Universitat d'Alacant
Universidad de Alicante
Escola Politècnica Superior
Escuela Politécnica Superior

**Proyecto Fin de Carrera**
**Ingeniería Técnica de Telecomunicación**
**Sonido e Imagen**

# SÍNTESIS DE VOCES CORALES MEDIANTE CSOUND

Autor:
José Javier Valero Mas

Tutor:
José Manuel Iñesta Quereda

Septiembre, Año 2009

# Agradecimientos

Aunque pueda parecer un tópico, este proyecto no hubiera llegado a buen término sin la ayuda y el apoyo de mucha gente que me gustaría citar a continuación:

En primer lugar a José Manuel Iñesta, por haberme ofrecido la realización de este trabajo y por su gran ayuda y paciencia en el desarrollo del mismo; a David Rizo, por prestar su voz para realizar muestras de sonido y por su gran aporte de ideas para la mejora de todo el sistema; a Tomás Pérez por su colaboración en la implementación del sistema de control y a José Bernabeu por su ayuda en la preparación del sistema de grabación.

También me gustaría agradecer a toda mi familia, en especial a mis padres, por todo el apoyo prestado y la paciencia, que no es poca, "derrochada" en escucharme.

Finalmente, a mis amigos (tanto a los de siempre por su apoyo para con el *Casio*, como a los *telecos* por tres años fantásticos), de los cuales, además de su gran apoyo en todo momento, tengo la grandísima suerte de poder decir que son demasiados para citarlos uno a uno.

En realidad, lo aquí comentado no hace, para nada, justicia con la labor que de verdad han realizado cada una de estas personas, pero imagino que entenderéis lo que quiero exponer.

¡Gracias y espero que os guste!

# Prólogo

El presente trabajo parte como una propuesta de José Manuel Iñesta para la realización de un proyecto final de carrera que prosiguiera la labor iniciada años atrás, también en otro estudio de estas características, por David Rizo en el campo de la síntesis de voces corales de tipo vocal.

A modo de introducción muy básica, este proyecto tiene la doble finalidad siguiente:

- Creación de un sistema de control del sintetizador.
- Estudio de diferentes técnicas de síntesis de sonido aplicadas a la creación de fonemas de tipo consonántico para su posterior aplicación al sintetizador ya existente.

A lo largo de los diferentes capítulos se tratará de introducir al lector en la problemática de una forma más concreta, así como de mostrar y justificar cada una de las soluciones adoptadas en cada caso, aportando a la vez todos aquellos conocimientos previos requeridos para una comprensión exitosa del presente escrito.

Como último punto importante en este prólogo, es destacable el hecho de que, por y para este proyecto, se ha creado una página web alojada en un servidor del Departamento de Lenguajes y Sistemas Informáticos (DLSI) de la Universidad de Alicante, la cual alberga esta memoria y los diferentes sonidos que se pueden encontrar a lo largo de ella, además de todos los códigos fuente implementados y utilizados. La dirección de dicha página es: http://grfia.dlsi.ua.es/cm/worklines/jjvalero-pfc

# Índice general

# Índice de figuras

# Lista de algoritmos

# Capítulo 1

# Introducción al proyecto

## 1.1. Motivación

La idea de este proyecto es la de afrontar el diseño de un sintetizador que sea capaz de crear coros articulando palabras y que sea software libre y de código abierto. Para el control del mismo se hará uso del estándar MIDI: gracias a él se podrán definir las diferentes palabras a "cantar", así como la altura de las mismas.

El proyecto retomará la tarea que se inició unos años atrás con la creación de una primera versión de un sintetizador de coros que utilizaba únicamente las vocales para complementarlo con la síntesis de consonantes y así poder formar palabras. También cabe destacar que esta parte del proyecto se realizará mediante el lenguaje de programación CSound, el cual ya comentaremos más adelante.

La estructura general de realización de este proyecto es la que sigue:

- Realización de un programa que permita transponer datos de formato MIDI a formato partitura de CSound
  Con este primer punto de este proyecto se pretende automatizar la edición del archivo de partitura de CSound. Aunque todavía no se ha explicado nada acerca de esta herramienta, lo necesario para entender este punto es que este lenguaje de programación siempre necesita dos archivos de entrada (llamados ficheros orquesta y partitura, respectivamente) para generar un archivo de sonido. Mientras que el fichero de orquesta es el fichero que define los "instrumentos" que utilizaremos para el sintetizador (en nuestro caso serían las distintas voces que tendremos que hacer sonar, tanto para sonidos vocálicos como con-

sonánticos), el fichero de partitura será el que establezca qué ha de sonar y cuándo (en definitiva, lo que hace una partitura para cualquier instrumento musical).

Actualmente la construcción de estos ficheros supone un problema: mientras que el archivo de orquesta sólo es necesario redactarlo una vez (sería como el verdadero sintetizador), la parte de la partitura necesita una nueva redacción cada vez que se quiere crear algo distinto y esto supone el tener que escribir un fichero en un lenguaje de programación, lo cual acarrea las mismas dificultades de siempre: necesidad de conocer ese lenguaje, dificultad de ver lo que realmente se está haciendo, propensión a fallos...

Ante toda esta problemática se plantea la solución de utilizar el estándar MIDI y un extractor de datos del mismo para realizar así las partituras: la idea consiste en, mediante un secuenciador MIDI cualquiera, crear un archivo de este tipo y, mediante un pequeño programa, realizar la conversión de este formato de fichero al formato de partitura de CSound automáticamente.

- Síntesis de sonidos vocálicos
  En esta segunda parte del proyecto se persigue realizar la implementación de un sintetizador que sea capaz de crear de las cincos vocales del castellano en las cuatro voces que se pretende obtener: soprano – contralto – tenor – bajo. Para la creación de este sintetizador utilizaremos el lenguaje CSound. Como se ha especificado antes, esta tarea se llevará a cabo mediante la implementación del sistema de síntesis por formantes.

- Síntesis de sonidos consonánticos
  Finalmente llegamos a la parte que realmente introduce la mayor novedad para este sintetizador: la síntesis de fonemas de tipo consonántico. En este caso, se estudiarán diferentes técnicas de síntesis para escoger la que mejor se adapte a cada fonema que intentemos crear.

## 1.2. Pequeña reseña histórica

La historia de la síntesis de voz cantada tiene un origen intrínseco a la historia de la síntesis de voz hablada, por lo que se hace necesario el introducir antes ciertos elementos de ésta.

Los primeros indicios que se tienen acerca de la producción artificial de voz datan del siglo X, cuando el Papa Silvestre II construyó, como después harían Alfonso X el Sabio o Roger Bacon, una primitiva forma de cabezas parlantes capaces de crear sonido similar al habla[1]. Sin embargo, no se tiene gran información acerca de estos artilugios.

El siguiente dato importante que se tiene acerca de mecanismos de este estilo ocurre ya en el siglo XVIII de la mano de Christian Gottlieb Kratzenstein, profesor de la Universidad de Copenhague, el cual consiguió construir un primitivo sintetizador de vocales modificando tubos de órgano.

[2]Por esa misma época, el científico austríaco Wolfgang von Kempelen, conocido sobretodo por su supuesto autómata jugador de ajedrez llamado *El Turco*[3], construye el primer sintetizador de habla con una máquina que intentaba reproducir el sistema del habla humano. Kempelen había estudiado bastante este mecanismo con la finalidad de dar soluciones de tipo terapéutico a gente con deficiencias en su aparato fonador. Todos estos estudios, junto con los planos de su máquina, se encuentran publicados en el libro *Mechanismus der menschlichen Sprache nebst der Beschreibung seiner sprechenden Maschine*[4], libro que estableció a este científico como uno de los primeros investigadores en el campo de la fonética experimental.

Figura 1.1: Fotografía de la máquina de Kempelen

[1]http://www.lpi.tel.uva.es/~nacho/docencia/ing_ond_1/trabajos_05_06/io3/public_html/
/historia/historiasintesis.html

[2]http://www.ling.su.se/staff/hartmut/kemplne.htm

[3]En realidad no era un autómata sino que, internamente, era controlado por un operario. Este artefacto llegó a "jugar" con gente como Charles Babbage, Napoleón Bonaparte o Benjamin Franklin.

[4]Mecanismo del habla humana con descripción de su máquina parlante.

Posteriormente fueron apareciendo máquinas del estilo de la ideada y construida por Kempelen, como por ejemplo el artilugio *Tecnefon* del español Severino Pérez, pero que no incluían ninguna mejora fundamental. Sin embargo, a mediados del siglo XIX, Joseph Faber construye otro sintetizador de voz, llamado *Euphonia*[5], basado en el del científico austríaco pero con una gran mejora, al menos desde el punto de vista del presente proyecto: esta máquina, además de hablar, era capaz de cantar.

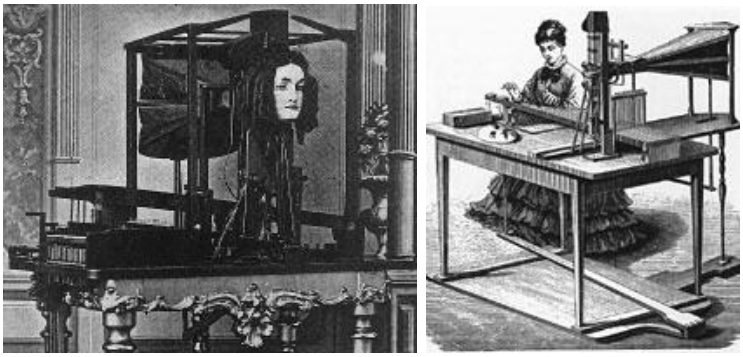

Figura 1.2: Diversas imágenes del aparato *Euphonia*

Aproximadamente un siglo después, a mediados del siglo XX, con la ayuda de la gran evolución que había tenido lugar en la electrónica, Homer Dudley[6] construyó el primer sintetizador de voz totalmente electrónico, llamado *VODER*.

Figura 1.3: Operaria utilizando el *VODER* de Dudley

[5]http://www.alpoma.net/tecob/?p=280
[6]http://120years.net/machines/vocoder/

A pesar de que hasta entonces la síntesis de voz se había entendido como un pasatiempo, fue más o menos a partir de la aparición del artilugio de Dudley que se comenzó a estudiar el tratamiento de la señal de voz, con el objetivo principal de reducir el ancho de banda necesario para su transmisión y para la seguridad en las comunicaciones, lo que originó la aparición de los llamados *VOCODER*[7], aparatos capaces de codificar la voz[8]. Dentro de este apartado cabe destacar la labor de los Laboratorios Bell a finales de los años 50, por la creación de gran cantidad de máquinas de síntesis de voz, capaces también de realizar voz cantada.

Finalmente, con la aparición y popularización de los ordenadores, el procesado de señales digitales adquiere gran importancia, llevando consigo el desarrollo de algoritmos de tratamiento de voz como, por ejemplo, la Codificación Lineal Predictiva (LPC) o el vocoder en fase (Phase-Vocoder) [10], que actualmente tanto uso tienen en la compresión de voz y en la síntesis de la misma.

## 1.3. Software para la síntesis de voz cantada

En este apartado previo al desarrollo del trabajo citaremos algunos ejemplos de software ya existentes que sean capaces de llevar a cabo la tarea que nosotros pretendemos realizar: sintetizar coros, aunque no sea de la manera que vamos a utilizar en la implementación de este proyecto.

### 1.3.1. Flinger

Es un sintetizador de voz cantada que crea la señal de voz a partir de un fichero MIDI de entrada. Está basado en el programa de síntesis de voz hablada *Festival* (de hecho, *Flinger* es el acrónimo de *Festival Singer*).

Este software, el cual es libre y gratis, es desarrollado por la Universidad de Oregon (Oregon Graduate Institute of Science and Technology Research Center), concretamente en su grupo de Tecnologías del Lenguaje Hablado (Spoken Language Technologies).

Se puede encontrar más información sobre este sintetizador en la web oficial: http://cslu.cse.ogi.edu/tts/flinger/

[7] VOice CODER.

[8] http://en.wikipedia.org/wiki/Vocoder

### 1.3.2. Vocaloid

Es un programa para la síntesis de voz cantada distribuido por Yamaha (software propietario). Ha sido desarrollado principalmente por la Universidad Pompeu Fabra (Barcelona) y se basa en la síntesis concatenativa, aunque también es capaz de crear efectos de audio (vibrato, pitch bend...) para el procesado de ésta vía software.

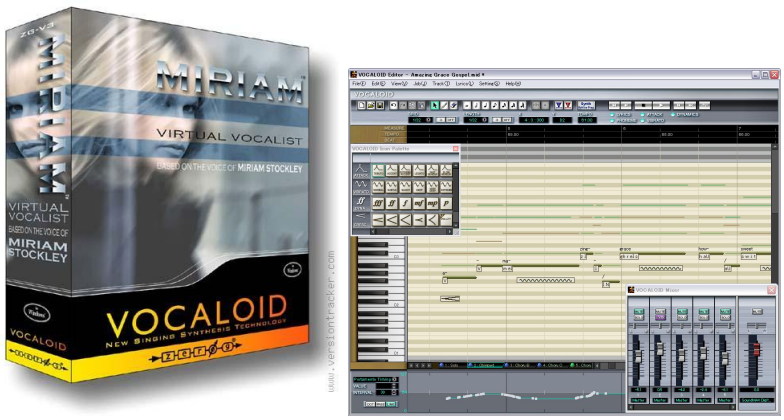


Figura 1.4: Detalle de una de las diferentes versiones de *Vocaloid*

También es posible encontrar referencias a un software llamado *Daisy* debido a que éste era el nombre del sintetizador en el que está basada esta tecnología, también desarrollado por la Universidad Pompeu Fabra.

### 1.3.3. Cantor

Sintetizador de voz cantada de la empresa VirSyn. Consta de dos editores para crear cada sonido: *Voice*, el sintetizador que emula las cuerdas vocales y la respiración, y *Phonemes*, simulación de unos filtros variables para cambiar la envolvente de la seãl generada y así crear otros sonidos.

En la edición 2.0, además de la edición manual de partitura que se ofrecía en versiones anteriores, se ofrece mayor funcionalidad MIDI, pensado para el control del sintetizador en tiempo real.

Es software propietario y la versión Cantor 2.0 vale alrededor de 299 €[9].

[9]Este precio es a fecha de agosto de 2009 en la tienda on-line Thomann.

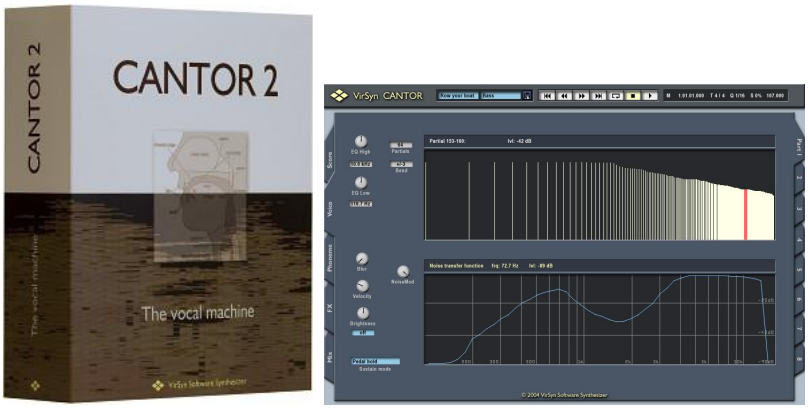


Figura 1.5: Detalle del sintetizador software *Cantor*

### 1.3.4. Lyricos

Es el predecesor del software *Flinger* antes comentado. Sus características más importantes son las que siguen:

- Síntesis por medio de suma de ondas sinusoidales[10].
- Consta de controles musicales como puede ser el vibrato.
- Crear un archivo de sonido a partir de una secuencia MIDI.

Como en el caso de *Flinger*, estamos hablando de software libre y gratuito.

### 1.3.5. Burcas

Software para la síntesis de la voz cantada en Sueco. Las características de este programa son las que siguen:

- Utiliza la técnica de síntesis por concatenación (basado en el algoritmo MBROLA[11]).
- Emplea el control por MIDI: a partir de los ficheros MIDI extrae información acerca de la altura y la duración de cada nota.
- Está escrito en lenguaje Perl.

[10]También llamado síntesis aditiva.

[11]El Proyecto MBROLA, de naturaleza libre y gratuita, busca la creación de un sintetizador de habla humana mediante concatenación de difonos. Su página oficial es http://tcts.fpms.ac.be/synthesis/mbrola.html

- Utiliza un fichero de texto para guardar las palabras que ha de sintetizar.

### 1.3.6. SPASM

Sintetizador de canto mediante el modelado físico del sistema de voz humano. Creado por Perry R. Cook para su tesis doctoral [12].

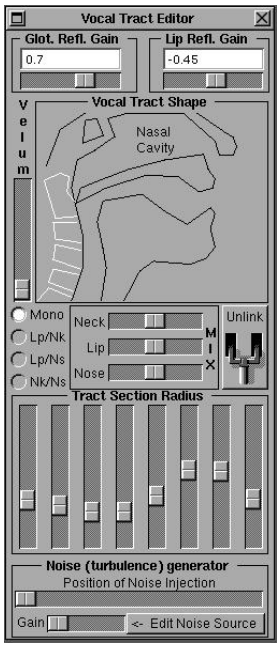


Figura 1.6: Detalle del software *SPASM*

### 1.3.7. Vocimerus

Es un producto, distribuido por la empresa NUSofting, que implementa síntesis de voces corales mediante una modificación del algoritmo VOSIM[13] que crea señales con formantes sin necesidad de filtros.

Cabe destacar que esta aplicación no es como las que se han especificado antes, sino que se trata de un programa de los llamados plug-in de audio para

[12]La página web de esta tesis es http://www.cs.princeton.edu/~prc/SingingSynth.html

[13]Acrónimo de VOice SIMulator. Es un algoritmo que se basa en modelado espectral para conseguir la señal deseada, en este caso voz cantada.

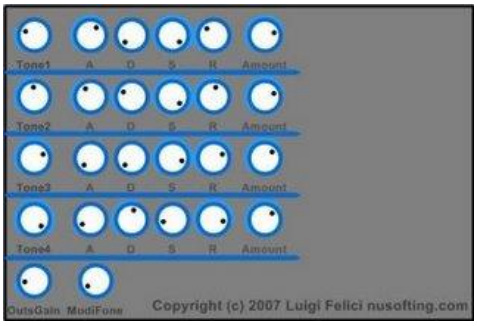


Figura 1.7: Detalle del software *Vocimerus*

utilizar en diferentes programas de edición, concretamente en este caso, bajo el estándar VST.

Por último destacar que su distribución, aunque no de código libre, sí que es gratuita.

### 1.3.8. Elvis

No es un sintetizador de voz cantada, pero sí que es un sistema de procesado de voz que permite modificar la señal que emite una persona al cantar para así adecuarla a la voz de otra persona, en este caso, otro cantante. El método utilizado para conseguir esto es el denominado *morph*[14]: es una técnica que permite obtener una nueva señal de audio en base a otras dos con las características deseables de ambas. Cabe destacar que este proceso utiliza la técnica de Síntesis por Modelado Espectral (SMS) desarrollada por Xavier Serra.

