max(xg{m})+min(xg{m})))/length(hist(xg{m}));
    plot (fv(m),maxhist,'ro')%Plot most probable loudness
    hold off;
    clear maxhist maxhist_arg num histout xg{m};
end

%---CROSS ADAPTIVE PROCESSING
l=mean(fv); %obtain average Loudness
for m=1:length(x)
    cva(m)=l/fv(m); %compensate for average loudness
end

%---Unity gain normalisation to maintain system stability
cvasum=sum(cva); %obtain total non-nomalized
for m=1:length(x)
    cv(m)=cva(m)/cvasum; %normalize for cvasum
end

%Print Loudness, control variables and gain
Feature_Loudness=[fv(1) fv(2) fv(3) fv(4) fv(5) fv(6) fv(7) fv(8)]
Control_variables=[cv(1) cv(2) cv(3) cv(4) cv(5) cv(6) cv(7) cv(8)]
Overal_gain=sum(cv) %overal gain equals 1

%=================================================================
function [yL,yR]=LoudnessProcessing(x,cv)
%---AUDIO OUTPUT for 8 Mono Files, where y{m} is the output to channel m.
%---Fader GAIN PROCESSING
for m=1:length(x)
    y{m}=x{m}*cv(m);
    clear x{m}
end;

%---Split mono results to stereo for channels 1 to 8
yL=0; %Left summing bus initialisation
```

```
yR=0; %Right summing bus initialisation

for m=1:length(y)
    yL=yL + y{m}*(1/sqrt(2)); %Scale to split mono to stereo
    yR=yR + y{m}*(1/sqrt(2)); %Scale to split mono to stereo
    clear y{m};
end

%================================================================
function [out]=Loudness95dB_at_Fs44100(in)%% LOUDNESS FEATURE EXTRACTION
%---Biquad Filter no.1 HPF
B = [1.176506 -2.353012 1.176506]; A = [1 -1.960601 0.961086];
in= filter(B,A,in);

%---Biquad Filter no.2 Peak Filter
B = [0.951539 -1.746297 0.845694]; A = [1 -1.746297 0.797233];
in= filter(B,A,in);

%---Biquad Filter no.3 Peak Filter
B = [1.032534 -1.42493 0.601922]; A = [1 -1.42493 0.634455];
in= filter(B,A,in);

%---Biquad Filter no.4 Peak Filter
B = [0.546949 -0.189981 0.349394]; A = [1 -0.189981 -0.103657];
in= filter(B,A,in);

%---Peak averaging
S=20000; %Frame size for peak averaging
cumin=[zeros(S,1); cumsum(in)];
avin=(cumin((S+1):end)-(cumin(1:(end-S))))/S; % Calculate running average
clear cumin;
Six = (S+1):S:(length(avin));% Times at wich peak amp will be returned
peakA=nan(size(Six));% Create vector holding peaks
for i=1:length(Six)% Calculete peak average
    Si = Six(i);
    peakA(i)=max(abs(avin((Si-S):Si)));%Output peak averaging
end