Este software es distribuido por Yamaha con carácter propietaria (no es libre ni gratis) y fue creado por el Grupo de Tecnología Musical (MTG) de la Universidad Pompeu Fabra. Por su característica de poder cambiar una voz a otra, este software ha sido muy demandado por los karaokes.

### 1.3.9. CHANT

Programa de síntesis de voces corales creado por el centro IRCAM de París a principios de los años 90. Es bastante destacable por ser el proyecto

[14]Abreviatura de *morphing*, técnica bastante utilizada en temática de vídeo para simular la transformación progresiva de un elemento (persona, animal, objeto...) en otro.


Figura 1.8: Detalle del software *Elvis*

que originó (de hecho se desarrolló para esto), la síntesis de Formas de Ondas Formantes, conocidad por síntesis FOF[15].

## 1.4. Antecedentes a este proyecto / Objetivos concretos

Como apunte final en este capítulo, y a modo de recordatorio sobre todo lo comentado anteriormente, vamos a resumir de una forma esquemática de dónde parte el presente proyecto y cuáles son los objetivos a conseguir para así ver claramente el camino a recorrer.

Los antecedentes a la realización de este proyecto son:

- Sintetizador con capacidad para crear cuatro voces distintas: Soprano - Contralto - Tenor - Bajo.
- Únicamente se realiza síntesis de vocales, concretamente las del idioma castellano.
- Utiliza el método de síntesis por formantes sustractiva (modelo de fuente más filtros resonantes para moldear el espectro de la señal).
- El fichero de la partitura se crea a mano.

Los objetivos concretos planteados para el siguiente proyecto son:

[15]Está técnica fue desarrollada por Xavier Rodet.

- Creación de un sistema de control que permita, en lugar de tener que redactar el fichero de partitura a mano directamente en el formato de CSound, crear una secuencia MIDI, por medio de cualquier sistema destinado a ello, para posteriormente, con nuestro sistema de control, transformar esa secuencia en nuestro fichero de partitura. Este sistema permitirá también suministrar a nuestro sintetizador ficheros MIDI que puedan descargarse de sedes webs especializadas.
- Estudio del sintetizador previo para su comprensión y su posible mejora.
- Estudio de diferentes técnicas de síntesis, con una evaluación comparativa sobre resultados, para la generación de sonidos consonánticos.
- Integración de la síntesis de consonantes al anterior sintetizador.

# Capítulo 2

# Conocimientos previos necesarios

En este segundo capítulo de la memoria se plantea la necesidad de introducir una serie de conceptos que será necesario conocer para así poder comprender correctamente el resto del proyecto. Concretamente se introducirán al lector dos herramientas básicas para este proyecto: por un lado el lenguaje de síntesis de sonido CSound, el cual, como se podrá ver después, será el encargado de crear el sonido que se escuchará, mientras que por otro lado se introducirá el estándar MIDI, herramienta clásica en la música por computador, la cual utilizaremos como herramienta de control en nuestro sintetizador. Finalmente también cabe destacar que en este capítulo se introducirán diferentes técnicas de síntesis de sonido, las cuales serán realizadas en la práctica en capítulos posteriores para escoger la que mejor resultado dé a nuestra finalidad.

## 2.1. El lenguaje CSound

El lenguaje de programación CSound, lenguaje ideado para la síntesis de sonido, surge en el Instituto Tecnológico de Massachusetts (M.I.T.) de la mano de Barry Vercoe. Toma este nombre debido a que su núcleo está programado en lenguaje C.

Se basa en una serie de ficheros que debemos compilar para obtener el resultado deseado. En este caso, son siempre dos ficheros[1] que, con la finalidad de buscar un símil musical, se conocen como:

[1]Se pueden comprimir en uno, pero nosotros mantendremos esa distinción.

- Fichero de orquesta: en este fichero definimos los instrumentos que sonarán en nuestra composición.
- Fichero de partitura: en este fichero definimos, sobretodo, cuándo sonarán los instrumentos creados en la orquesta.

Este sintetizador de sonido utiliza tablas de onda, por lo que siempre, en toda composición que se realice (entendiendo que la composición es una unión entre un fichero de orquesta y otro de partitura) se deberán incluir siempre una serie de tablas que serán las que utilicemos para crear los sonidos deseados.

Como base para la creación de sonidos, este lenguaje de programación tiene una serie de rutinas programadas, denominados *opcodes* y que se pueden entender como funciones en un lenguaje de programación. Los que sean utilizados para la realización del presente proyecto serán descritos en un anexo a la memoria.

Una vez aclaradas estas características básicas, vamos a pasar a describir, de forma más concreta, la estructura de cada fichero de los que se compone:

- Fichero de orquesta
  La estructura básica de este fichero la encontramos en la figura 2.1.

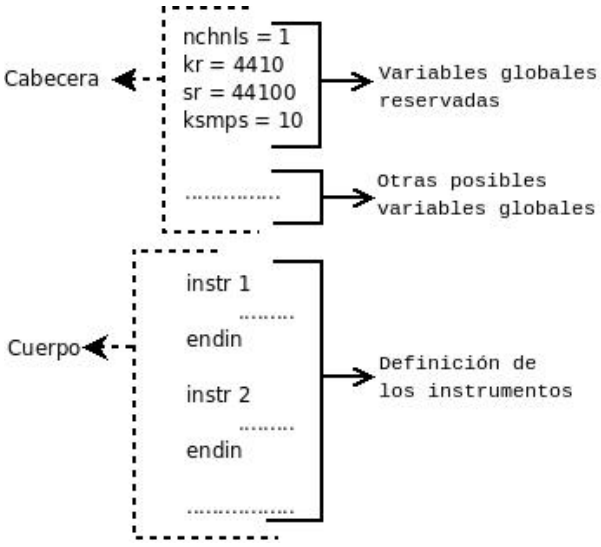


Figura 2.1: Estructura básica de un fichero de orquesta de CSound.

Vamos a describir a continuación cada una de las diferentes partes que componen este fichero:

- Variables globales reservadas
  1. **nchnls**: Hace referencia al número de canales que se pueden utilizar. Su valor por defecto es 1 y los posibles valores son:
     - 1: Indica una orquesta de tipo mono.
     - 2: Indica una orquesta de tipo estéreo.
     - 4: Indica una orquesta de tipo cuadrafónico.
  2. **sr**: Es la frecuencia de muestreo del sistema. Por defecto tiene el valor 44100 Hz.
  3. **kr**: Es la frecuencia de control. Es la frecuencia a la que se recalculan las variables de tipo control. Su valor por defecto es 4410 Hz.
  4. **ksmps:** Cantidad de muestras calculadas con unos valores de las variables de control determinados. Define cuántas muestras se calculan antes de renovar las variables de control. Su valor por defecto es 10 muestras y se puede calcular como en cociente $\frac{sr}{kr}$.

- Definición de instrumentos
  - Sintaxis de un instrumento genérico
    La sintaxis para la definición de instrumentos es bastante sencilla y siempre tiene esta forma:

    **instr** *NUMERO*
    *Definicion de instrumento (sentencias)*
    **endin**

    Como podemos observar, una declaración de instrumento siempre requiere la pareja de palabras reservadas **instr** – **endin**. Además, también es importante destacar que el instrumento siempre se identifica por un número y no por una palabra.

  - Sintaxis de una sentencia genérica
    Las sentencias que se utilizan para definir las características de un instrumento en CSound tienen esta estructura:

    *[etiqueta: ] salida opcode argumentos*[2] *[; comentarios]*

[2]Cada uno de ellos va separado por una coma del resto.

Cabe destacar que los argumentos que hemos puesto entre corchetes son opcionales, mientras que el resto son los propios de una declaración genérica con *opcodes* de CSound.

- Tipos de variables
  Las variables en CSound no se declaran de antemano, definiendo por tanto su tipo y tamaño, sino que se crean cuando son usadas por primera vez y su tipo depende de la letra con la que empiecen. Los cuatro tipos de variables existentes en este lenguaje son:
  1. **g-**: Variables globales. Han de ser de algún tipo de los que siguen, por lo que después de la letra *g* debe haber alguna de las otras tres. Toman valor al inicio de la compilación.
  2. **i-**: Variable tipo nota. Toma valor al inicio de cada una de las notas que toca el intrumento.
  3. **k-**: Variables de tipo control. Toman un nuevo valor cada *kr* veces por segundo. Cada valor sustituye al que tenía antes.
  4. **a-**: Variables de tipo audio. Toman valor *sr* veces por segundo. Son vectores (no se sustituye el valor anterior por el nuevo, sino que se almacena uno a continuación del otro).

- Fichero de partitura
  La estructura básica del fichero la encontramos en la figura 2.2.

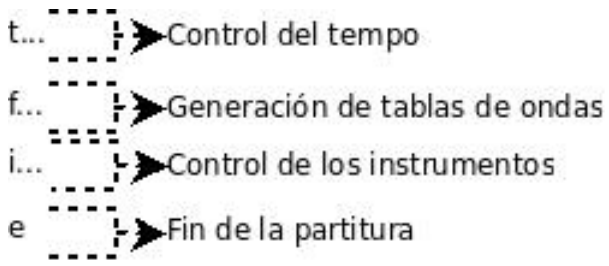


Figura 2.2: Estructura básica de un fichero de partitura de CSound.

Vamos, por tanto, a definir las distintas órdenes de partitura:

- Letra reservada **t**
  Se utiliza para el control del tempo y su estructura es la que sigue:

**t** *0 T1 tA T2 tB T3...*

siendo *0*, *tA*, *tB*... los valores de los tiempos musicales (en beats) en los que definimos el tempo y *T1*, *T2*, *T3*... los valores del tempo.

- Letra reservada **f**
  Hace referencia a las rutinas de generación de tablas (las rutinas **GEN**). La estructura de este control es:

  **f** *Número_tabla Tiempo_inicio Longitud* **Función_GEN** *Parámetros*

- Letra reservada **i**
  Es el comando dedicado a la ejecución de notas. Es el ejemplo más representativo de comunicación entre orquesta y partitura. La estructura que se define en este caso es:

  **i** *Número_instrumento Tiempo_inicio Duración Otros_parámetros*

  Para algunos campos de este comando encontramos ciertos caracteres especiales que son:

  - **.**: Toma el mismo valor que el mismo parámetro de la línea anterior.
  - <: Toma el valor que resulta de interpolar los valores que están en ese mismo campo en las líneas anterior y posterior.
  - +: Sólo se aplica a *Tiempo_inicio* y se sustituye por *Tiempo_inicio*+ *Duracion* (línea anterior).

- Letra reservada **s**
  Comando de fin de sección. No tiene una estructura como las anteriores, sino que simplemente se pone la letra.

- Letra reservada **e**
  Comando de fin de partitura. Como en el caso anterior, basta con poner la letra.

- Letra reservada **a**
  Comando de avance. Su estructura es:

  **a** *0 Tiempo_desde Tiempo_hasta*

- Letra reservada **r**
  Comando para repetición: se repite desde el punto en que se encuentre el comando hasta el fin de sección un número de veces determinado. Su estructura es:

  **r** *Número_veces*

Por último destacar que la página web principal de este lenguaje es http://www.csounds.com

## 2.2. El estándar MIDI

MIDI, acrónimo de *Musical Instrument Digital Interface*, es un protocolo digital de comunicaciones surgido con la finalidad de que los equipos musicales electrónicos pudieran comunicarse entre ellos y con los ordenadores. Es un sistema basado en mensajes, por lo que un dispositivo MIDI estará continuamente lanzando tramas de bits al medio para poder comunicarse con el otro dispositivo.

Las tramas de bits que utiliza este estándar se agrupan en los llamados bytes MIDI, cuya longitud es siempre de 10 bits, dispuestos de la siguiente forma:

0bbbbbbbb1

Como podemos observar, el primer bit de esta cadena es siempre 0 y el último es siempre 1 (necesidad de una referencia al ser una comunicación serie asíncrona[3]), por lo que en la práctica sólo quedan 8 bits puramente de datos y por ello, al hablar de cualquier tipo elemento del estándar que haga referencia a bits, siempre lo hace a esos 8 pues son lo únicos que pueden variar.

Una vez definido qué es un byte MIDI a nivel puramente electrónico (o computacional), debemos diferenciar dos tipos de bytes por su significado dentro del estándar:

- Bytes de estado: Son los bytes que establecen la órdenes que se han de ejecutar.
- Bytes de datos: Son simplemente los parámetros que se ha de pasar con los bytes de estado.

[3]A diferencia de las conexiones de tipo síncrono en las que se difencia cada trama de bits por una señal de reloj, en las conexiones asíncronas se diferencian las tramas por ciertos bits reservados para su delimitación.

A continuación definiremos más a fondo estos dos tipos de bytes que son los que conforman el estándar MIDI.

- Bytes de estado
  Los bytes de estado siempre comienzan por el valor 1 (recordemos que nos referimos únicamente a los 8 bits que pueden modificar su valor dentro de un byte MIDI), por lo que únicamente tenemos $2^7 = 128$ posibilidades. Podemos verlo de una forma más visual así:

  1eeecccc

  En esta codificación, la letra $e$ hace referencia a los estados posibles (en total serán $2^3 = 8$) y la letra $c$ al canal, concepto que definiremos a continuación, al que afectará ese estado.

  Cuando hablamos de canal MIDI nos referimos a diferentes direcciones, de tipo lógico[4], a las que podemos enviar los mensajes. De una forma más sencilla e intuitiva, podemos pensar en los canales MIDI como diferentes intrumentos que se pueden controlar desde el mismo dispositivo y cable de forma independiente [7]. Como apunte a lo aquí comentado, cabe destacar que el estándar MIDI comprende $2^4 = 16$ canales.

  Retomando en tema anterior a los canales MIDI, cabe destacar que hay tres tipos de estados en función de los parámetros que necesiten:

  - **0 Bytes de datos**: No necesitan parámetros, por lo que si les llegan los ignorarán.
  - **1 Byte de datos**: Necesitan un único parámetro para funcionar, por lo que si le llegan más asumirán que es otro nuevo.
  - **2 Bytes de datos**: Necesitan parámetros de dos en dos, por lo que si, tras una secuencia de bytes MIDI queda uno suelto, lo ignorarán.

- Bytes de datos
  Los bytes de datos siempre comienzan por el bit 0, dejando por tanto $2^7 = 128$ posibilidades, lo cual se puede ver de una forma más gráfica así:

  0bbbbbbb

[4]Se refiere a que los elementos de direccionan por medios electrónicos (bytes, estados de tensión...) pero no por medios puramente físicos (cables, ondas...).

El hecho de tener 128 valores para representar datos en definitiva quiere decir que tenemos 128 valores para codificar las notas que queramos. En el estándar MIDI se establece que la nota 60 sea el $DO_3$ (261.63 Hz), quedando la nota 0 como $DO_{-2}$ (8.176 Hz) y la nota 127 como $SOL_8$ (12,544 Hz).

En el sistema MIDI podemos distinguir diferentes tipos de mensajes, que son los que siguen:

- *Mensajes de canal*: Mensajes que se dirigen a un canal concreto.
  - **De voz**: Son los que mandan sonar al sintetizador.
  - **De modo**: Indican cómo ha de sonar el sintetizador.
- *Mensajes de sistema*: Mensajes que se dirigen al sistema en general. Al no llevar canal, se utilizar esos bits para especificar el mensaje.
  - **Comunes**: Para gestionar la reproducción de secuencias
  - **De tiempo real**: Son mensajes usados por dispositivos maestros que almacenan y reproducen secuencias de mensajes MIDI.
  - **Exclusivos**: Para intercambio de información entre dispositivos del mismo fabricante.

Por no ser totalmente necesario, además de para no extender demasiado esta introducción al estándar MIDI, se ha decidido incluir un anexo (Apéndice D) en la memoria del presente proyecto que abarca más información que la aquí recogida.

## 2.3. Clasificación de los diferentes fonemas

En este apartado vamos a realizar una clasificación de los diferentes fonemas que encontramos para las diferentes consonantes en el idioma castellano desde varios puntos de vista.

### 2.3.1. Según el modo de producción del fonema

- Oclusivos
  Consonantes caracterizadas por un bloqueo total del flujo de aire causado por una obstrucción completa que se crea cuando un articulador activo hace contacto total con un articulador pasivo.

Los fonemas que encontramos en este apartado son:

| Fonema | Ejemplo | Fonema | Ejemplo |
|---|---|---|---|
| p | Paco | b | Bueno |
| t̪ | Tocar | d̪ | Dado |
| k | Casa | g | Gato |

- Fricativos
  Consonantes que se articulan forzando el aire a través de una hendidura estrecha creada por el cierre parcial o total de tracto vocal pero sin que se interrumpa el flujo de aire.

  Los fonemas que encontramos en este apartado son:

| Fonema | Ejemplo | Fonema | Ejemplo |
|---|---|---|---|
| f | Fresco | ʒ * | Rodilla |
| ϴ | Cena | ʝ | Maya |
| S | Suma | X | Gente |
| S̻ ** | Suma | χ ** | Gente |
| ʃ * | Relleno | h * | Mes |

- Africados
  Consonantes cuya articulación incluye una fase de obstrucción total seguida de una fase de fricción.

  Los fonemas que encontramos en este apartado son:

| Fonema | Ejemplo | Fonema | Ejemplo |
|---|---|---|---|
| ʧ | Chico | ʤ | Yunque |

- Nasales
  Consonantes que se producen cuando realizamos un descenso del velo del paladar permitiendo, entonces, la entrada de aire en la cavidad nasal. Esto, desde un punto de vista de filtrado de señales, se puede modelar con ceros en la respuesta del filtro del tracto vocal.

  Los fonemas que encontramos en este apartado son:

| Fonema | Ejemplo | Fonema | Ejemplo |
|---|---|---|---|
| m | Misa | n̪ | Concha |
| ŋ | Inferior | ɲ | Año |
| n̺ ** | Ciencia | ɳ * | Mango |
| n̻ | Cuando | n | Nombre |

- Vibrantes
  Consonantes cuya articulación requiere la vibración de la lengua, sin interrupción del flujo de aire.

  Los fonemas que encontramos en este apartado son:

| Fonema | Ejemplo | Fonema | Ejemplo |
|---|---|---|---|
| ɾ | Cero | r | Tierra |

- Laterales
  Consonantes en las cuales la lengua produce un bloqueo central pero el aire pasa por uno o ambos lados de la misma hacia el exterior.

  Los fonemas que encontramos en este apartado son:

| Fonema | Ejemplo | Fonema | Ejemplo |
|---|---|---|---|
| $\underset{+}{\text{l}}$ ** | Alza | $\overset{\vee}{l}$ | Colchón |
| $\underset{\cap}{\text{l}}$ | Falda | ʎ ** | Lluvia |
| l | Lola | – | – |

- Espirantes
  Consonantes cuya producción requiere que los labios o cualquier otro articulador crea, junto a otro, una hendidura amplia por la cual el aire escapa sin causar ruido turbulento.

  Los fonemas que encontramos son los siguientes:

| Fonema | Ejemplo | Fonema | Ejemplo |
|---|---|---|---|
| $\underset{\text{T}}{\beta}$ | Lavar | ɣ | Amigo |
| ð | Cada | – | – |

### 2.3.2. Según el lugar de producción del fonema

Se refiere a la zona, dentro de nuestro sistema de creación de habla, en la cual se crea el sonido.

- Labial
  Consonantes que se articulan por la acción de los labios. Pueden ser:

  - Bilabial
    Los distintos fonemas de este tipo que encontramos son:

| Fonema | Ejemplo | Fonema | Ejemplo |
|---|---|---|---|
| p | Paco | $\overset{\beta}{\underset{\intercal}{}}$ | Lavar |
| b | Vago | m | Mano |

- Labio–dental
  Los fonemas que encontramos son:

| Fonema | Ejemplo | Fonema | Ejemplo |
|---|---|---|---|
| f | Fresco | ɱ | Inferior |

- Coronales
  Consonantes que se crean por la acción de la corona de la lengua. Pueden ser:

  - Interdental
    Los fonemas que encontramos de este tipo son:

| Fonema | Ejemplo | Fonema | Ejemplo |
|---|---|---|---|
| Θ ** | Cima | $\underset{+}{\text{l}}$ ** | Alza |
| $\overset{\eth}{\underset{\intercal}{}}$ | Cada | $\underset{+}{\text{n}}$ ** | Panza |

  - Dental
    Podemos incluir estos fonemas en esta clasificación:

| Fonema | Ejemplo | Fonema | Ejemplo |
|---|---|---|---|
| $\underset{\sqcap}{\text{t}}$ | Roto | $\underset{\sqcap}{\text{l}}$ | Saltar |
| $\underset{\sqcap}{\text{d}}$ | Mundo | $\underset{\sqcap}{\text{n}}$ | Dónde |

  - Alveolar
    A este tipo corresponden los siguientes fonemas:

| Fonema | Ejemplo | Fonema | Ejemplo |
|---|---|---|---|
| ɾ | Triste | n | Enrique |
| r | Zorro | s | Cosa |
| l | Lote | $\underset{\sqcup}{\text{s}}$ ** | Mesa |

  - Alveopalatal
    A este tipo corresponden los siguientes fonemas:

| Fonema | Ejemplo | Fonema | Ejemplo |
|---|---|---|---|
| ʃ * | Vaya | ʤ | Cónyuge |
| ʒ * | Rodilla | n̬ | Lancha |
| ʧ | Choza | l̬ | Salchicha |

- Dorsales
  Consonantes que se crean por la acción del dorso de la lengua. Pueden ser:
  - Velar

| Fonema | Ejemplo | Fonema | Ejemplo |
|---|---|---|---|
| k | Queso | ɤ | Amigo |
| g | Tango | ɳ * | Tenga |
| χ * | Ajo | – | – |

- Palatales
  Consonantes que se articulan con el dorso de la lengua elevándose hacia el paladar.

  Podemos encontrar los siguientes fonemas como pertenecientes a este tipo:

| Fonema | Ejemplo | Fonema | Ejemplo |
|---|---|---|---|
| ʝ | Bella | ɲ | Ñapa |
| ʎ * | Doncella | – | – |

- Guturales
  Consonantes que se articulan en la faringe o en la laringe.

  Encontramos los siguientes fonemas con estas características:

| Fonema | Ejemplo | Fonema | Ejemplo |
|---|---|---|---|
| h * | Mes | – | – |

### 2.3.3. Según la utilización de las cuerdas vocales

Hace referencia al uso o no de las cuerdas vocales en la producción del fonema correspondiente.

- Sonoros
  Consonantes en cuya articulación entra en juego la vibración de las cuerdas vocales.

  Podemos encontrar, con estas características, los siguientes fonemas:

| Fonema | Ejemplo | Fonema | Ejemplo |
|---|---|---|---|
| b | Sombra | d̪ | Celda |
| g | Venga | ʒ * | Mayo |
| ʤ | Inyectar | m | Campo |
| ɱ | Enfoque | n̟ ** | Encima |
| n̪ | Cuando | n | Enrique |
| n̻ | Lancha | ɲ | Campaña |
| ɳ * | Arenga | *β* | Selva |
| ð̪ | Verdad | ʝ | Arroyo |
| ɣ | Amigo | l | Balsa |
| l̟ ** | Alza | l̪ | Saltar |
| l̻ | Colchón | ʎ ** | Llama |
| ɾ | Caro | r | Raro |

- Sordos
  Consonantes en cuya articulación las cuerdas vocales no entran en vibración.

  Podemos encontrar, con estas características, los siguientes fonemas:

| Fonema | Ejemplo | Fonema | Ejemplo |
|---|---|---|---|
| p | Sopa | t̪ | Cita |
| k | Saco | f | Sofá |
| θ ** | Voz | s | Mesa |
| s̺ ** | Seco | ʃ * | Vaya |
| x | Gente | χ ** | Caja |
| h * | Lista | ʧ | Chiste |

* : Estos fonemas pertenecen a dialectos del castellano, como el dialecto caribeño o el rioplatense.
** : Estos fonemas son exclusivos del dialecto castellano.

### 2.3.4. Inclusión de fonemas en el sintetizador

A pesar de que estos son los diferentes fonemas que podemos encontrar en el idioma castellano (incluyendo dialectos del mismo), no todos tendrán cabida finalmente en nuestro sintetizador por escapar este objetivo al alcance del actual proyecto.

Nuestra prioridad serán los fonemas de tipo sonoro, en especial los nasales y los espirantes, por ser los más cercanos a las vocales ya creadas y así poder darles solución de una forma similar. Posteriormente se tratará de sintetizar los fonemas sordos que sea posible para dar un mayor rango de posibilidades al sintetizador.

Aunque de lo anteriormente comentado pueda parecer que se va a tratar de crear todos los sonidos existentes, no es así. A continuación mostramos todo aquello que no se recreará en nuestro sistema:

- Por un lado, no se tratará de dar solución a las diferentes situaciones que puede tener una consonante como, por ejemplo en los fonemas nasales, en los que la consonante *n* tiene un fonema distinto en función de las consonantes o vocales que le acompañen.
- Tampoco se tratará de dar solución a fonemas relativamente parecidos como los que pueden representar la *b* y la *v* o la *c* y la *k*, sino que se modelarán como si fueran el mismo.
- Por otro lado, no se tratará de dar solución a la situación en la que hayan dos consonantes seguidas, como puede ser situaciones del tipo *fr*, *cl*, *gl*...

Para finalizar, destacar que, aunque estos sean los objetivos iniciales, todo dependerá finalmente del resultado de las diferentes técnicas que se apliquen para dar solución al problema aquí planteado, por lo que no se puede afirmar que lo aquí descrito sea lo que finalmente se encontrará en el sintetizador, sino lo que se intentará que aparezca.

## 2.4. Métodos para la síntesis de sonido

En este apartado vamos a realizar un repaso a algunas técnicas de síntesis de sonido. La finalidad de este apartado no es realizar una gran exposición de todas las técnicas existentes, sino que únicamente se citarán las que se posteriormente se utilicen para dar solución a los problemas planteados en este proyecto.

### 2.4.1. Síntesis sustractiva

Este tipo de síntesis se basa en la idea del modelo fuente–filtro: un sonido de partida, preferiblemente de banda ancha, es procesado por un banco de filtros para crear un timbre determinado. El esquema genérico para este tipo

de síntesis es el que muestra la figura 2.3.

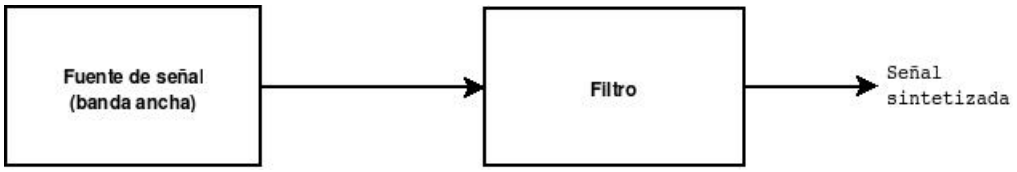


Figura 2.3: Diagrama básico de la síntesis sustractiva.

Normalmente, la fuente se modela mediante una señal de pulsos periódica con una frecuencia fundamental de repetición (produce segmentos de sonido estacionarios) o mediante ruido de tipo blanco o rosa (produce segmentos de sonido de tipo no estacionario), quedando el diagrama de bloques como se puede ver en la figura 2.4.

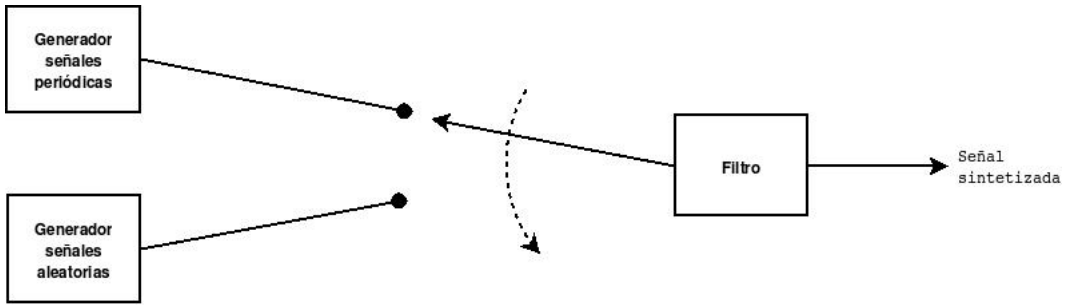


Figura 2.4: Diagrama de síntesis sustractiva con los modelos típicos de fuente de señal.

Dentro de este método hemos de distinguir dos variantes en función de los parámetros necesarios para la síntesis de la señal:

- Modelo de análisis–resíntesis
  En este modelo no seguimos una serie de reglas preestablecidas de antemano para la generación del sonido, sino que lo que hacemos es ir analizando una señal representativa del sonido que se quiere conseguir para ir extrayendo los parámetros necesarios de la misma y, posteriormente, sintetizar ese sonido utilizando esos parámetros en el modelo de síntesis.

Como ejemplo de este modelo destaca el llamado método de Codificación Lineal Predictiva (LPC) utilizado habitualmente en codificación de voz.

- Modelo de síntesis por reglas
  En este caso, la señal se crea en base a una serie de reglas ya establecidas extraídas, a priori, del sonido a imitar. Cada formante se modela con un filtro (generalmente un resonador de dos polos) y permite, en teoría, sintetizar un número infinito de sonidos. Algunos de los parámetros posibles que se utilizan como reglas pueden ser el período de la señal periódica, la amplitud de cada formante y la frecuencia a la que se encuentra, la ganancia del filtro

  Tiene dos estructuras básicas que comentaremos a continuación:

  - Estructura en cascada
    Básicamente es un conjunto de resonadores paso–banda conectados en serie. Como información de control (es decir, las reglas comentadas antes) únicamente necesita la frecuencia a la que se sitúan los formantes y no requiere información sobre el ancho de banda del filtro ni sobre la amplitud relativa de cada formante.

    El diagrama de bloques que describe esta estructura lo encontramos en la figura 2.5.


Figura 2.5: Banco de filtros tipo serie.

Este tipo de configuración es favorable para sonidos de tipo no nasal, teniendo no muy buen resultado al sintetizar fonemas de tipo fricativo u oclusivo.

- Estructura en paralelo
  Este tipo de estructura se basa en un conjunto de resonadores conectados en paralelo. Sobre cada resonador se aplica la misma

señal y, a la salida de todos, se suma la señal que produce cada uno con una precaución: las salidas de resonadores adyacentes se suman con fase opuesta para evitar antirresonancias.

Como parámetros de control, esta estructura requiere la amplitud relativa de cada formante, la frecuencia a la que se encuentra el mismo y el ancho de banda que ha de tener.

El diagrama de bloques que describe lo encontramos en la figura 2.6.

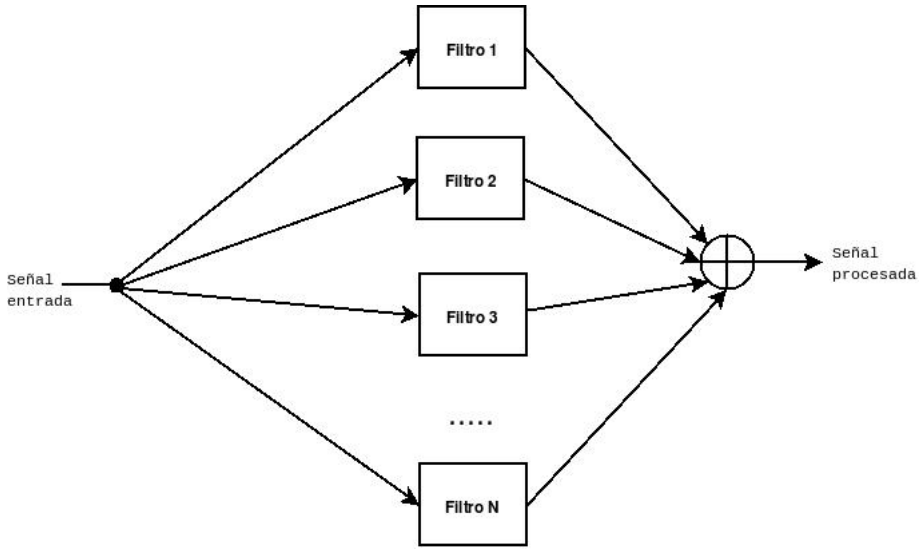


Figura 2.6: Banco de filtros tipo paralelo.

Normalmente esta configuración se utiliza para crear sonidos de tipo nasal, fricativos y oclusivos, en los cuales muestra un buen comportamiento.

- Modelo PARCAS[5]
  En sí no supone un nuevo tipo de estructura ya que en realidad es una combinación de los modelos de estructura en paralelo y en serie.

[5]Acrónimo de Paralelo–Cascada.

Este tipo de estructura intenta reunir las ventajas de la estructura paralelo y la estructura serie, llegando a crear sonidos que incluso no pueden crear ninguno de las otras dos configuraciones por separado.

Por último, destacar que en este tipo de síntesis se suele conocer a la señal a filtrar como señal excitadora o simplemente excitador por ser, en un símil electrónico, la *energía* inicial de todo el sistema. Este tipo de nomenclatura se utilizará bastante en posteriores capítulos.

### 2.4.2. Síntesis aditiva

Este tipo de síntesis, la cual se puede ver como el extremo opuesto a la anterior, es un tipo de síntesis que se basa en la suma señales simples (normalmente, ondas sinusoidales de diferente amplitud, frecuencia y/o fase) para crear ondas más complejas.

Considerando lo anteriormente dicho sobre que las señales suelen ser de tipo sinusoidal, podríamos representar una señal sintetizada por este método con la siguiente expresión:

$$s(t) = \sum_{i=1}^{L} A_i \cos(2\pi f_i t + \theta_i) \tag{2.1}$$

Cabe destacar que este método ha sido empleado en la síntesis de voz cantada obteniendo resultados que, aunque no totalmente inteligibles, consiguen dar una buena entonación a la señal por la facilidad de la variación de los parciales en el espectro de la señal ([2],[11]). También es resaltable el hecho de que esta forma de síntesis se comporta de una manera eficiente computacionalmente cuando se trata de síntesis de sonidos de tipo periódico[6] mientras que para la síntesis de señales ruidosas se necesita gran cantidad de osciladores, aumentando bastante la complejidad del sistema ([12],[13]). Para reducir la complejidad computacional en este tipo de casos se suele recurrir a una mezcla entre la síntesis aditiva y la sustractiva[7] ([13]).

[6] Si únicamente se sintetizan los parciales más destacados, se necesitarán pocos osciladores.

[7] Aunque hay diferentes variantes, una de ellas es que se suele sintetizar el sonido con ondas sinusoidales, se resta a la señal original la que se acaba de sintetizar y, finalmente, se almacena esa señal resta, conocida como residuo, para incorporarla, junto a la del modelo sinusoidal, a la síntesis final.

### 2.4.3. Síntesis por modulación de frecuencia (FM)

La síntesis por modulación en frecuencia, también llamada síntesis FM, fue creada por John Chowning en la Universidad de Stanford en los años 70.

La base de esta técnica de síntesis consiste en utilizar una señal portadora, normalmente una señal sinusoidal, y variar su frecuencia con otra señal llamada moduladora, la cual también suele ser una sinusoidal. Esto se puede ver con la siguiente expresión:

$$y(t) = A_c \operatorname{sen}(2\pi(f_c + A_M \operatorname{sen}(2\pi f_M t + \phi_M))t + \phi_C) \tag{2.2}$$

Lo que estamos produciendo con este proceso es que alrededor de la frecuencia de la señal portadora aparezcan una serie de parciales en posiciones que vienen dadas por razones aritméticas[8] entre las frecuencias de la señal portadora y moduladora. Podemos ver un ejemplo de este tipo de síntesis en la figura 2.7.

Para la síntesis de voz, lo que se suele implementar es la suma de varias señales sinusoidales cuyo valor de portadora es el valor de la frecuencia central del formante que se pretenda conseguir, estando cada una de ellas modulada en frecuencia para así obtener un formante de mayor anchura y, por consiguiente, más realista.

### 2.4.4. Síntesis concatenativa

Se basa en la utilización de un banco de sonidos grabados[9] que se van uniendo para conseguir la señal deseada. Esta técnica es la forma de síntesis que, actualmente, consigue producir un sonido más inteligible y natural. Sin embargo, tiene dos inconvenientes que, además, están relacionados:

1. Necesita bastante memoria para conseguir un banco de sonidos que tenga una biblioteca amplia para poder sintetizar diferentes sonidos.

2. No existe la unidad a grabar idónea para todos los casos: podemos grabar palabras, sílabas, difonemas, fonemas... y de ello depende tanto el resultado final como la memoria a utilizar.

[8] Tras la modulación quedan $2K + 1$ parciales, siendo $K = I + 1$ e $I = \frac{A_M}{f_M}$ en las frecuencias $|f_c \pm k \cdot f_m|$ con $k = 0, 1, 2..., K$.

[9] Los sonidos en los bancos son conocidos como samples y de ahí que esta técnica sea conocida también como síntesis por samplers.

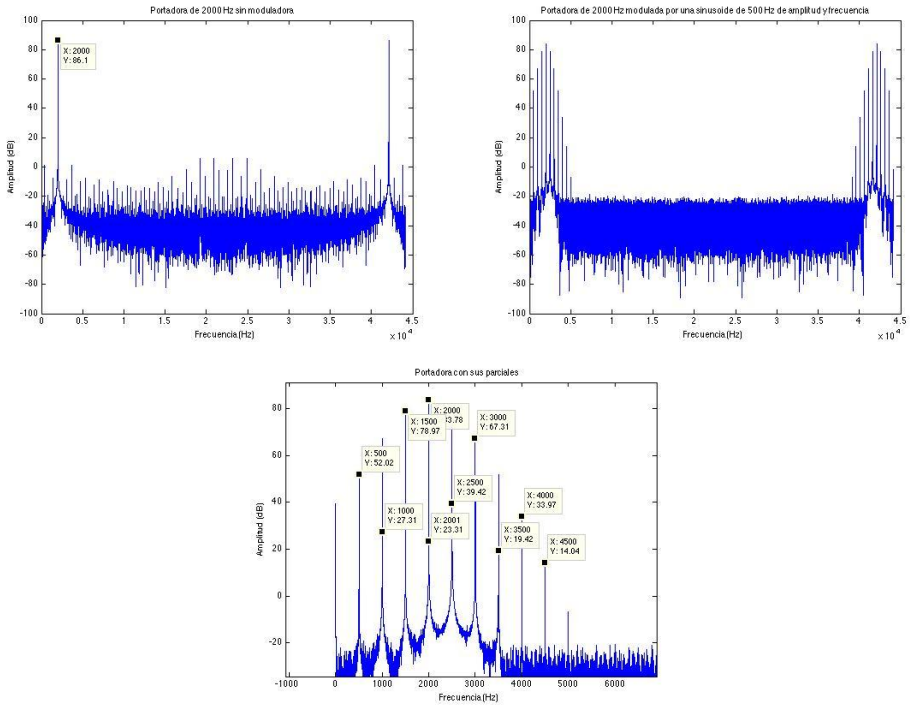


Figura 2.7: Espectros de portadora de 2000 Hz sin modular (superior izda.), portadora modulada (superior dcha.), y captura de los parciales presentes tras la modulación (inferior).