out=peakA;
```

13.4.4 Equaliser

In order to achieve a spectrally balanced mix, a careful perceptual balancing of the spectral content of each channel is needed. Equalisation per channel is not only done because of the individual properties of the signal, but also because it needs blending with other channels in the mix. In order to achieve a spectrally balanced mix we will employ a cross-adaptive architecture to relate the perceptual loudness of the equalisation bands amongst channels.

Even when we have achieved overall equal loudness in the mixture, as in the previous subsections of this chapter, it is apparent that some frequency ranges appear to have significant spectral masking. For this reason a multi-band implementation of the AM-DAFX gain fader algorithm has been implemented. This algorithm should not only comply with achieving equally balanced loudness per channel, but should simultaneously ensure that there is equal loudness per channel

for all equalisation bands. We will apply a signal-processing section per m channel consisting of a graphic equaliser with K bands, using filters described by the transfer function $hq_{k,m}$. By definition, a graphic equaliser has its Q and cut-off frequency points fixed, therefore we will only manipulate the gain parameter for each band. For implementing an AM-DAFX equalisation system we comply with the following design constraints:

(1) Equal loudness probability per band: All the input signals involved should tend to the same average loudness per band.

(2) Minimum gain changes: The algorithm should perform the most optimal gain changes, therefore it should increment or decrement band gain from a natural starting point such as the overall average of the mix.

(3) Overall equal loudness probability: The system must simultaneously achieve equal loudness per band and full bandwidth equal loudness.

The signal-processing section of such a device is

$$y_m(n) = \sum_{k=0}^{K-1} cv_{k,m}(n) \left[hq_{k,m}(n) \star x_m(n) \right], \tag{13.16}$$

where $cv_{k,m}$ is the desired control vector that drives the gain of each k band of the equaliser given a channel m. Further details can be found in [PGR09b]. The side-chain architecture of such perceptually driven automatic equalisers will be detailed next.

Feature extraction implementation

We use an adaptive gating at the input of our feature extraction implementation. The system prototype is described by $fv_{k,m}(n) = f(x_e(n), x_m(n))$, where $x_m(n)$ denotes the side-chain inputs, $x_e(n)$ is the external input used to derive the adaptive threshold and $fv_{k,m}$ is the feature vector describing each k band for every channel m.

We start the feature extraction process by designing a spectral decomposition bank whose filters $h_{k,m}$ match the cut-off frequencies of the filters $hq_{k,m}$ used in the signal-processing equalisation section.

For performing multi-band adaptive gating the system takes each of the spectrally decomposed bands of $x_m(n)$ and $x_e(n)$ and outputs a clean version of each of the bands of $x_m(n)$ denoted as $xg_{k,m}(n)$.

The loudness model per band is the same as presented in the previous subsection, except for the fact that it is performed per decomposition band. This means that there are KM loudness weightings overall. The equation for performing the weighting is given by Equation 13.17,

$$xl_{k,m}(n) = \frac{1}{S} \sum_{i=1}^{S} \left(xg_{k,m}(n) \star w(SP(n)) \right)_i, \tag{13.17}$$

where $w(SP(n))$ is the loudness weighting that changes as a function of the measured sound pressure level $SP(n)$ derived from the external input $x_e(n)$. Given that all the gated and spectrally decomposed bands $xg_{k,m}(n)$ are convolved with the loudness weighting, then $xl_{k,m}(n)$ is a representation of the loudness of the input signal $x_m(n)$. S is the averaging constant used to determine longer-term loudness, as opposed to instantaneous loudness.