Cabe destacar que en este tipo de síntesis, en contra de lo que pueda parecer más obvio, no es necesario el obtener una muestra de cada uno de los sonidos que se quieran obtener, sino que, si lo que deseamos es una señal que ya tenemos pero con una altura diferente, existen algoritmos de transposición de la misma. Sin embargo, no es posible utilizar estos algoritmos de forma indiscriminada ya que, por ejemplo en transposiciones de alturas relativamente grandes, no pueden obtener un resultado de calidad. Por ello, y también por el hecho de que normalmente el timbre suele variar con la altura, se recurre a una solución de compromiso: se obtiene una serie de muestras para actuar como base de toda la síntesis y el resto se obtienen mediante alguno de los algoritmos antes citados.

### 2.4.5. Síntesis por modelos físicos

Es un tipo de síntesis que intenta reproducir los órganos vocales (desde órganos articuladores hasta las cuerdas vocales) de la forma más perfecta

posible utilizando gran cantidad de modelos físicos y matemáticos. En este tipo de síntesis, en lugar de parametrizar y estudiar la señal que se desea obtener, lo que se hace es un estudio de los elementos que la producen (en este caso, todo el aparato fonador) y se intenta modelar cada uno de los elementos internos para que se comporten de la misma forma que los reales, produciendo así, en teoría, los mismos resultados.

Es el modelo que, se supone, mejor resultado puede dar. Sin embargo, por ahora es inviable, al menos de cara a estar fuera de un laboratorio, por la gran cantidad de potencia de cálculo necesaria para operar en tiempo real, por el hecho de la gran cantidad y tamaño de los modelos que se han de hacer.

### 2.4.6. Síntesis granular

La síntesis granular se basa en el principio de que cualquier sonido puede descomponerse (o reproducirse) en una serie de elementos más simples denominados gránulos sónicos.

Un gránulo sónico, base de esta forma de síntesis, es cualquier sonido de duración arbitraria (desde una simple frase hasta incluso canciones enteras) que ha sido comprimido[10] a una duración menor a 50 milisegundos. Con este proceso tenemos elementos que, si los intentamos escuchar, no tienen altura ni timbre que nosotros podamos distinguir. Sin embargo, aquí viene la clave de esta forma de síntesis: para crear un sonido con una determinada frecuencia fundamental lo que hacemos es coger un gránulo sónico y lo vamos repitiendo cada cierto tiempo, concretamente el período que marque la frecuencia fundamental a conseguir. Con este simple proceso conseguimos un sonido con la altura deseada y con un timbre que será función del timbre de la señal original que hemos convertido en gránulo sónico.

Dando un giro más a esta forma de síntesis, y por ser aplicación directa en el presente proyecto, la forma de conseguir formantes es por un proceso de enventanado de señales, concretamente sinusoidales: con una sinusoidal de frecuencia la del formante que queremos y con una envolvente concreta[11] realizamos una multiplicación en el tiempo[12], realizando una modulación con

[10]Esto no es del todo cierto ya que si el sonido que tenemos ya tiene una duración menor a 50 milisegundos, no se comprime.

[11]El espectro de este formante dependerá, en gran medida, de la forma temporal/frecuencial de la envolvente.

[12]Esto implica convolución en el dominio frecuencial.

la que conseguimos que el espectro de la envolvente se sitúe a la frecuencia de la sinusoide. Con este proceso únicamente obtenemos un formante, pero si realizamos procesos similares con más envolventes y más sinusoides, sumando los resultados de cada uno de ellos obtendremos el espectro de formantes deseado.

Por último, en lo relativo a esta forma de síntesis, destacar que la primera persona en especular algo sobre esto fue el físico Dennis Gabor, entre otras cosas ganador del Premio Nóbel de Física por la invención de la holografía.

### 2.4.7. Síntesis de formas de ondas formantes (FOF)

La síntesis FOF, creada en el centro IRCAM de París para su proyecto de voces cantadas *CHANT*, se basa en la creación de los formantes que ha de tener una señal para conseguir imitarla. Esto se puede ver, en realidad, como otra forma de nombrar la síntesis sustractiva debido a que lo que se está haciendo es modelar un espectro para obtener formantes. Sin embargo, la síntesis FOF a la que aquí nos referimos es una variante de la síntesis granular: en ella, los gránulos son sinusoides amortiguadas que harán las veces de formantes (la frecuencia del formante será la frecuencia de la sinusoidal), conformando así el espectro de la señal final y el período (o frecuencia) de repetición de ese gránulo conformará la frecuencia fundamental de la señal, es decir, su altura. Por tanto, la síntesis de voz con este método se dará sumando diferentes señales, todas con una misma frecuencia fundamental (altura) pero cuyos gránulos serán diferentes, a pesar de ser típicamente sinusoidales, en función del formante que se quiera crear.

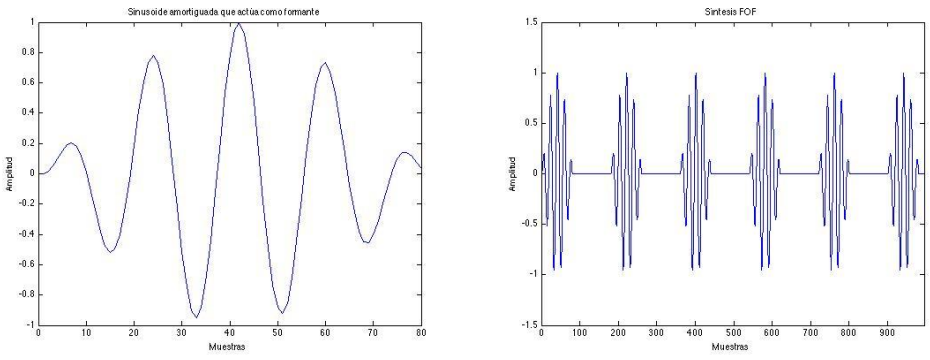


Figura 2.8: Detalle de una sinusoide amortiguada (izda.) y de una simulación de funcionamiento del algoritmo FOF (dcha.).

# Capítulo 3

# Estudio de los programas de partida del proyecto

En el tercer capítulo de la presente memoria vamos a realizar el estudio de aquellos programas que se van a utilizar para la consecución de los objetivos planteados en el primer capítulo.

Las herramientas software que de las que partimos son el extractor de metadatos *midi2letra*, el cual nos proporciona la herramienta de lectura de archivos MIDI, y el propio sintetizador de vocales sobre el que parte el presente proyecto. En base al estudio de estas herramientas, posteriormente se realizará una modificación de las mismas para así conseguir nuestros objetivos.

## 3.1. Estudio del extractor midi2letra

Antes de comenzar, cabe destacar que el programa *midi2letra* que se va a estudiar es una modificación realizada sobre un programa del Grupo de Reconocimiento de Formas e Inteligencia Artificial (GRFIA) del Departamento de Lenguajes y Sistemas Informáticos (DLSI) de la Universidad de Alicante llamado *metamidi*[1]. El software *metamidi* fue creado para llevar a cabo la extracción de metadatos de cualquier tipo dentro de un archivo MIDI por lo que, aprovechando esta característica de este programa, se realizó, por y para el presente proyecto, *midi2letra*, cuyo función es la localización de metaeventos de tipo texto (concretamente, los tipificados como metadatos $05_{\text{H}}$).

[1] http://grfia.dlsi.ua.es/gen.php?id=resources

Retomando el análisis del programa, básicamente, la forma en que este programa funciona, se puede resumir con el siguiente diagrama de bloques:

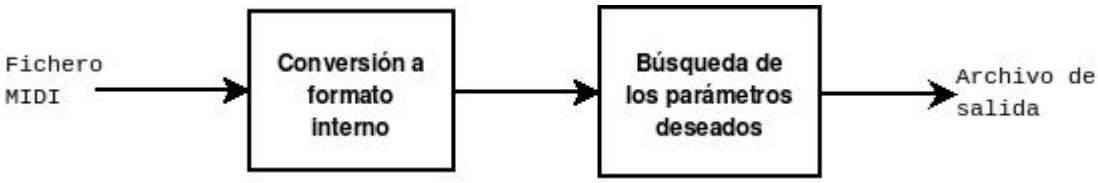


Figura 3.1: Funcionamiento básico del programa *midi2letra*.

El formato de la estructura TMidi en la que nosotros almacenamos cada una de las características de una secuencia MIDI es el que sigue:

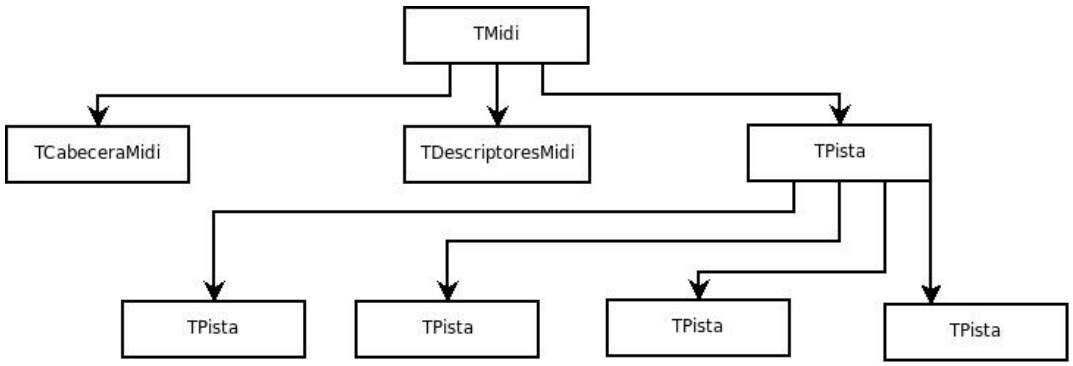


Figura 3.2: Relación de las diferentes estructuras dentro del programa *midi2letra*.

A continuación describiremos detalladamente lo que realiza cada estructura y lo que significa cada una de las variables (estructuras realmente) que tiene asociadas internamente. Antes de comenzar con la descripción, cabe destacar que algunos campos de las diferentes estructuras a comentar utilizan unos tipos de datos propios, definidos al comienzo de todo. Estos tipos de datos, especificados en lenguaje C, son los que siguen:

```
typedef char CODE4[5];
typedef unsigned long DOBLE;
typedef short int DATOS;
typedef unsigned char DATOS8;
```

Una vez definido esto, podemos proceder a la definición de las diferentes estructuras:

- Estructura TMidi
  Es la estructura que representa el fichero MIDI leído, es decir, es como si fuera el propio archivo MIDI pero almacenado dentro del programa pudiendo, por tanto, acceder a cualquiera de los datos requeridos.

  El código en C de esta estructura es el que sigue:

```
typedef struct {

   TCabeceraMidi     cabecera;
   TDescriptoresMidi descriptores;
   TPista            pistas[64];

} TMidi;
```

- Estructura TCabeceraMidi
  Es la estructura que almacena los datos de la cabecera del archivo MIDI que estemos procesando.

  Su estructura, en lenguaje C, es la que sigue:

```
typedef struct {

   CODE4  id;
   DOBLE  size;
   DATOS  format;
   DATOS  numpistas;
   DATOS  delta;
   char* name;

} TCabeceraMidi;
```

- Estructura TDescriptoresMidi
  Estructura que almacena los descriptores de una pista que contenga el archivo MIDI que estemos procesando.

  Se define, en lenguaje C, de la siguiente manera:

```
typedef struct {
  char *path;
```

```
  long int bytes;
  int formato;
  int division;
  int numpistas;
  char* texto;
  float tempo;
  int num_compas;
  int den_compas;
  char *compases;
  int tono;
  int modo;
  char* key;
  int c_tempo;
  int c_metrica;
  int c_tono;
  int duracion;
  int hasSysEx;
  char* instrumentos;

} TDescriptoresMidi;
```

- Estructura TPista
  Estructura que almacena todo lo referente a una pista del archivo MIDI.

  En lenguaje C se define de la siguiente manera:

```
typedef struct {

   TCabeceraPista     cabecera;
   TEvento            *evento;
   TNota       *nota;
   TDescriptoresPista descriptores;

} TPista;
```

- Estructura TCabeceraPista
  Estructura que almacena los datos de las cabecera de una pista del archivo MIDI.

  Su declaración, en lenguaje C, es la que sigue:

```
typedef struct {
```

```
   CODE4   id;
   DOBLE   size;
   DATOS8  *data;
   DATOS   numeventos;
   DATOS   numnotas;

} TCabeceraPista;
```

- Estructura TEvento
  Estructura que almacena información de cada uno de los eventos que tienen lugar en la pista MIDI.

  La podemos definir de la siguiente manera en lenguaje C:

```
typedef struct {

   DATOS   tipo;
   DATOS   mensaje;
   DATOS   canal;
   DATOS   dato1;
   DATOS   dato2;
   DOBLE   dato3;
   char*   texto;
   int     tini;
   int    t_delta;

} TEvento;
```

- Estructura TNota
  Estructura que guarda la información de las notas de una pista MIDI.

  Su definición en lenguaje C es la que sigue:

```
typedef struct {

  DATOS nota;
  DATOS velocidad;
  int   inicio;
  int   duracion;
  DATOS ligada;
  DATOS compas;
```

```
} TNota;
```

- Estructura TDescriptoresPista
  Estructura que almacena los descriptores de una pista MIDI.

  La podemos definir en lenguaje C de la siguiente forma:

```
typedef struct {

  char *texto;
  int   nota_baja;
  int   longbytes;
  float media;
  float desviacion;
  int   nota_alta;
  int   programa;
  int   canal;
  int   duracion;
  int   d_sonido;
  float sonido;
  float polifonica;
  float d_absoluta;
  float s_absoluto;
  int   n_cambiosPrograma;
  char *cambiosPrograma;
  int   n_notas;
  int   n_msgModulacion;
  int   n_msgAltura;
  int   n_msgPostPul;
  int   maxPolyphony;
  float avgPolyphony;

} TDescriptoresPista;
```

## 3.2. Estudio del sistema de síntesis de vocales existente

En este apartado vamos a estudiar qué hace el código que tenemos sobre el sintetizador de vocales en el que nos vamos a basar para crear nuestro

sintetizador que, además de las vocales, también incluirá las consonantes.

El sintetizador a estudiar se basa por completo en la síntesis de tipo sustractivo: tenemos un excitador[2] para cada instrumento (cada una de las voces, que son soprano, contralto, tenor y bajo) más después cinco filtros para cada uno de esos excitadores que se encargarán de generar las vocales. Como cuando se vaya a sintetizar un sonido se necesitará hacer uso de dos instrumentos diferentes (uno hará de excitador y otro el filtro), en este programa se hace necesario la utilización de variables de tipo global que comuniquen ambos instrumentos.

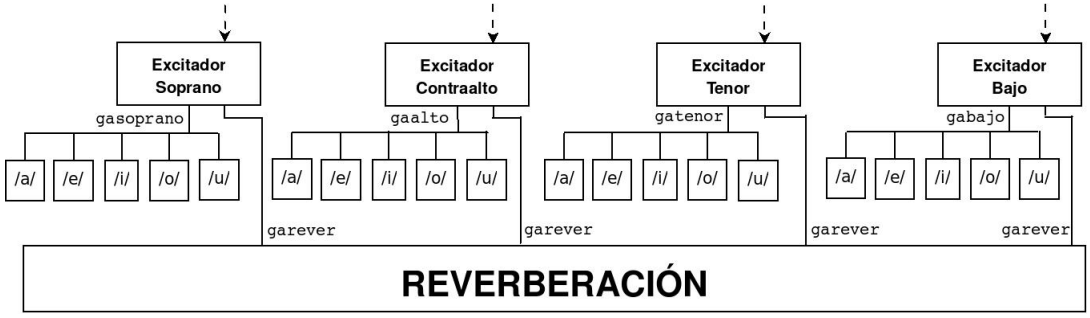


Figura 3.3: Diagrama de bloques de la orquesta del coro de vocales.

### 3.2.1. Estudio de los excitadores

Antes de comenzar el análisis de estos instrumentos, cabe destacar que la codificación que utiliza el sintetizador para los mismos es un único número que representa la voz que crea, de la siguiente manera:

1. Voz Soprano
2. Voz Contralto
3. Voz Tenor
4. Voz Bajo

Siguiendo con el estudio de estos excitadores, la estructura de los instrumentos de excitación es relativamente sencilla, pudiendo ser representada por el diagrama de bloques de la figura 3.4.

[2]Recordemos, del Capítulo 2, que el excitador es la *energía* inicial del sistema.

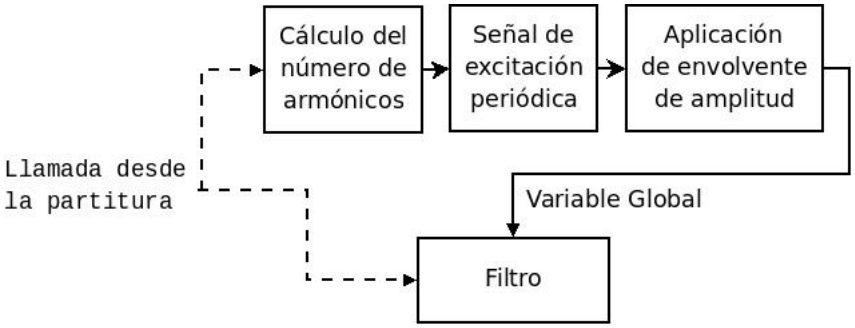


Figura 3.4: Diagrama de bloques genérico de los instrumentos de excitación.

Vamos a estudiar cada uno de estos bloques genéricos que se han expuesto aquí para comprender mejor qué es lo que realizan:

- Cálculo del número de armónicos
  El cálculo que da nombre a este apartado se ha realizado de la siguiente forma:
  $$\text{Número armónicos} = 0{,}4 \cdot \frac{f_S}{f_O} \tag{3.1}$$
  donde $f_S$ representa la frecuencia de muestreo y $f_O$ es la frecuencia de la señal a obtener.

  Como podemos observar, este cálculo tiene bastante sentido porque lo que estamos haciendo es que la cantidad de armónicos que ha de tener la señal, lo cual obviamente depende de forma inversa de su frecuencia, quepa en el ancho de banda del que disponemos para no producir aliasing haciendo cumplir el criterio de Nyquist: que el ancho de banda que tengamos sea menor que la mitad de la frecuencia de muestreo del sistema.

  Finalmente destacar que este cálculo es necesario hacerlo porque el *opcode* que utilizaremos para crear la señal lo necesita.