The same accumulative process as the one presented in the previous subsection is used per spectral band. In the case of a spectral band having a loudness level greater than the previous loudness value, all bands in all channels, should be renormalised to the new maximum value. The resulting peak value of each probability mass function is an accurate representation of the most

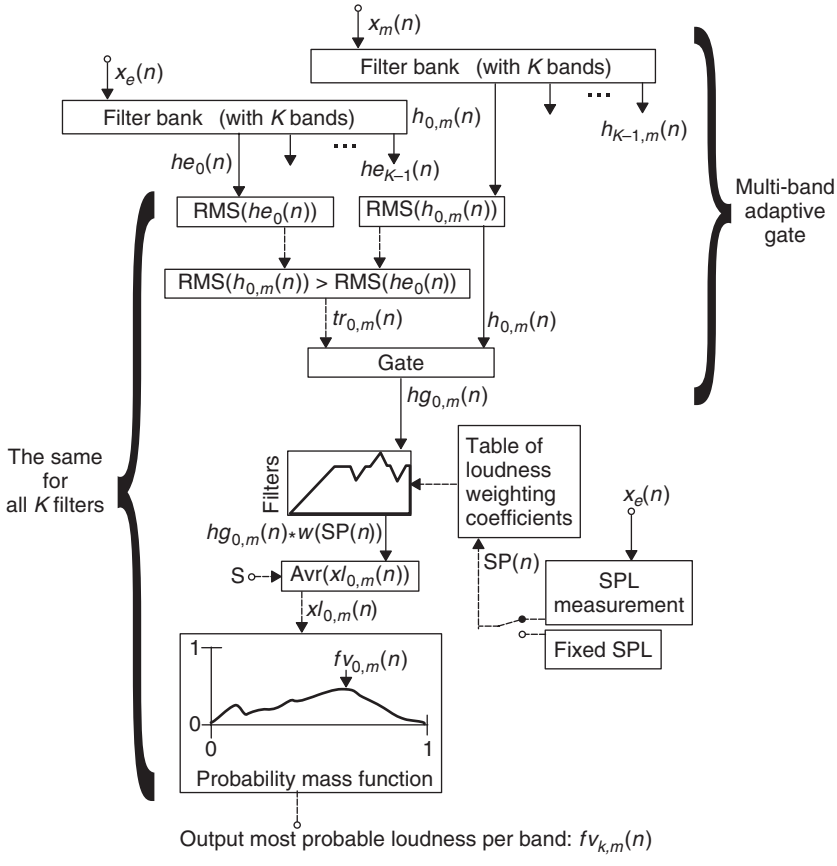

Figure 13.11 Block diagram of an automatic equaliser.

probable loudness per band of a given input and is denoted by $fv_{k,m}(n)$. The block diagram of this multi-band loudness feature extraction method is shown in Figure 13.11.

Cross-adaptive feature processing

Cross-adaptive feature processing consists of mapping the perceptual loudness of each spectral band to its amplitude level so that, by manipulating its gain per band level, we can achieve a desired target loudness level. We aim to achieve an average loudness value $l(n)$, therefore we must decrease the loudness of the equalisation bands for signals above this average and increase the band gain for signals below this average. This results in a system in which we have a multiplier $cva_{k,m}(n)$ per band, such that we scale the input bands $x_{k,m}(n)$ in order to achieve a desired average loudness $l(n)$. The function of the system can be approximated by $Hl_{k,m}(n) = l(n)/(cv_{k,m}(n)x_{k,m}(n))$, where the control vector $cv_{k,m}(n)$ is given by

$$cv_{k,m}(n) = \frac{l(n)}{Hl_{k,m}(n)x_{k,m}(n)}, \tag{13.18}$$

where $cv_{k,m}(n)$ is the control gain variable per band used to achieve the target average loudness $l(n)$ and $Hl_{k,m}(n)$ is the function of the desired system. The feature extraction block has a function

$Hl_{k,m}(n) = fv_{k,m}(n)/x_{k,m}(n)$, so $cv_{k,m}(n) = l(n)/fv_{k,m}(n)$. The target overall average loudness $l(n)$ is

$$l(n) = \sum_{m=0}^{M-1} \left(\sum_{k=0}^{K-1} fv_{k,m}(n)/K \right) /M, \tag{13.19}$$

where $l(n)$ is the average of all $fv_{k,m}$ for all k bands and m channels.

The overall system block diagram is given by Figure 13.10, where the spectral band index k must be added to $fv_m(n)$ and $cv_m(n)$ in order to accommodate the feature vector $fv_{k,m}(n)$ and the control vector $cv_{k,m}(n)$.

The final mixture is

$$mix(n) = \sum_{m=0}^{M-1} \sum_{k=0}^{K-1} cv_{k,m}(n) \left[hq_{k,m}(n) \star x_m(n) \right]. \tag{13.20}$$

Where $mix(n)$ is the overall mix after applying the cross-adaptive processing. The control attenuation for every m source is given by $cv_m(n)$.

The algorithm presented here is a simple equalisation algorithm that only takes into account gain modification per band based on perceptual loudness features. This implementation could benefit from also implementing automatic centre-frequency assignment and automatic filter Q control. One of the areas of automatic mixing which is more underdeveloped is automatic equalisation and there is great scope for further research on this subject within the DAFX community, [Ree00] and [BLS08, BJ09].

13.4.5 Polarity and time offset correction

In music mixing it is a common practice to use more than one signal for a single source. This is the case for using multiple microphones to record a single instrument. A dry signal direct from an electric instrument may be mixed with an acoustic representation of it, such as an electric guitar signal added to the signal of a microphoned guitar amplifier. Paths with different latencies may be summed, such as when adding an unprocessed signal to a processed version. In some cases using multiple representations of a source can improve the quality of the final mix, but if the signals are correlated to each other artefacts in the form of destructive interference can be introduced. The destructive interference can be present in the form of cancelation, due to opposite polarity, and comb filtering, due to offset between the signals. This destructive interference has a direct effect on the frequency content of the mix, introducing undesired frequency cancelations. Given that these artefacts are due to different times of arrival and inverted magnitude sign, we aim fix this by adding delay and polarity compensation. The goal is to achieve a sample accurate compensation in an automatic manner, by implementing an AM-DAFX with the following objectives:

(1) Optimise the delay path of each signal so that there is minimal destructive interference due to time offset errors.

(2) Ensure that the overall delay compensation used for all signal paths is as small as possible.

(3) Optimise polarity of each signal so that it matches the polarity of the signal with more delay offset.

The signal processing of such a device is described by $y_m(n) = cv_{pm}(n) \cdot x_m(n - cv_{\tau m}(n))$, where the polarity signal-processing multiplier is given by $cv_{pm}(n)$ and the signal-processing delay is represented by $x_m(n - cv_{\tau m}(n))$, such that, $cv_{\tau m}(n)$ is the delay in samples added to the signal $x_m(n)$. A detailed implementation of the automatic polarity and time offset correction can be found in [PGR08d]. The side-chain processing architecture of such a device is presented here.

Feature extraction implementation

The transfer function of a system is the Fourier transform of the impulse response of the system. The transfer function can be computed by dividing the Fourier transform of the output of the system by the Fourier transform of the input of the system. The impulse response can then be computed by the inverse Fourier transform.The impulse response of a system determines its dynamic characteristics. If we derive the impulse response of a reference signal with respect to another, given that they are correlated, we can determine the delay between them. The polarity of the maxima of the resultant impulse response can be used to determine the polarity relationship between the two signals with a common source.

In this implementation, $x_\mu(n)$ is denoted as the reference measurement and $x_m(n)$ as the measured signal. We can approximate the transfer function of a system by dividing the FFT of the output of the system by the FFT of its input. Therefore we aim to develop a feature extraction system that calculates the impulse response of a reference signal $x_\mu(n)$ with respect to each and every input given by $x_m(n)$, by deriving their FFTs. The system is given by a function prototype,

$$\mathbf{fv}_m(n) = f(x_\mu(n), x_m(n), tr, reset), \tag{13.21}$$

where $\mathbf{fv_m(n)}$ is the output vector containing the delay and polarity information for every $x_m(n)$ with respect to the reference channel $x_\mu(n)$. This type of transfer function measurement has the advantage of being independent of the input source content. On the other hand, noise added to the system can alter the measurement accuracy in an adverse manner. Therefore great care is taken to ensure that the delay estimation measurements are robust against noise.

The feature processing starts by assigning an arbitrary reference channel $x_\mu(n)$ where μ can take any arbitrary assignment, from $0, \ldots, M-1$. The inputs and reference signals are weighted by $w_{HN}(n)$, a Hanning window, in order to reduce FFT artefacts. In order to ensure clean valid measurement, the signals $x_\mu(n)$ and $x_m(n)$ must be gated with respect to the RMS amplitude of $x_\mu(n)$, according to a threshold tr. For the sake of simplicity tr is considered to be a constant, but adaptive gating techniques can also be used. The resulting gated outputs will be expressed as $xg_\mu(n)$ and $xg_m(n)$. Once the reference signal $x_\mu(n)$ and its inputs signals, $x_m(n)$, have been gated and windowed, we can proceed to apply an FFT transform to each. Their FFTs are expressed by $X_m(k) = \text{FFT}[w_{HN}(n) \cdot xg_m(n)]$ and $X_\mu(k) = \text{FFT}[w_{HN}(n) \cdot xg_\mu(n)$, and therefore we can approximate the transfer function of the reference channel against an input signal,