- Creación de la señal de excitación
  Para la creación de esta señal se ha utilizado el *opcode* **gbuzz** que permite crear una serie aditiva de parciales coseno relacionados armónicamente. Aunque hay otro *opcode* relacionado con éste (la sentencia **buzz**), el que se está utilizando permite que la serie de armónicos no tengan todos la misma amplitud, algo que le confiere cierto realismo

al sintetizador. El espectro de ambos *opcodes* lo podemos ver en las gráficas de la figura 3.5, además del detalle sobre la armonicidad de *opcode* **gbuzz** en la figura 3.6.

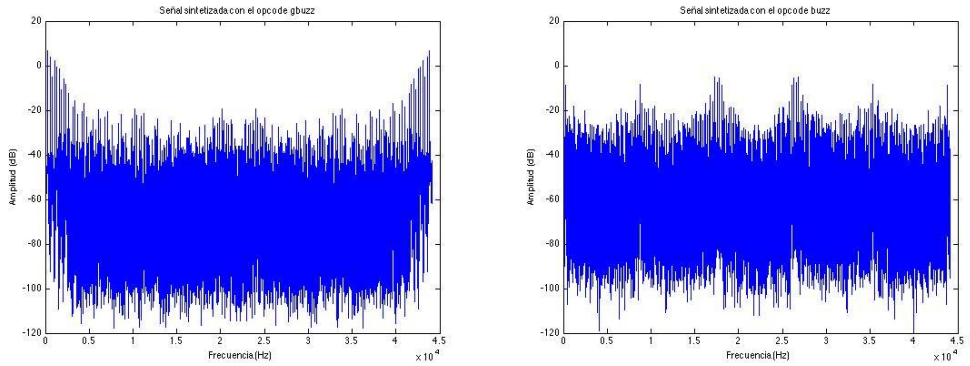


Figura 3.5: Espectros en potencia de los *opcodes* **gbuzz**(izda.) y **buzz**(dcha.).

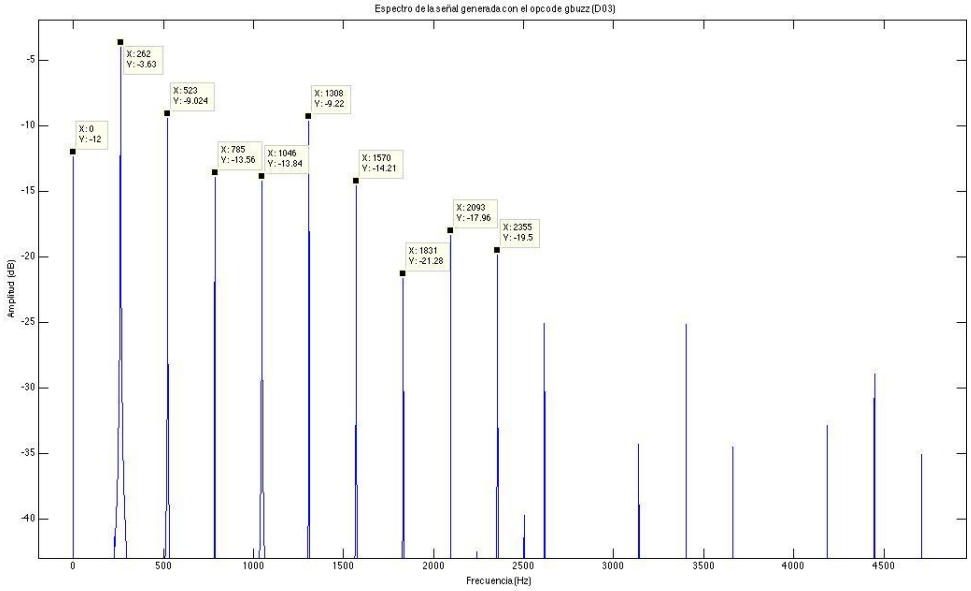


Figura 3.6: Detalle de la armonicidad del *opcode* **gbuzz**.

Como podemos observar, el *opcode* **gbuzz** crea una señal cuya amplitud del espectro en altas frecuencias es, en promedio, menor que en las

frecuencias bajas. En cambio, el *opcode* **buzz**, aunque no mantiene un nivel constante, sí que oscila siempre sobre los mismos valores.

- Aplicación de una envolvente de amplitud
  Una vez hemos sintetizado la señal excitadora, aunque posteriormente la vayamos a tratar en el instrumento de filtrado correspondiente, hemos de aplicarle una envolvente de amplitud, en este caso se aplica de tipo ASR, para evitar ruidos que se puedan producir en el inicio y el final de cada sonido.

  Para llevar a cabo esta tarea se utiliza el *opcode* **linseg** y, tras haber generado la envolvente, se aplica a la señal sintetizada multiplicando directamente la envolvente por la propia señal en el dominio temporal.

  En todas las señales de excitación se está empleando la envolvente definida en la figura 3.7.

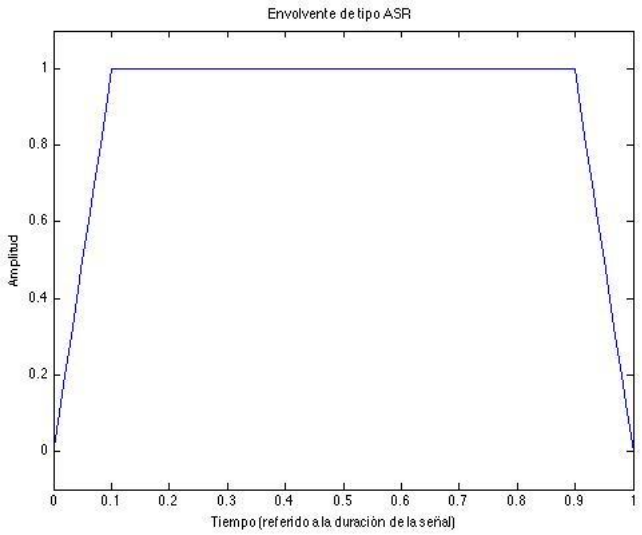


Figura 3.7: Envolvente de tipo ataque–sostenimiento–relajación (ASR) aplicada a la señal de excitación.

Del estudio del código se extrae la conclusión de que tanto el tiempo de ataque como el tiempo de relajación son estáticos (siempre duran 0.1 segundos) mientras que el que sí que vara es el tiempo de sostenimiento

de la señal. Esto que en principio puede parecer incorrecto por el hecho de utilizar tiempos estáticos, se puede justificar por los siguientes motivos:

1. La duración del tiempo de ataque y de relajación no dependen de la duración de la propia nota, imitando así el canto real en el cual, si no hay directrices contrarias en la partitura, siempre se "canta" la nota de la misma forma.
2. En principio no debe de haber problema porque los tiempos de ataque y relajación sean estáticos por el hecho de que, normalmente, las notas en canto coral son más largas que los tiempos establecidos.

- Envío a la variable global
  Finalmente, y como se ha especificado antes, por la forma en que se ha implementado este sistema, se hace necesario el uso de variables globales para conseguir pasar la señal sintetizada desde el instrumento excitador hasta el filtro, con lo cual, llegados a este punto, simplemente sumamos el valor de la señal sintetizada al valor de la variable global correspondiente (en función de si estamos en el excitador de soprano, contralto, tenor o bajo).

### 3.2.2. Estudio de los filtros

Antes de comenzar, como ya hemos hecho antes, cabe definir cómo se están numerando estos instrumentos para más adelante poder asociarlos a la voz que queremos. Para la numeración de los filtros se está utilizando un número de dos cifras de las cuales la primera es la voz numerada de la misma manera que los excitadores (1, 2, 3 o 4 según la voz) y el segundo número depende de la vocal como aquí se especifica:

| Voz | a | e | i | o | u |
|---|---|---|---|---|---|
| Soprano | 11 | 12 | 13 | 14 | 15 |
| Contralto | 21 | 22 | 23 | 24 | 25 |
| Tenor | 31 | 32 | 33 | 34 | 35 |
| Bajo | 41 | 42 | 43 | 44 | 45 |

En este caso, como ocurría en los excitadores, los instrumentos de filtrado también guardan una gran similitud entre ellos cambiando sólo algunos parámetros en concreto.

La estructura genérica de estos instrumentos se puede ver con el diagrama de bloques de la figura 3.8.

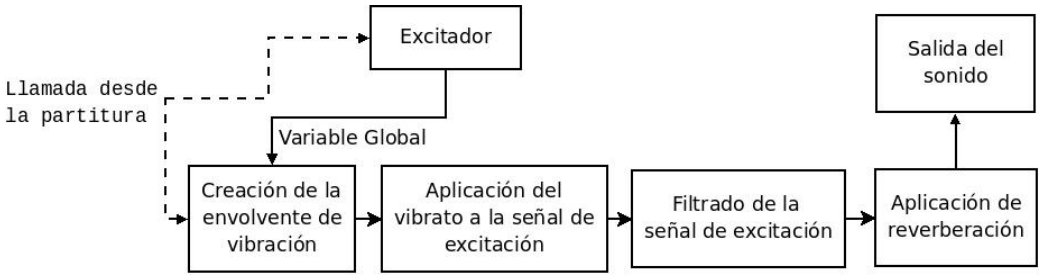


Figura 3.8: Diagrama de bloques genérico de los instrumentos de filtrado.

A continuación vamos a describir estas unidades:

- Creación de la envolvente de vibrato[3]
  El vibrato, como efecto de sonido, se aplica siempre sobre notas largas o, al menos, la parte más larga de alguna de las notas. Esto tiene siempre una doble finalidad:

  1. Ayudar al cantante a ajustar el tono de una nota con la melodía.
  2. Mantener la atención del oyente haciendo que una nota larga no sea, desde un punto de vista de la altura, constante.

  Retomando el análisis del algoritmo, para crear la envolvente de vibrato lo que el programa realiza son dos acciones:

  1. En primer lugar, mediante cuatro osciladores de baja frecuencia (LFO's), se crean cuatro valores de retardo de señal distintos que serán utilizados en la siguiente etapa.
  2. Aquí lo que tiene lugar es la creación de una línea de retardo con la extracción de cuatro valores de esa línea en distintos instantes (los cuatro instantes obtenidos en el primer apartado).

- Aplicación del vibrato a la señal de excitación
  En lo referente a la aplicación del vibrato, se puede resumir en que se suman las cuatro señales obtenidas en el apartado anterior como resultado de crear cuatro retardos distintos, ponderando cada una de ellas por un cuarto del valor total, para no saturar la señal resultante.

[3]El efecto de vibrato es la variación periódica de la frecuencia fundamental de una señal.

- Filtrado de la señal de excitación
  En esta parte del instrumento se implementa el filtrado de la señal, ya con el vibrato, por medio de filtros paso-banda para así crear los diferentes formantes deseados.

  En función de cuál sea la vocal a sintetizar y de cuál sea la voz que la ha de decir encontramos las diferentes frecuencias (con sus diferentes amplitudes) a las que se crea ese filtrado, las cuales se adjuntan en el Apéndice B de la presente memoria.

  La implementación de esto se ha llevado a cabo mediante un modelo de síntesis sustractiva en paralelo: filtramos la señal con diferentes filtros en paralelo para después sumar el resultado de cada una de las líneas. Al estar realizando un filtrado en paralelo, a la hora de sumar todas las señales hemos de dar una amplitud relativa a cada una de ellas, que hará las veces de amplitud del formante que estemos tratando.

- Aplicación de la reverberación
  Mandamos por la variable global *garever* la proporción de reverberación que queramos, a la unidad que crea este efecto. En todos los casos, se manda un 47 % de la señal recién sintetizada.

- Salida al archivo de sonido
  Una vez creado el sonido especificado por la partitura (cada una de las notas que corresponden a diferentes vocales), éste se manda al archivo de sonido especificado. En lo relativo a este apartado cabe destacar dos características:

  1. Los instrumentos son de tipo estéreo lo que implica que nosotros podemos, fácilmente, panoramizarlos[4] a nuestro gusto. En la orquesta original, se envía siempre la misma señal tanto por el canal izquierdo como por el derecho, y siempre con la máxima amplitud.
  2. El archivo de sonido que se crea puede ser de cualquiera de los diferentes formatos soportados por CSound[5], siendo WAV el tipo de archivo por defecto en sistemas PC como Windows o Linux, mientras que en sistemas Mac es el formato AIFF.

[4] Hace referencia a la posibilidad de simular la situación de los instrumentos en diferentes posiciones del espacio.

[5] AIFF, WAV, IRCAM.

### 3.2.3. Estudio de la unidad de reverberación

La unidad de reverberación, de acuerdo con la información que se adjunta en el propio código, no utiliza los *opcodes* que tiene CSound para crear el efecto comentado, sino que se basa en una unidad de reverberación llamada *Feedback Delay Network*[6] surgida del trabajo Stautner y Puckette [2].

Retomando el análisis del instrumento de reverberación, podemos representarlo por el diagrama de bloques de la figura 3.9.

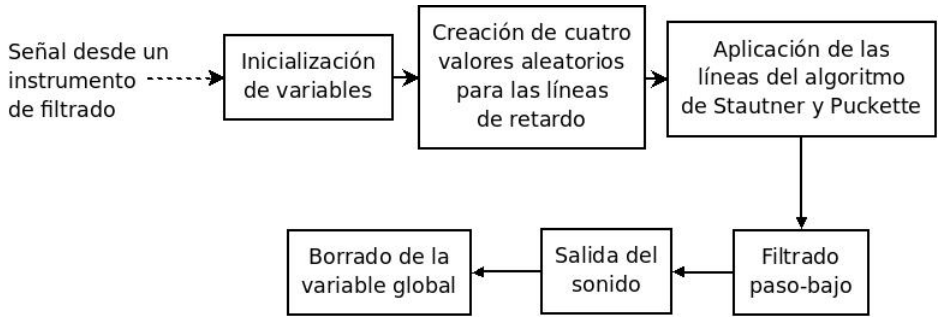


Figura 3.9: Diagrama de bloques de la unidad de reverberación

A continuación describiremos cada uno de estos bloques detalladamente:

- Inicialización de variables
  En un primer momento, lo que el algoritmo realiza es dar valor nulo a cuatro variables que luego utilizará para el filtrado paso-bajo de diferentes señales.

- Creación de valores para las líneas de retardo
  Mediante un *opcode* de creación de valores aleatorios (en este caso, **randi**) obtiene cuatro valores aleatorios que luego aplicará en las líneas de retardo de la señal para simular la reverberación.

- Aplicación de las líneas de retardo
  Este bloque resume dos actividades:

  1. Creación y almacenamiento de una gran cantidad de retardos sobre la señal que llega por la variable global de la reverberación con el *opcode* **multitap**.

[6]Para una mejor comprensión de esta unidad se recomienda la lectura del Apéndice E.

2. Creación de cuatro líneas de retardo que modifican la señal en función del valor aleatorio obtenido anteriormente, además de dos parámetros que se envían desde la partitura (cantidad de retardo general y modificación de la frecuencia fundamental). Además, estas líneas están configuradas según describe la matriz de ganancias del algoritmo de Stautner y Puckette [2].

- Filtrado paso-bajo
  Se filtra paso-bajo la señal para causar mayor realismo ya que, en la realidad, por efecto de la gran atenuación del medio con las altas frecuencias, éstas se pierden bastante rápido tras muy pocas reflexiones. En general, los reverberadores digitales incluyen este tipo de filtrado para emular el efecto real.

- Salida del sonido
  Se envía el sonido, en estéreo según la configuración que propone el algoritmo, a la señal principal que se está creando.

- Borrado de la variable global
  Borramos la variable que envía la señal de reverberación para poder reutilizarla, si fuese necesario.

# Capítulo 4

# Creación del software para el control del sintetizador ya existente

Este capítulo tiene por finalidad el crear un programa que permita extraer de un archivo MIDI una partitura de CSound para nuestro sintetizador, es decir, que queremos realizar un sistema que facilite el control del sintetizador. Para llevar a cabo esta tarea antes será necesario plantear un formato de archivo MIDI concreto para así simplificar el diseño de este sistema y, a partir de ahí, se podrá construir la aplicación de generación automática de partituras para el sintetizador de coros.

## 4.1. Restricciones al diseño

Para ahorrar complejidad en el diseño y creación del programa, se ha establecido que el fichero MIDI de origen tenga siempre una estructura muy concreta (puede diferir un poco de lo que aquí se exponga, pero si lo hace demasiado el sistema no funcionará correctamente).

Las restricciones que se van a imponer son las siguientes:

- El fichero tendrá 5 pistas: la primera (pista número 0) en la que, opcionalmente, se podrá incluir información en general (autor, copyright, tonalidad, métrica, tempo...) pero que no contendrá notas (no ocurrirá nada si las tiene ya que el programa las ignorará) y las otras cuatro (pistas 1 a 4) que corresponderán a las cuatro voces a sintetizar. Esta característica es estándar para los ficheros MIDI multipista.

- Las pistas 1 a 4, en numeración ascendente, corresponderán a las voces, según se especifica a continuación:

  1. Soprano
  2. Contralto
  3. Tenor
  4. Bajo

- Todas las notas que se quiera que el programa lleve a la partitura deberán llevar algo en su metadato correspondiente[1] a letra (metadato tipo $05_{\text{H}}$). Las que no lleven nada serán ignoradas.

- Si falta alguna de las pistas el programa podrá funcionar sin generar sonidos para esa voz, recordando que toda pista 0 será ignorada por el programa en lo relativo a extracción de notas.

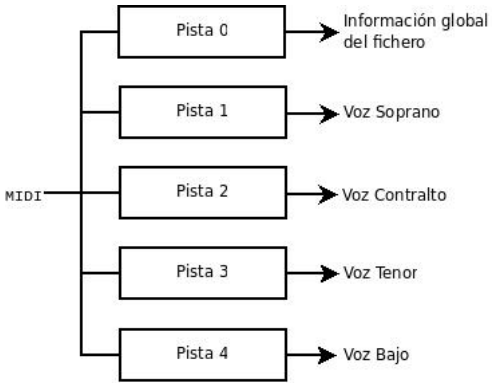


Figura 4.1: Diagrama de la estructura básica de nuestro archivo MIDI necesario.

## 4.2. Creación del programa midi2letra

Como ya sabemos, el control del sintetizador consiste en un programa capaz de transformar la información proveniente de un archivo MIDI, que ha de tener la estructura antes descrita. Este programa será, realmente, un *back-end* del programa midi2letra, cuya estructura general, en forma de diagrama

[1] Se refiere a que el metadato con el texto sea simultáneo al inicio de la nota que le da su altura.

de flujo, podemos encontrar en la figura 4.2.

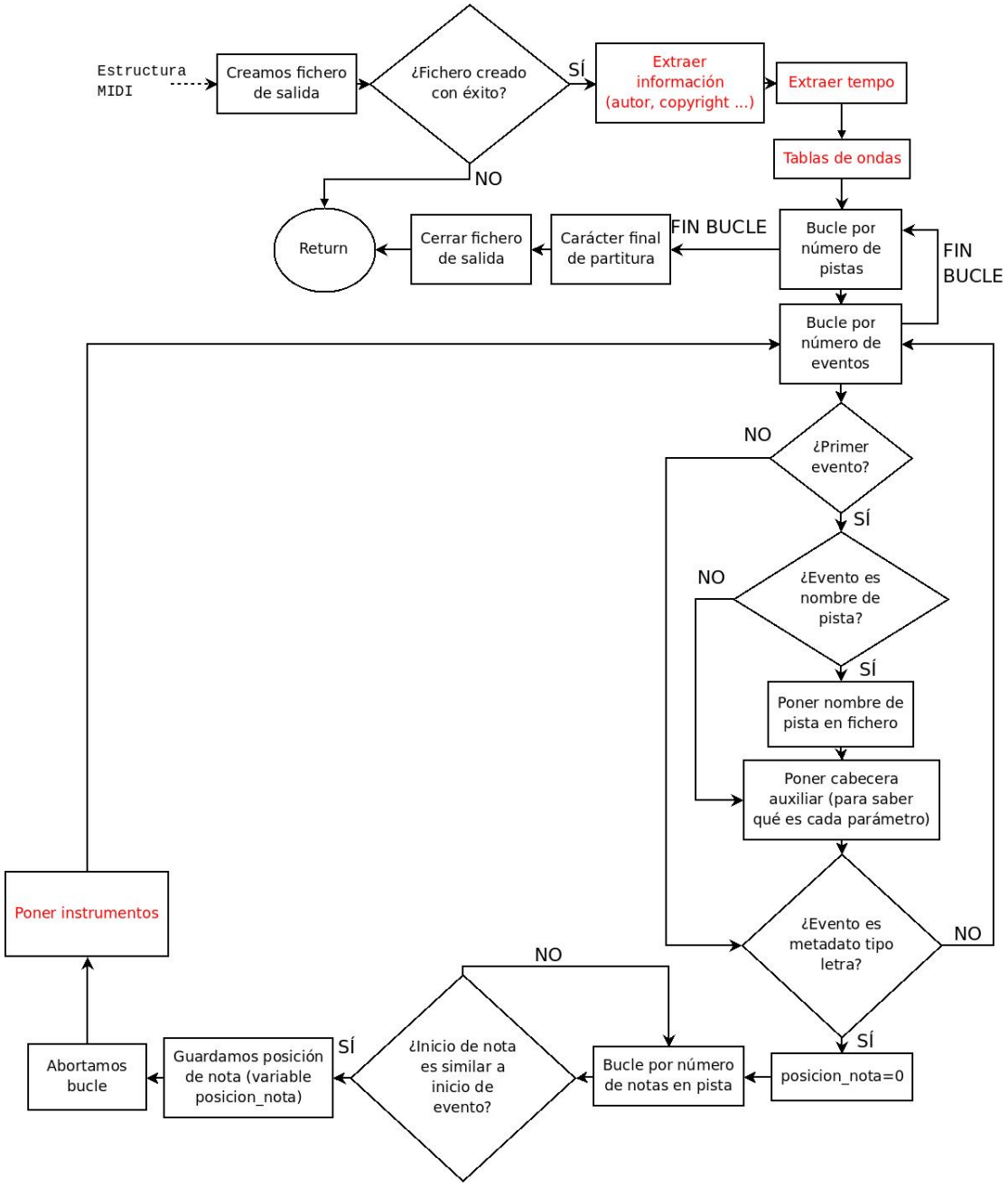


Figura 4.2: Diagrama de bloques de la función principal del *back-end* del programa *midi2letra*.

Como podemos observar, el único parámetro de entrada que necesita esta función es la estructura de tipo TMidi que el programa *metamidi* crea y utiliza internamente.

Esta función, a su vez, llama internamente a otras funciones, las cuales aquí hemos representado con un único bloque con letra roja pero que vamos a explicar a continuación:

- Extraer información
  Esta función viene definida por el siguiente diagrama de bloques de la figura 4.3.

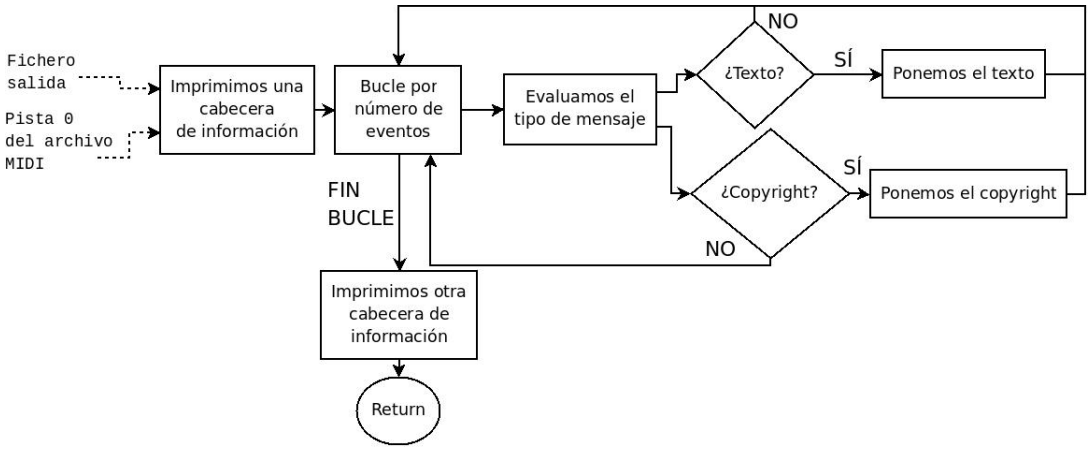


Figura 4.3: Diagrama de bloques de la función de extracción de información.

Como podemos observar, esta función recibe como parámetros el fichero de salida sobre el que estamos escribiendo y la pista 0 del archivo MIDI a procesar (obviamente, es la pista 0 pero en formato de estructura TMidi, que es como trabaja este programa) porque es la pista en la que, según nuestro diseño, estará toda la información no musical que el autor quiera incluir.

Lo que se ha implementado aquí es simplemente un recorrido por cada uno de los eventos de la pista para buscar aquellos que nos interese. Actualmente sólo está implementada la búsqueda de metaevento de tipo Texto, para incluir el nombre de la pieza secuenciada o cualquier texto, y de tipo Copyright, para incluir el autor de la secuencia que se

está procesando. El programa de extracción de partitura incluirá estos datos al comienzo de la misma para informar de estos datos a cualquiera que los desee conocer.

Por último, para dejar totalmente definido el diagrama, cabe resaltar que los bloques titulados *Imprimimos una cabecera de información* e *Imprimimos otra cabecera de información* se refieren a la inclusión de las cadenas de texto "*;Datos presentes en el archivo midi acerca del contenido del mismo:*" y "*;Partitura extraída del archivo*" respectivamente en el archivo de partitura, con la finalidad de limitar la zona de información no necesaria para la síntesis del sonido dentro de la misma.

- Extraer tempo
  El diagrama de bloques que define esta función lo encontramos en la figura 4.4.

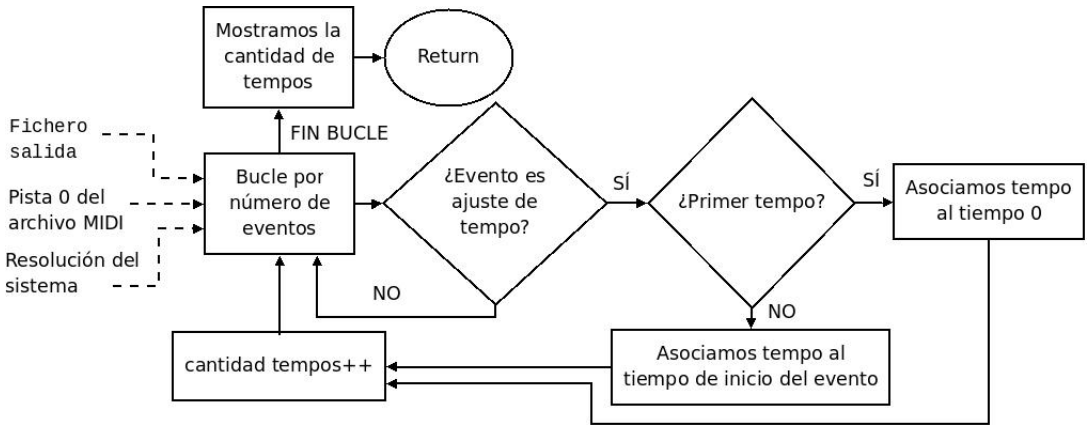


Figura 4.4: Diagrama de bloques de la función de extracción de tempos del archivo MIDI.

Como podemos observar, lo que estamos implementando es un recorrido por todos los eventos de una pista de las del fichero en busca de aquéllos que produzcan cambios de tempo. Cabe destacar que como el resultado esperado es independiente de la pista a utilizada, se ha establecido la utilización de la pista 0 por la seguridad de que siempre existe en un archivo MIDI, al menos si sigue nuestro diseño propuesto.

- Tablas de onda
  El diagrama de bloques de esta función es el que muestra la figura 4.5.

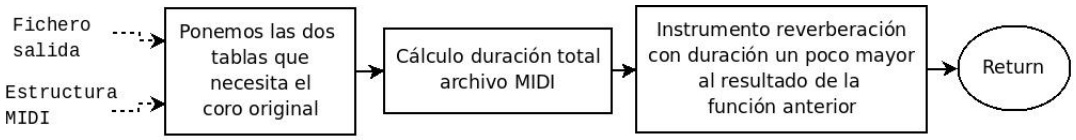


Figura 4.5: Diagrama de bloques de la función de escritura de tablas de onda necesarias.

Esta función es necesaria por el hecho de que, en el fichero de partitura, hemos de incluir las tablas de ondas que se van a utilizar durante el programa, las cuales se muestran en la figura 4.6.

Aunque parezca que ambas formas de onda son exactamente iguales, en realidad no lo son porque hay un pequeño desfasaje entre ellas. La razón de la existencia de dos tablas de onda tan parecidas pero diferentes no se ha podido esclarecer[2]. Por último, las expresiones matemáticas de ambas tablas de onda son:

$$\begin{aligned} y &= \operatorname{sen}(2\pi f t) \\ y &= \operatorname{sen}\left(2\pi f t + \tfrac{\pi}{720}\right) \end{aligned} \tag{4.1}$$

Como podemos observar, también creamos en esta función el instrumento de reverberación porque, al igual que las tablas de onda, es algo que va a estar siempre en el programa. Sin embargo, a pesar de que la mayoría de parámetros los podemos poner de forma estática (frecuencia de corte del filtro paso-bajo, tiempo de retardo en las líneas...), la que sí que debemos de saber es la duración total del fichero. Para ello se ha implementado una función que calcula la duración de la pista más larga del fichero, cuyo diagrama de bloques lo encontramos en la figura 4.7.

Como vemos, es un algoritmo de cálculo del máximo valor de una serie de valores, por lo que no requiere mayor explicación salvo que recibe la estructura MIDI como parámetro (por tener todas las pistas), va

[2]Al menos, auditivamente, parece obtenerse el mismo resultado tanto si utilizamos las dos tablas propuestas como si utilizamos dos iguales.

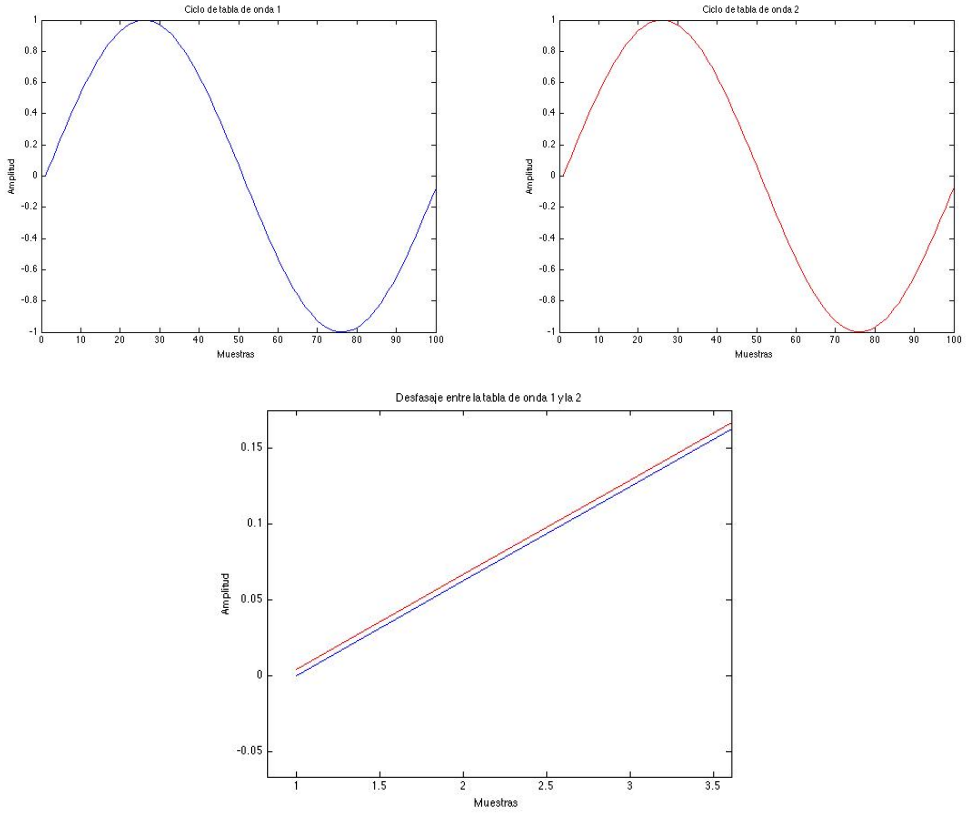


Figura 4.6: Ciclo de onda de la tabla 1 (superior izda.), ciclo de onda de la tabla 2 (superior dcha.) y desfase entre ambas (inferior).

evaluando el campo *duracion* de cada pista (dentro de la estructura *descriptores* de cada una) y que finalmente devuelve el máximo valor obtenido.

Destacar finalmente que, aunque tengamos la duración total del fichero, a la hora de escribir el instrumento en el archivo de partitura, nosotros siempre añadimos un poco más de duración del fichero (hemos puesto 5 tiempos musicales de más) para que la parada de la reverberación no sea súbita y, por tanto, dé un mejor resultado.

- Poner instrumento
  Esta función es la que se encarga de poner los instrumentos correctos. Cada llamada a esta función origina dos líneas de llamadas a instrumentos en la partitura con la finalidad de imitar la síntesis sustractiva:

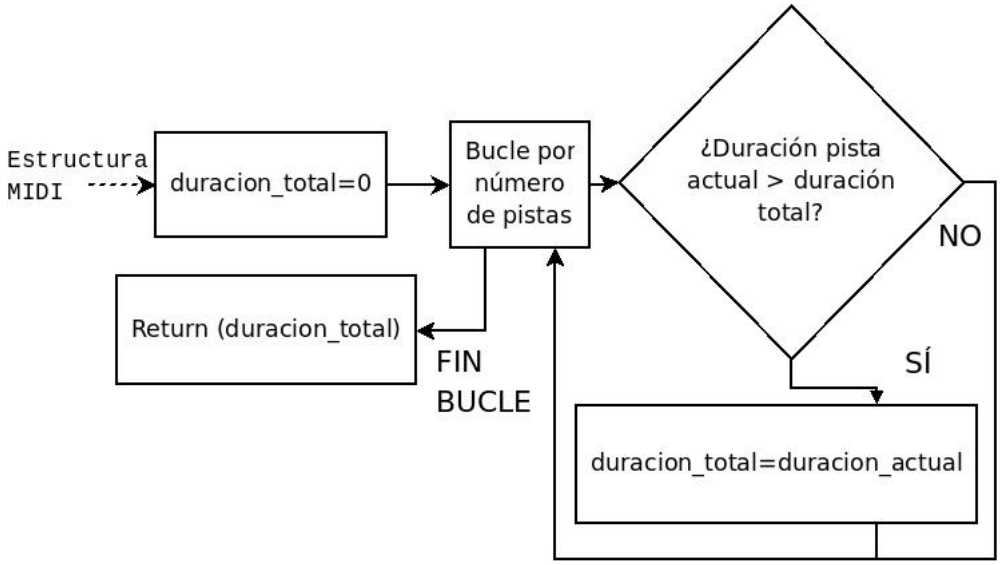


Figura 4.7: Diagrama de bloques de la función de cálculo de la duración del fichero MIDI.

hacemos una llamada al instrumento excitador y, posteriormente, al filtro.

El diagrama de bloques que representa esta función es el que muestra la figura 4.8.

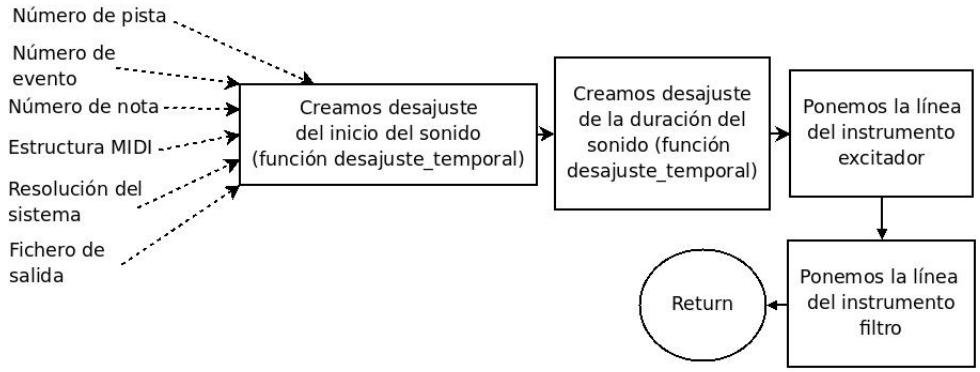


Figura 4.8: Diagrama de bloques de la función de inclusión de los instrumentos.

La función recibe los siguientes parámetros para su correcto funcionamiento:

- El número de la pista en que estamos.
- El número de evento en que nos encontramos.
- El número de la nota actual.
- El archivo MIDI.
- La resolución del sistema.
- El fichero de salida, es decir, la partitura.

Lo primero que realizamos en esta función es generar una serie de valores aleatorios que modificarán tanto el inicio como la duración de cada instrumento (realmente de cada par de instrumentos, es decir, que la variación de inicio y de duración será la misma cada vez que se ejecute esta función para que así coincidan instrumento de excitación y filtro en el tiempo) para dar una mayor sensación de realismo. El valor máximo de desajuste temporal está declarado como una constante. La función que devuelve este valor aleatorio es la que sigue:

- Desajuste temporal
  El diagrama de bloques de esta función es el que encontramos en la figura 4.9. Como vemos, como parámetro inicial recibe el tiempo de inicio de la señal (en el caso del desajuste de la duración, realmente recibirá la duración de la nota) y, en base a eso, lo único que hacemos es obtener un valor aleatorio, cuyo máximo es el antes comentado, que puede ser positivo o negativo, siempre que la

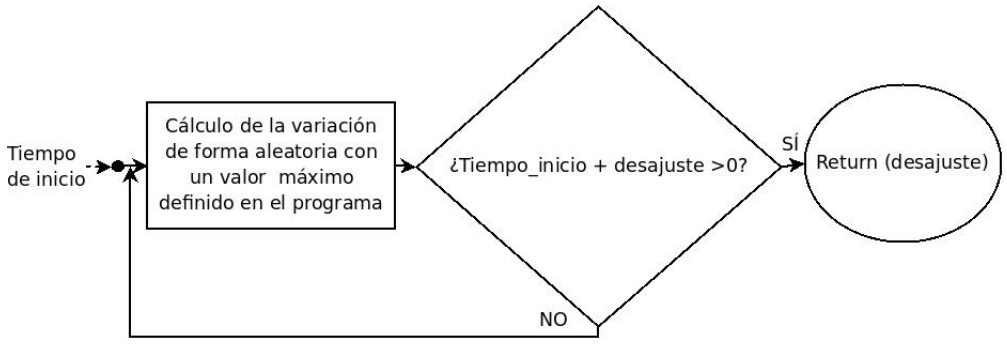


Figura 4.9: Diagrama de bloques de la función de creación del desajuste temporal.

suma del parámetro de entrada y el valor aleatorio sean mayores que cero, para no obtener tiempos negativos.