$$Ha_m(k) = \frac{X_m(k)}{X_\mu(k)}. \tag{13.22}$$

In order to make the feature extraction more robust to external system noise we can apply a series of techniques. We start by aiming to obtain an unbiased transfer function [Mey92] by obtaining the auto-spectrum of the measured signal, $X_{mm}(k) = X_m(k)X_m(k)^*$, and the cross-spectrum of the reference signal, $X_{m\mu}(k) = X_m(k)X_\mu(k)^*$, in order to obtain the equivalent cross-transfer function

$$H_m(k) = \frac{X_{mm}(k)}{X_{m\mu}(k)}. \tag{13.23}$$

Thus when the measurement is contaminated by noise, the transfer function may be improved, since the noise is averaged out by the determination of the cross-function of the system.

The transfer-function measurement can also be made more resilient to noise by performing complex averaging. This is achieved by averaging its complex components frames, such that

random noise being added to the complex vector is averaged out. The vector averaging is described by

$$Hv_m(k) = \frac{1}{S}\sum_{i=1}^{S}(H_{Rm}(k))_i + j\frac{1}{S}\sum_{i=1}^{S}(H_{Im}(k))_i, \tag{13.24}$$

where S is a constant representing a number of iterations over which the frequency vectors are to be averaged. The larger the value of S, the longer the system will take to compute.

We use a well-known method for estimating the difference between the arrival times of two signals with a common source [KC76], known as the *the phase transform*, (PHAT). It has been shown that the PHAT presents a prominent peak in correspondence with the actual time delay. Consider two digital signals $x_\mu(n)$ and $x_m(n)$ with a common source, its PHAT is defined by

$$\delta_{PHATm}(n) = \text{IFFT}[Hv_m(k) \cdot |Hv_m(k)|^{-1}]. \tag{13.25}$$

Finally, we use adaptive averaging to obtain reliable and steady delay and polarity measures. When performing adaptive averaging, the number of accumulations needed in order to output a valid datum, adaptively increase or decrease in inverse proportion to the absolute magnitude of the impulse response. This method asumes that the higher the amplitude of the impulse response, the better its signal to noise ratio. This means that if the signal to noise ratio is small, more accumulations are needed before a valid time delay position is output,

$$\delta_m(n) = \frac{1}{B_m}\sum_{i=1}^{B_m}(\delta_{PHATm}(n))_i, \tag{13.26}$$

where B_m is the averaging value that adapts over time, to the absolute maxima of the impulse response

$$B_m = \text{int}\left(\frac{\alpha}{|\max(\text{abs}(\delta_{PHATm}(n)))|}\right), \tag{13.27}$$

where α has been chosen to be 2 in order to duplicate the minimum number of operations to validate the calculated delay time. Once we have a stable valid impulse response we can determine its delay with respect to the reference signal by calculating the maximum argument of the impulse response function. This is given by

$$\tau_{\mu m}(n) = \arg\max_n(\text{abs}(\delta_m(n))). \tag{13.28}$$

By evaluating the impulse response by $\tau_{\mu m}(n)$ and extracting the sign we can derive the polarity of the measured signal with respect to the measurement signal,

$$p_{\mu m}(n) = \text{sgn}(\delta_m(\tau_{\mu m}(n))). \tag{13.29}$$

$\mathbf{fv_m(n)}$ is the output vector containing the delay and polarity information for every $x_m(n)$ with respect to the reference channel $x_\mu(n)$. The feature extraction method for finding the delay and polarity between the input signals $x_m(n)$ and the references signal, $x_\mu(n)$, is depicted in Figure 13.12.

In Figure 13.12 it can be seen that an extra delay block is applied to the reference signal. This delay allows the feature extraction to see negative delays. This is useful since the initial reference signal may be selected arbitrarily, and some of the measured signals may contain a negative delay. The applied delay is N/4 samples long, where the FFT resolution is equal to N.