Tras el cálculo del desajuste llamamos a dos funciones que incluyen las líneas de código relativas a los instrumentos, que son:

- Instrumento excitador
  El diagrama de bloques que describe esta función es el que sigue:

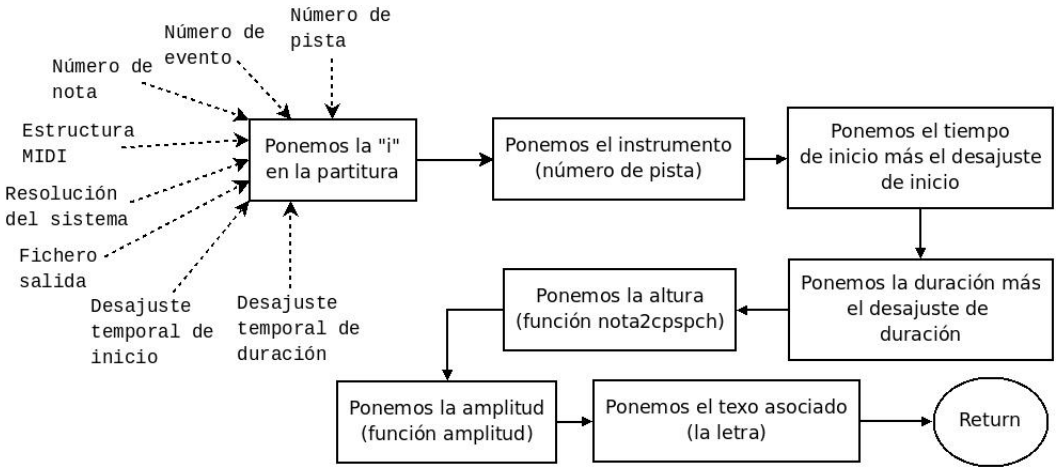


Figura 4.10: Diagrama de bloques de la función que crea la llamada al instrumento excitador desde la partitura.

La función recibe bastantes parámetros, que son:

  - El número de la pista en que estamos.
  - El número de evento en que nos encontramos.
  - El número de la nota actual.
  - El archivo MIDI.
  - La resolución del sistema.
  - El fichero de salida, es decir, la partitura.
  - El desajuste del tiempo de inicio.
  - El desajuste de la duración de la nota.

  Con esta función lo que crearemos será la llamada al excitador que corresponda según la voz para después filtrar esa señal con el instrumento de filtrado correspondiente. Para ello hemos de ir

poniendo los parámetros correspondientes según se haya diseñado el instrumento, los cuales serán, por este orden:

- La letra reservada **i** para la llamada a la orquesta.
- El número del instrumento que, por como se ha realizado el diseño del archivo MIDI de origen, coincide con el número de pista.
- El tiempo inicial (es el campo *inicio* dentro de la estructura *nota* de tipo TNota que está dentro de cada pista del archivo) más su desajuste correspondiente.
- La duración de la nota (campo *duracion* dentro de la estructura *nota* de tipo TNota que está dentro de cada pista del archivo) más el correspondiente desajuste.
- La altura del sonido, la cual calculamos con la función *nota2cpspch*, y que describiremos a continuación.
- La amplitud de la nota, la cual calculamos con una función externa, que posteriormente comentaremos, y que va asociada a la velocidad de la nota.
- La letra asociada, la cual encontramos en el campo *texto* de la estructura evento en la pista correspondiente, precedida de un símbolo de comentario para que el compilador la ignore.

A continuación, comentaremos la función *nota2cpspch*, cuyo funcionamiento podemos ver en el pseudocódigo 1.

Esta función tiene la finalidad de transformar una nota dada en notación MIDI a *octava.nota*[3]. Para realizar esto, aunque se puede ver claramente en el pseudocódigo, vamos a ir describiendo el proceso para clarificarlo:

- Tomamos como nota de referencia la nota MIDI 60 ($DO_3$ en música) que se corresponde con la nota 8.00 en el formato de *octava.nota*.
- Calculamos la distancia entre la nota propuesta y la referencia (simplemente una resta).
- A partir de este momento, aunque diferenciemos entre un caso de distancia positiva y negativa, la filosofía es la misma: calculamos las octavas de distancia (dividiendo la distancia entre

[3]Formato de notación musical en que las notas se especifican como la octava en que se encuentran seguida de la nota en cuestión. Por ejemplo, $FA\sharp_5$ se representa como 10.06 y $RE\flat_2$ es 7.01

**Algoritmo 1** Pseudocódigo de la función *nota2cpspch*.

**Entrada:** Variable *nota* con la nota en valor MIDI.
**Salida:** Variable *salida*.

```
1: salida = 0
2: resta = nota - 60
3: Si resta>0 Entonces
4:    octavas = resta/12
5:    semitonos = resta - octavas · 12
6:    Bucle desde 0 hasta octavas
7:       salida = salida + 1
8:    Fin Bucle
9:    salida = salida + (float)semitonos/100
10: Else
11:   resta = -1 · resta
12:   octavas = resta/12
13:   semitonos = resta - octavas · 12
14:   Bucle desde 0 hasta octavas
15:      salida = salida - 1
16:   Fin Bucle
17:   Si semitonos>0 Entonces
18:      salida = salida - 1
19:      semitonos = 12 - semitonos
20:      salida = salida +(float)semitonos/100
21:   Fin Si
22: Fin Si
23: Return salida
```

12), aumentando posteriormente la referencia esas mismas veces; calculamos los semitonos (resto de la diferencia entre la distancia total y las octavas) y los subimos a la referencia.

- Devolvemos el valor.

Por último, la función que permite calcular la amplitud de la nota es la llamada *amplitud*, siendo su diagrama de bloques el que muestra la figura 4.11.

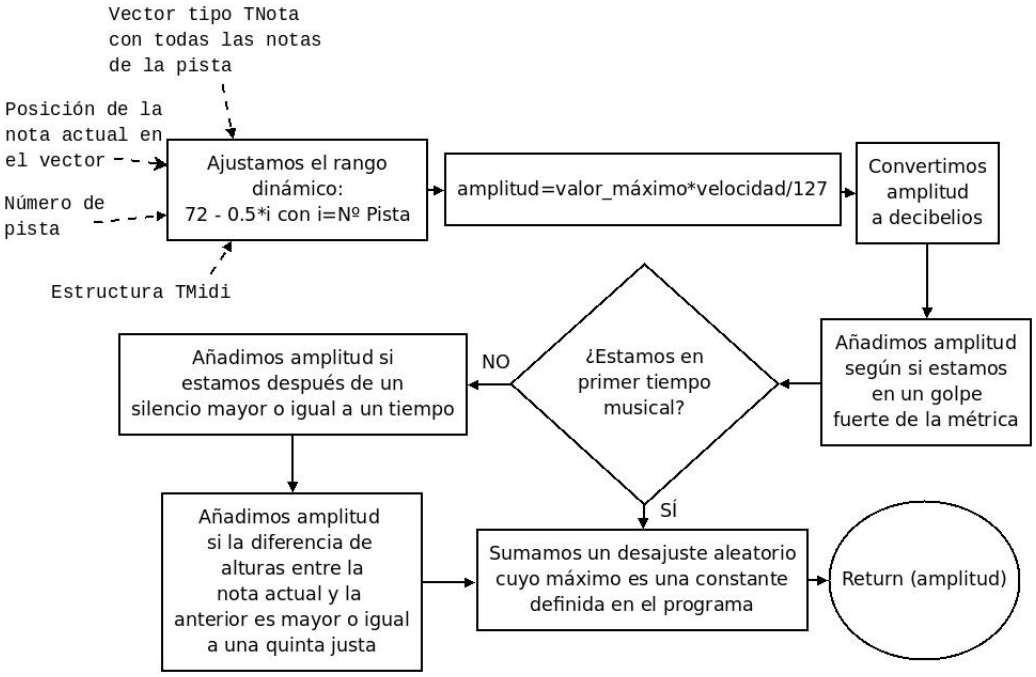


Figura 4.11: Función de cálculo de la amplitud de la nota.

Como podemos ver, a esta función llegan cuatro parámetros:

- Un vector de tipo TNota con todas las notas que contiene la pista.
- La posición de la nota que estamos tratando en el vector.
- El número de la pista en que nos encontramos.
- La estructura TMidi principal.

A continuación describiremos cada uno de los pasos llevados en esta función para una total comprensión de la misma:

1. Ajuste del rango dinámico
   En esta primera parte de la función lo que hacemos es limitar la amplitud máxima de serie que puede tener una nota en función de la pista en la que se encuentre. Para ello simplemente establecemos la operación $72 - 0{,}5 \cdot$ Número de pista, lo que asegura que el la pista de la voz soprano el rango máximo inicial[4] sea 72 dB y el del bajo sea 70.5 dB para que, al menos de partida, las voces graves no enmascaren las agudas.

2. Obtención del valor de amplitud de la nota
   Con el valor máximo de amplitud obtenido antes, y por medio de una regla de tres[5], obtenemos la amplitud en PCM, la cual pasamos después a decibelios.

3. Amplitud según métrica
   Con esta función lo que tratamos de implementar es la interpretación que se da a la métrica en las partituras: con la métrica definimos una serie de tiempos fuertes y tiempos débiles[6] dentro de cada compás de la composición. Por ello, conociendo la métrica de la propia composición se ha implementado una función que detecte en qué tipo de tiempo nos encontramos y, en el caso de encontrarnos en uno fuerte, se añada cierta cantidad de amplitud.

   De una forma muy básica, obviando bastantes problemas que se encuentran asociados al algoritmo que se implementa aquí y que después comentaremos, lo que se realizrá para llevar a cabo la tarea de encontrar el tiempo fuerte es una realizar una división entre cada uno de los tiempos en los que se encuentran las notas y el denominador de la métrica del compás, asumiendo que cuando obtengamos un cero tenemos el tiempo fuerte, o uno de ellos, del compás en que nos encontramos.

   La función que implementa esto viene descrita por el pseudocódigo 2.

[4]Decimos inicial porque en las siguientes funciones puede que se añada amplitud, siendo entonces el rango dinámico superior al máximo del que aquí hablamos.

[5]La regla de tres es ampltiud_resultante = valor_máximo(PCM) $\cdot$ velocidad/127.

[6]En realidad existen también otro tipo de tiempos como los medio–fuertes, pero estamos realizando una primera aproximación.

**Algoritmo 2** Pseudocódigo de la función *amplitud_metrica*.

**Entrada:** Estructura TNota de Nota actual, Resolución del sistema, Estructura TMidi y puntero a entero *primer_tiempo.

**Salida:** Variable *salida*.

1: **Llamada a función** *extraer_numyden* (extraer numerador, denominador y cantidad de fusas por beat del compás)
2: factor_bajada = Inicio Nota/resolución/numerador
3: Inicio Nota = Inicio Nota − factor_bajada · numerador · resolución
4: división = Inicio Nota % (int)(denominador · (int)resolución · *corrector_fpb*)
5: **Si** división == 0 **Entonces**
6: salida = 6
7: **Si** Inicio Nota == 0 **Entonces**
8: *primer_tiempo = 0
9: **Fin Si**
10: **Else**
11: salida = 0
12: **Fin Si**
13: **Return** salida

Con este código lo que realmente estamos implementando es el siguiente razonamiento:

*a*) Conseguir el numerador, denominador y cantidad de fusas por beat del compás en el que nos encontramos, lo cual se consigue por medio de la función *extraer_numyden* que comentaremos después.

*b*) Bajar la numeración de tiempos del compás a una serie de módulo el numerador del compás[7]. Esto lo conseguimos obteniendo el resto de la división del tiempo métrico entre el valor del numerador.

*c*) Con la ayuda del denominador y del número de fusas por beat, obtenemos si estamos en el tiempo fuerte del compás[8] o en cualquier otro punto[9].

*d*) En caso de estar en el primer tiempo añadimos cierta cantidad de amplitud y ponemos el puntero **primer_tiempo*

[7]Es decir, que la numeración de tiempos del compás tenga, como máximo, el número de la métrica del compás, para así facilitar los cálculos.

[8]Recordemos que, en nuestra aproximación, para que esto ocurra, el resultado de la división de módulo el denominador ha de ser cero.

[9]El resultado de la división será distinta de cero.

a 1. Si no estamos en el primer tiempo no añadimos amplitud ni cambiamos el valor del puntero.

De lo anteriormente comentado, se puede entender todo menos la razón del puntero **primer_tiempo*. Para simplificar más el sistema, este puntero hace que siempre que nos encontremos en un primer tiempo de compás, el resto de opciones para añadir amplitud (tras un silencio y tras una subida de altura) no se tengan en cuenta. Las razones que han llevado a esta decisión son las que siguen:

*a*) Por un lado, no tener una nota con mucha amplitud si es que, por un casual, se dieran todas las situaciones a la vez.

*b*) Por otro lado, intentar evitar la situación problemática que puede dar la primera notas de todas las de una pista: en caso de tener que comprobar la subida de altura o el tiempo transcurrido tras un silencio necesitaríamos datos de una nota anterior, la cual no existiría, por lo que o bien se produciría un error de desbordamiento o bien se cogerían valores residuales de memoria, con lo cual no se obtendría un resultado correcto.

Una vez comentado todo el razonamiento de esta función, pasaremos a comentar las funciones auxiliares que necesita este algoritmo para poder funcionar.

La primera función que necesita es la denominada *extraer_numyden*, la cual podemos representar con el pseudocódigo 3.

El objetivo de esta función, como se ha comentado antes, es el de obtener el numerador y el denominador de la métrica del compás en el que se encuentra la nota ya que no tiene necesariamente que ser el mismo que se da al inicio de la partitura.

El razonamiento que se sigue para la extracción de estos datos es el que sigue:

*a*) Lo primero que hacemos es, sabiendo la cantidad de métricas que tenemos (es el campo *c_metrica* dentro de la estructura *descriptores* que hay en TMidi), reservamos memoria para cuatro vectores de tipo entero que contendrán

**Algoritmo 3** Pseudocódigo de la función *extraer_numyden*.

**Entrada:** Estructura TMidi, Estructura TNota de Nota actual y punteros a entero *numerador *denominador y *fpb.

**Salida:** Numerador, denominador y cantidad de fusas por beat por referencia.

1: Extraemos el tiempo de inicio de la nota
2: Extraemos la cantidad de métricas
3: **Reservamos Memoria** de cuatro vectores de tipo entero con longitud la cantidad de métricas (*vector_numeradores, vector_denominadores, vector_pulsos, vector_fpb*)
4: Extraemos numeradores, denominadores, pulsos y fusas por beat
5: Buscamos la zona en la que nos encontramos (con los pulsos) y extraemos el numerador, el denominador y la cantidad de fusas por beat
6: **Liberamos Memoria**
7: **Return**

los numeradores, denominadores, instantes de inicio de las métricas y la cantidad de fusas por beat de toda la partitura.

*b*) Por medio del campo *compases* dentro de la estructura *descriptores* de TMidi, y sabiendo que la forma en que almacena los datos sobre la métrica en la composición es *Numerador*/*Denominador*(*Pulso Inicio*)|*Fusas por beat*[10], extraemos todos los valores de métrica de la partitura.

*c*) Sabiendo el instante en el que nos encontramos, extraemos los datos de métrica que nos interesan.

*d*) Finalmente, liberamos la memoria dinámica utilizada.

Como último apunte acerca de esta función, cabe destacar que, desde un punto de vista de eficiencia computacional, no es totalmente correcta ya que para cada una de las notas de toda la partitura estamos volviendo a obtener todos los datos de la métrica de la misma cuando, en realidad, con realizarlo una vez bastaría. Sin embargo, finalmente no se modificó la función porque el tiempo de ejecución del programa no se veía afectado y por intentar conseguir mayor limpieza en el código.

[10]En realidad, en un principio la estructura de este campo era *Numerador*/*Denominador*(*Pulso Inicio*) pero se tuvo de modificar para poder obtener el número de fusas por beat.

Finalmente, la última función necesaria para realizar el ajuste de amplitud en función de la métrica, es la llamada *corrector_fpb*, cuyo funcionamiento encontramos en el pseudocódigo 4.

**Algoritmo 4** Pseudocódigo de la función *corrector_fpb*.

**Entrada:** Variable *valor_fpb* con las fusas por beat y variable *denominador* el denominador del compás.

**Salida:** Variable *valor_salida*.

```
1: posición = log2(valor_fpb)
2: Bucle desde 0 hasta 5
3:    vector_fusasporbeat[i] = 2^(6−i−1)
4: Fin Bucle
5: valor_salida=vector_fusasporbeat[posición]/denominador
6: Si valor_salida < 1 AND denominador<vector_fusasporbeat[posición]
   Entonces
7:    valor_salida=1/valor_salida
8: Fin Si
9: Return valor_salida
```

La base de esta función es la de crear un factor corrector que se aplique al denominador de la métrica del compás a la hora de realizar la división que establece si estamos en un tiempo fuerte o no. La razón de la necesidad de este factor corrector es la diferencia entre la medida de los tiempos entre el formato de la métrica y el número de fusas por beat[11] que el archivo MIDI establece, ya que en ocasiones, y simplemente como ejemplo ya que podríamos encontrar muchos más ejemplos, el denominador establece la métrica en corcheas (valor de denominador 8, lo que implicaría que por cada beat tendríamos 4 fusas) y el número de fusas por beat es 8.

[11]Es la forma que tiene MIDI de cuantificar los tiempos musicales y se basa en que, por cada tiempo, tienen que haber una serie de fusas que pueden ser desde valores normales de nota (8 fusas son una negra, 32 fusas una blanca...) o incluso notas con alteraciones (12 fusas es una negra con puntillo, 6 fusas una corchea con puntillo...), aunque en este algoritmos sólo se ha considerado el primero de los dos casos (si tenemos una métrica del segundo, el algoritmo, previsiblemente, funcionará, pero no dará bien la medida de los tiempos).

El razonamiento seguido con esta función es el que sigue:

*a*) La base de este algoritmo es la tabla siguiente, la cual devuelve la relación entre el valor de fusas por beat y el que debería corresponder con el denominador de la métrica. Cabe destacar que en el programa esto se ha implementado como un único vector cuyas posiciones vienen dadas por el exponente de dos que crea el número de fusas por beat que se busca.

| $\log_2$(Fusas_por_beat) | Valor de la nota de referencia en formato denominador de compás |
|---|---|
| 0 | 32 |
| 1 | 16 |
| 2 | 8 |
| 3 | 4 |
| 4 | 2 |
| 5 | 1 |

*b*) El factor corrector lo obtenemos como el cociente entre el valor devuelto por la tabla anterior y el denominador de la métrica del compás.