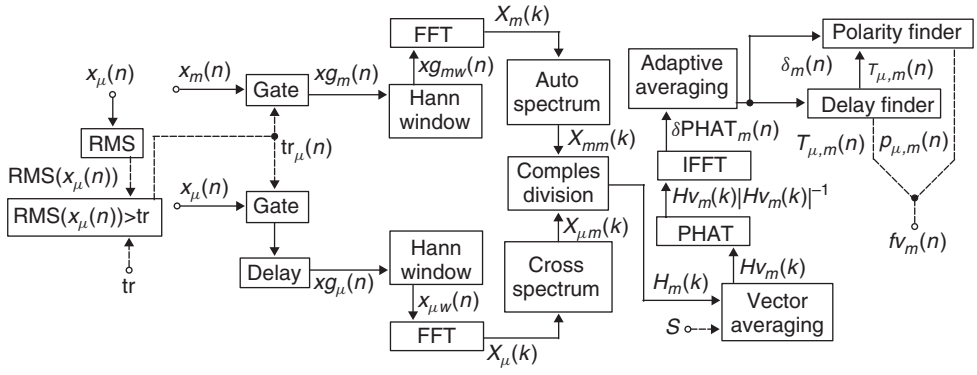

Figure 13.12 Block diagram of a feature extraction algorithm for determining delay between signals.

Cross-adaptive feature processing

If $\mathbf{fv}_{\tau\mu} \neq \max(\mathbf{fv}_{\tau(\mathbf{0}\ldots\mathbf{M-1})}(\mathbf{n}))$ we must start by reassigning $x_\mu(n)$ such that the delay added to all $x_m(n)$ is minimum. We then reset the feature extraction process and start a recalculation of the feature vector $\mathbf{fv_m}(\mathbf{n})$ given the new assignation of $x_\mu(n)$.

The components of $\mathbf{cv_m}(\mathbf{n})$ are given by

$$\mathbf{cv}_{\tau\mathbf{m}}(\mathbf{n}) = \mathbf{fv}_\mu(\mathbf{n}) - \mathbf{fv_m}(\mathbf{n}), \tag{13.30}$$

and

$$\mathbf{cv_{pm}}(\mathbf{n}) = \begin{cases} 1 & \mathbf{fv_{p}}_\mu(\mathbf{n}) = \mathbf{fv_{pm}}(\mathbf{n}) \\ -1 & \mathbf{fv_{p}}_\mu(\mathbf{n}) \neq \mathbf{fv_{pm}}(\mathbf{n}) \end{cases}, \tag{13.31}$$

where $\mathbf{cv}_{\tau m}(n)$ corresponds to the delay control data value and $\mathbf{cv_{pm}}(\mathbf{n})$ corresponds to the polarity control data value per signal. Such cross-adaptive processing implementation has been depicted in Figure 13.13.

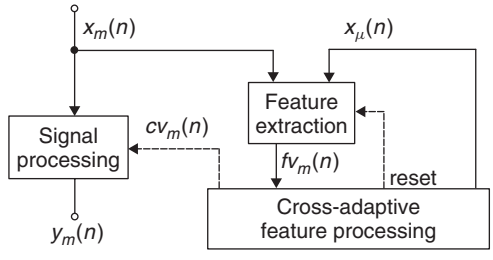

Figure 13.13 Block diagram of a cross-adaptive section of an offset polarity corrector.

The final mixture is then

$$mix(n) = \sum_{m=0}^{M-1} cv_{pm}(n) \cdot x_m(n - cv_{\tau m}(n)), \tag{13.32}$$

where $mix(n)$ is the mixture of all polarity and offset corrected input signals.

13.5 Conclusion

In this chapter we have introduced AM-DAFX; a class of digital audio effects which can do automatic mixing. As well as demonstrating a number of examples, a framework has been presented into which future AM-DAFX can be placed. The system architecture utilises cross-adaptive processing of features extracted from the input signals. Depending on the feature extraction mechanisms, the AM-DAFX have been classed as either dynamic or accumulative. Optimising the accuracy of the feature extraction mechanism can significantly improve the performance of the AM-DAFX.

The intention of AM-DAFX is to aid or replace certain tasks which are normally undertaken by the user. Only time will tell how autonomous a DAFX will become, and to what extent they will be accepted by the user. AM-DAFX is at present a growing field of research and several commercial devices based on such principles have emerged. The use of different configurations and topologies in the implementation remains to be explored. The AM-DAFX proposed here deal mainly with technical mixing constraints, and are meant to be used as a tool which allows the sound engineer to concentrate on more creative aspects of the mix. In the future, more extensive use of perceptual models may not only improve the performance of AM-DAFX, but may also allow more subjective mixing decisions to be explored. Research into these areas is, however, likely to remain controversial.

Video examples of the implementations mentioned in this chapter can be found at: http://www.elec.qmul.ac.uk/digitalmusic/automaticmixing/

References

[ABLAV03] X. Amatriain, J. Bonada, À Loscos, J. L. Arcos and V. Verfaille Content-based transformations. *Journal of New Music Research*, 32(1): 95–114, 2003.

[BJ09] J. Bitzer and J. LeBeuf. Automatic detection of salient frequencies. In *Proceedings of the 124th AES Convention*, Preprint 7704, Munich, Germany, May 2009.

[BLS08] J. Bitzer, LeBoeuf J. and Simmer U. Evaluating perception of salient frequencies: Do mixing engineers hear the same thing? In *Proceedings of the 126th AES Convention*, Preprint 7462, Amsterdam, The Netherlands, May 2008.

[BR09] D. Barchiesi and J. D. Reiss. Automatic target mixing using least-squares optimization of gains and equalization settings. In *Proceedings of the 12th International Conference on Digital Audio Effects (DAFx-09)*, Como, Italy, Sept 2009.

[Dug75] D. W. Dugan. Automatic microphone mixing. In *Journal of the Audio Engineering Society*, 23(6): 442–449, 1975.

[Dug89] D. W. Dugan. Application of automatic mixing techniques to audio consoles. In *Proceedings of the 87th AES Convention*, Preprint 2835, New York, USA, October 1989.

[JT84] S. Julstrom and T. Tichy. Direction-sensitive gating: A new approach to automatic mixing. In *Journal of the Audio Engineering Society*, 32(7/8): 490–506, 1984.

[KC76] C. Knapp and G. Carter. The generalized correlation method for estimation of time delay. In *IEEE Transactions on Acoustic, Speech and Signal Processing*, ASSP-24, pp. 320–327, 1976.

[Kol08] A. B. Kolasinski. A framework for automatic mixing using timbral similarity measures and genetic optimization. In *Proceedings of the 124th AES Convention*, Preprint 7496, Amsterdam, The Netherlands, May 2008.

[Mey92] J. Meyer. Precision transfer function measurements using program material as the excitation signal. In *Proceedings of the 11th International AES Conference: Test and Measurement*, Portland, Oregon, USA, May 1992.

[Pet78] R. W. Peters. Priority mixer control. US Patent, 4,149,032, filed 4 May 1978 and issued 10 Apr 1979.

[PGR07] E. Perez Gonzalez and J. Reiss. Automatic mixing: live downmixing stereo panner. In *Proceedings of the 7th International Conference on Digital Audio Effects (DAFx-07)*, Bordeaux, France, pp. 63–68, September 2007.

[PGR08a] E. Perez Gonzalez and J. Reiss. An automatic maximum gain normalization technique with applications to audio mixing. In *Proceedings of the 124th AES Convention*, Preprint 7830, Amsterdam, The Netherlands, May 2008.

[PGR08b] E. Perez Gonzalez and J. Reiss. Anti-feedback device. UK patent GB0808646.4, filed 13 June 2008 and published 19 November 2009.

[PGR08c] E. Perez Gonzalez and J. Reiss. Improved control for selective minimization of masking using interchannel dependency effects. In *Proceedings of the 8th International Conference on Digital Audio Effects (DAFx-08)*, Espoo, Finland, pp. 75–81, September 2008.

[PGR08d] E. Perez Gonzalez and J. Reiss. Determination and correction of individual channel time offsets for signals involved in an audio mixture. In *Proceedings of the 25th AES Convention*, Preprint 7631, San Francisco, USA, October 2008.

[PGR09a] E. Perez Gonzalez and J. Reiss. Automatic gain and fader control for live mixing. In *Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, New Paltz, New York, October 2009.

[PGR09b] E. Perez Gonzalez and J. Reiss. Automatic equalization of multi-channel audio using cross-adaptive methods. In *Proceedings of the 127th AES Convention*, Preprint 7830, New York, October 2009.

[PGR10] E. Perez Gonzalez and J. Reiss. An autonomous audio panning system for live music. In *EURASIP Journal Advances in Signal Processing, Special Issue on Digital Audio Effects*, Manuscript submitted on Jan, 2010.

[Ree00] D. Reed. A perceptual assistant to do sound equalization. In *Proceedings of the 5th International Conference on Intelligent User Interfaces*, New Orleans, Louisiana, United States, pp. 212–218, 2000.

[SBI00] Shure Brothers Inc. Chicago IL, Data sheet models FP410 portable automatic mixer. User Manual 27B8392(TB), 2000.

[TR09a] M. Terrell and J. Reiss. Automatic noise gate settings for multitrack drum recordings. In *Proceedings of the 12th International Conference on Digital Audio Effects (DAFx-09)*, Como, Italy, Sept 2009.

[TR09b] M. Terrell and J. Reiss. Automatic monitor mixing for live musical performance. In *Journal of the Audio Engineering Society*, 57(11): 927–936, 2009.

[VZA06] V. Verfaille, U. Zölzer and D. Arfib. Adaptive digital audio effects (a-dafx): A new class of sound transformations. In *IEEE Transactions On Audio, Speech, and Language Processing*, 14(5): 1817–1831, 2006.

[Zöl05] U. Zölzer. *Digital Audio Signal Processing*, 2nd edition. John Wiley & Sons, Ltd, 2005.