*c*) Finalmente, el factor corrector recién calculado será el correcto siempre que el denominador de la métrica sea mayor que el valor devuelto por la tabla, si no deberemos obtener el valor inverso del factor recién calculado, siendo entonces ese valor el factor corrector.

4. Amplitud según si estamos después de un silencio
Para realizar esta tarea se ha implementado la función *amplitud_silencio*, la cual se rige por el siguiente pseudocódigo 5.

Con esta función, como ya se ha dicho antes, lo que hacemos es añadir cierta amplitud más a la señal, concretamente 4 dB[12] siempre que haya un silencio de un tiempo o más entre dos notas. Esta implementación se justifica por el hecho de que normalmente, tras un silencio en el que el cantante descansa lo suficiente, a la hora de comenzar a cantar de nuevo se aumenta, aunque sea involuntariamente, la fuerza de la voz.

[12]Esta amplitud se estableció por medio de ensayo y error.

**Algoritmo 5** Pseudocódigo de la función *amplitud_silencio*.

**Entrada:** Estructuras TNota de Nota actual y Nota anterior y Resolución del sistema.

**Salida:** Variable *Salida*.

```
1: Si Duracion Nota_anterior + Tiempo Inicio Nota_anterior ⩽ Tiempo Inicio Nota Actual - 1 Entonces
2:    Salida = 4
3: Else
4:    Salida = 0
5: Fin Si
6: Return valor_salida
```

5. Amplitud según una subida de altura relativamente grande
   Esta tarea se ha implementado en la función *amplitud_dif_altura*, la cual se puede ver plasmada en el siguiente pseudocódigo 6.

**Algoritmo 6** Pseudocódigo de la función *amplitud_dif_altura*.

**Entrada:** Estructuras TNota de Nota actual y Nota anterior y Resolución del sistema.

**Salida:** Variable *Salida*.

```
1: Si Altura Nota_actual - Altura Nota_anterior ⩾ 7 Entonces
2:    Salida = 4
3: Else
4:    Salida = 0
5: Fin Si
6: Return valor_salida
```

La finalidad de esta función es imitar un efecto que suele ocurrir en el canto[13]: al realizar un cambio relativamente brusco de altura entre dos notas, y sobretodo cuando vamos de grave a agudo, se suele dar un poco más de energía a la nota a la que se pretende llegar. En nuestro caso, hemos considerado que ese cambio brusco de altura ya ocurre con un cambio de quinta justa[14] y que la amplitud que se añade es de 4 dB[15].

[13] Aunque ocurra a menudo, esto no implica que sea correcto.

[14] Una quinta justa son siete semitonos.

[15] Como en el caso anterior, este dato ha sido obtenido por medio de ensayo y error.

6. Desajuste aleatorio de amplitud
   Para la obtención simplemente llamamos a una función de obtención de valores aleatorios[16] y la limitamos para obtener un valor que se sitúe entre [-Amp_Max,+Amp_Max], siendo Amp_Max la amplitud de pico permitida y que se encuentra declarada, mediante un *define* en el propio programa *midi2letra*.

- Instrumento filtro
  El diagrama de bloques que define esta función es el que muestra la figura 4.12.

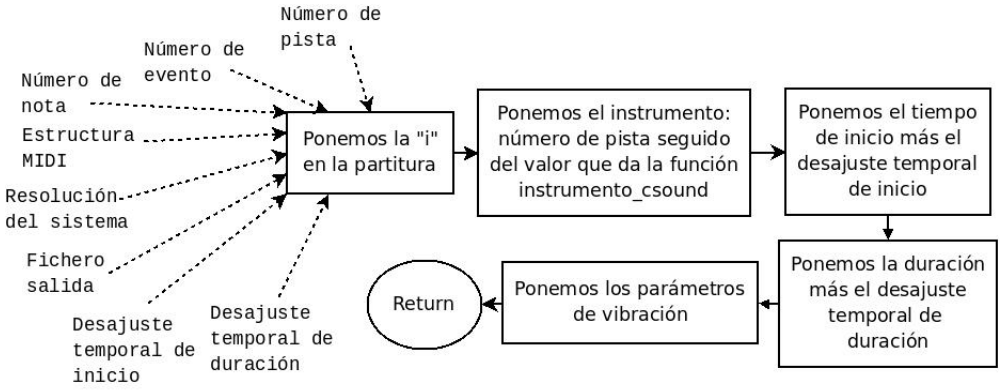


Figura 4.12: Diagrama de bloques de la función que crea la llamada al instrumento de filtrado desde la partitura.

Como podemos observar, los parámetros que recibe esta función son los mismos que los de la función anterior, por lo que no se hace necesario el comentarlos. Sin embargo, mientras que en la anterior hacíamos una llamada a un instrumento de tipo excitador, en este caso vamos a llamar al instrumento que se encargará de realizar el filtrado de la señal antes creada y así darle la forma, espectralmente hablando, que queremos.

Como en el caso anterior, vamos a describir los parámetros que necesitará la llamada al instrumento de filtrado correspondiente:

- La letra reservada **i** para la llamada a la orquesta.

[16] Concretamente la función **rand()** del lenguaje C.

- El instrumento correspondiente, el cual será un número que represente la voz que estemos tratando (es decir, el número de pista según nuestro diseño) seguido de otro número en función de la vocal a cantar, lo cual hacemos con la función *instrumento_csound* que comentaremos después.
- Ponemos el tiempo de inicio de la misma forma que en la función del instrumento excitador.
- Ponemos la duración de la nota de la misma forma que en la función del instrumento excitador.
- Ponemos los parámetros de vibración, los cuales son estáticos salvo uno que es la duración de la nota que podemos saber haciendo lo mismo que en el parámetro anterior.

Finalmente, vamos a comentar la función *instrumento_csound*, la cual se rige según el pseudocódigo 7.

**Algoritmo 7** Pseudocódigo de la función *instrumento_csound*.

**Entrada:** Variable *letra* con la vocal a evaluar.
**Salida:** Variable *instr*.

```
1: instr = 0
2: Si letra == 'a' Entonces
3:    instr = 1
4: Else, Si letra == 'e' Entonces
5:    instr = 2
6: Else, Si letra == 'i' Entonces
7:    instr = 3
8: Else, Si letra == 'o' Entonces
9:    instr = 4
10: Else, Si letra == 'u' Entonces
11:    instr = 5
12: Fin Si
13: Return instr
```

El único parámetro que pasamos a la función es la letra asociada a la nota a procesar. Dentro de la función simplemente vemos qué vocal de las posibles tenemos para devolver el número asociado a la misma, el cual hace referencia al instrumento que se pretende ejecutar para sintetizar los formantes de la misma.

## 4.3. Ejemplo de funcionamiento

Finalmente, tras la creación del programa que realiza la transformación de archivo MIDI a partitura de CSound, vamos a mostrar un ejemplo del resultado que se obtiene.

La partitura, en notación clásica, del archivo MIDI que será procesado la encontramos en la figura 4.13.

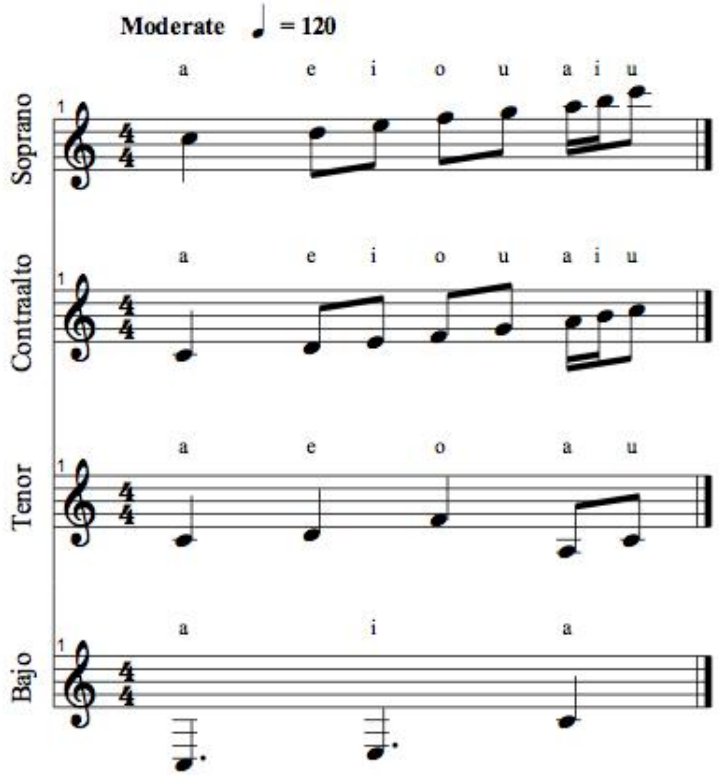


Figura 4.13: Partitura de ejemplo creada con el programa GuitarPro de Arobas Music.

Esta partitura, creada en un secuenciador MIDI, tendría la forma mostrada en la figura 4.14.

Cabe destacar que, aunque en la partitura hay sólo 4 voces distintas, en el secuenciador hemos creado 5 pistas porque la primera, como se ha explicado, nuestra implementación únicamente la examina en busca de metadatos de

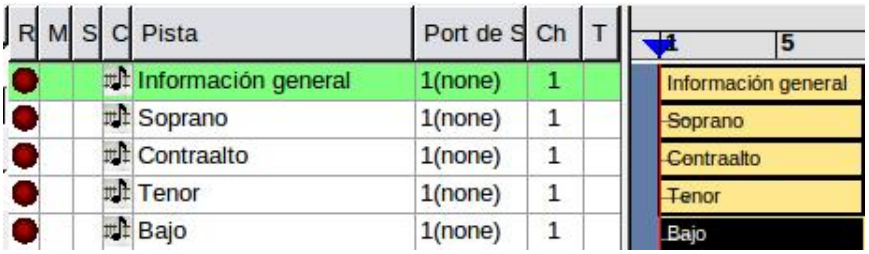


Figura 4.14: Detalle de las pistas creadas en el secuenciador MIDI MusE, distribuido bajo licencia GPL.

información general.

Dentro de cada una de las pistas que contienen las diferentes voces, las notas se han secuenciado con el formato de diagrama de pianola, quedando tal cual se muestra en la figura 4.15.

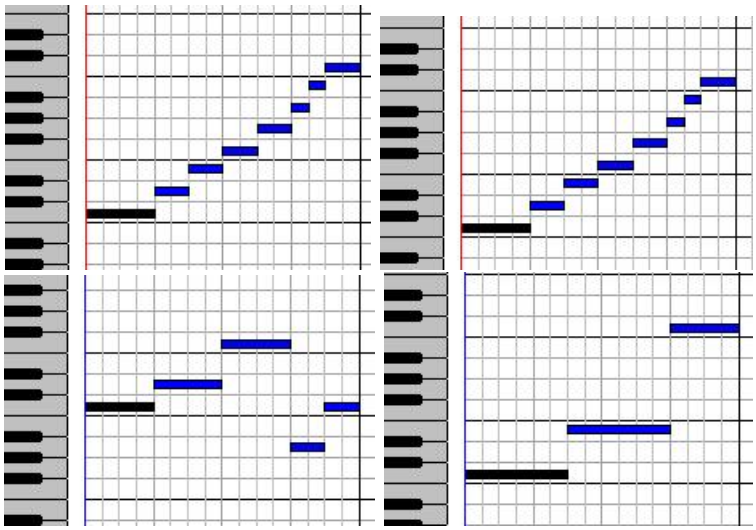

Figura 4.15: Diagramas de pianola de las cuatro voces creados con el secuenciador MusE: Soprano (superior izda.), Contralto (superior dcha.), Tenor (inferior izda.) y Bajo (inferior dcha.).

Respecto a los diagramas anteriores, parece que la secuencia creada para la voz soprano y la de contralto son iguales. En realidad, como se puede ver en la partitura, sí que son las mismas notas pero a diferente altura (la voz

soprano está cantando una octava por arriba de la voz contralto). Esto, sin embargo, no se puede observar con facilidad en los extractos de los diferentes diagramas de pianola porque el secuenciador utilizado no incluye los nombres en las diferentes notas en el teclado.

El resultado que devuelve el programa de la secuencia MIDI anterior es el que sigue:

```
;Datos presentes en el archivo midi acerca del contenido del mismo:
;Copyright: Jose Javier Valero Mas
;Texto: Secuencia creada para comprobar el programa de traduccin MIDI-CSound

;Partitura obtenida del archivo
;Cambios de tempo:
t 0 120 ; Hay 1 tempo(s)

f1 0 8192 10 1
f2 0 8192 9 1 1 .25
i99 0 9 0.95 0.95 0.8 2000 1


;Soprano
;Instrumento    Tini            TDur            Frecuencia      Amplitud        Letra
i       1       0.000000        1.000000        9.000000        75.41           ;/a/
i       11      0.000000        1.000000        1               10              1.000000

i       1       1.000000        0.500000        9.020000        75.41           ;/e/
i       12      1.000000        0.500000        1               10              0.500000

i       1       1.500000        0.500000        9.040000        69.41           ;/i/
i       13      1.500000        0.500000        1               10              0.500000

i       1       2.000000        0.500000        9.050000        75.41           ;/o/
i       14      2.000000        0.500000        1               10              0.500000

i       1       2.500000        0.500000        9.070000        69.41           ;/u/
i       15      2.500000        0.500000        1               10              0.500000

i       1       3.000000        0.250000        9.090000        75.41           ;/a/
i       11      3.000000        0.250000        1               10              0.250000

i       1       3.250000        0.250000        9.110000        69.41           ;/i/
i       13      3.250000        0.250000        1               10              0.250000

i       1       3.500000        0.500000        10.000000       69.41           ;/u/
i       15      3.500000        0.500000        1               10              0.500000


;Contraalto
;Instrumento    Tini            TDur            Frecuencia      Amplitud        Letra
i       2       0.000000        1.000000        8.000000        74.91           ;/a/
i       21      0.000000        1.000000        1               10              1.000000

i       2       1.000000        0.500000        8.020000        74.91           ;/e/
i       22      1.000000        0.500000        1               10              0.500000

i       2       1.500000        0.500000        8.040000        68.91           ;/i/
i       23      1.500000        0.500000        1               10              0.500000
```

```
i       2       2.000000        0.500000        8.050000        74.91           ;/o/
i       24      2.000000        0.500000        1               10              0.500000

i       2       2.500000        0.500000        8.070000        68.91           ;/u/
i       25      2.500000        0.500000        1               10              0.500000

i       2       3.000000        0.250000        8.090000        74.91           ;/a/
i       21      3.000000        0.250000        1               10              0.250000

i       2       3.250000        0.250000        8.110000        68.91           ;/i/
i       23      3.250000        0.250000        1               10              0.250000

i       2       3.500000        0.500000        9.000000        68.91           ;/u/
i       25      3.500000        0.500000        1               10              0.500000


;Tenor
;Instrumento    Tini            TDur            Frecuencia      Amplitud        Letra
i       3       0.000000        1.000000        8.000000        74.41           ;/a/
i       31      0.000000        1.000000        1               10              1.000000

i       3       1.000000        1.000000        8.020000        74.41           ;/e/
i       32      1.000000        1.000000        1               10              1.000000

i       3       2.000000        1.000000        8.050000        74.41           ;/o/
i       34      2.000000        1.000000        1               10              1.000000

i       3       3.000000        0.500000        7.090000        74.41           ;/a/
i       31      3.000000        0.500000        1               10              0.500000

i       3       3.500000        0.500000        8.000000        68.41           ;/u/
i       35      3.500000        0.500000        1               10              0.500000


;Bajo
;Instrumento    Tini            TDur            Frecuencia      Amplitud        Letra
i       4       0.000000        1.500000        7.000000        73.91           ;/a/
i       41      0.000000        1.500000        1               10              1.500000

i       4       1.500000        1.500000        7.040000        67.91           ;/i/
i       43      1.500000        1.500000        1               10              1.500000

i       4       3.000000        1.000000        8.000000        73.91           ;/a/
i       41      3.000000        1.000000        1               10              1.000000

e
```

Como podemos observar, un simple compás de cuatro tiempos, en CSound requiere una gran cantidad de código para la orquesta de control, siendo por tanto una tarea bastante tediosa la redacción de una partitura para una composición directamente en el lenguaje de programación frente a la relativa sencillez y rapidez del método de un secuenciador MIDI. Cabe destacar, como detalle final, que la secuencia MIDI creada tiene realmente 5 pistas, siendo la primera de todas la correspondiente a la información general (que es la que el programa extrae y pone en las primeras líneas de la partitura) seguida de otras 4, que ya son realmente las voces del coro.

Por último, cabe destacar que, en este caso y para una mejor comprensión de la transformación de MIDI a partitura, los valores aleatorios de tiempo y amplitud han sido anulados (lo cual se puede hacer sencillamente poniendo como valores máximos en la variación un cero).

# Capítulo 5

# Síntesis de fonemas consonánticos

En este capítulo de la memoria se describirán todas aquellas tareas que se han llevado a cabo para dar solución a nuestro problema de síntesis de fonemas, en especial los consonánticos. Cabe destacar que, en lo referente a este capítulo, se espera el poder dar solución a la síntesis de los fonemas consonánticos más comunes en nuestro idioma (por ejemplo, consonantes del tipo *k*, *m*, *r*, *s*...). Los fonemas vocálicos, al ya tener una solución posible por el hecho de poder ser creados por síntesis sustractiva, no son, en principio, un problema destacable en este apartado, aunque sí que se intentará buscar otra técnica que nos permita la síntesis de los mismos.

## 5.1. Método de los formantes

El primero de los métodos que se han llevado a cabo para la síntesis de diferentes consonantes ha sido el mismo que utiliza el sintetizador de vocales: un sintetizador de tipo sustractivo en el que únicamente necesitamos una señal de origen de banda ancha y, tras ello, una serie de filtros para modelar su espectro (es decir, para incluir los formantes necesarios).

Para poder partir de algo que ya se tenía, lo primero que se ha intentado sintetizar son consonantes de tipo nasal debido a que estas consonantes, en cierta forma, son de tipo sonoro y, por tanto, podemos utilizar el mismo sintetizador que se está utilizando para crear las vocales (utilizando el *opcode* **gbuzz** y los filtros **reson** y **resonz**). Posteriormente, con la intención de sintetizar sonidos de tipo sordo para la gran mayoría de consonantes, se utilizarán fuentes de ruido (*opcodes* del tipo **gauss**, **rand**, **randi**).

Tras este breve resumen vamos a estudiar cada una de las acciones llevadas a cabo:

- Primeros intentos
  Las primeras consonantes cuya síntesis se ha llevado a cabo son, como se ha dicho, consonantes de tipo nasal, es decir, las letras $m$ y $n$, debido a que eran las consonantes más parecidas a lo que ya se tenía (un sintetizador de vocales). El código correspondiente se incluye en el anexo de códigos de la presente memoria y los formantes que se han utilizado para el filtrado se incluyen en el anexo sobre formantes, en el apartado de consonantes.

  En la figura 5.1 podemos ver unos ejemplos sobre la síntesis de los dos sonidos antes citados.

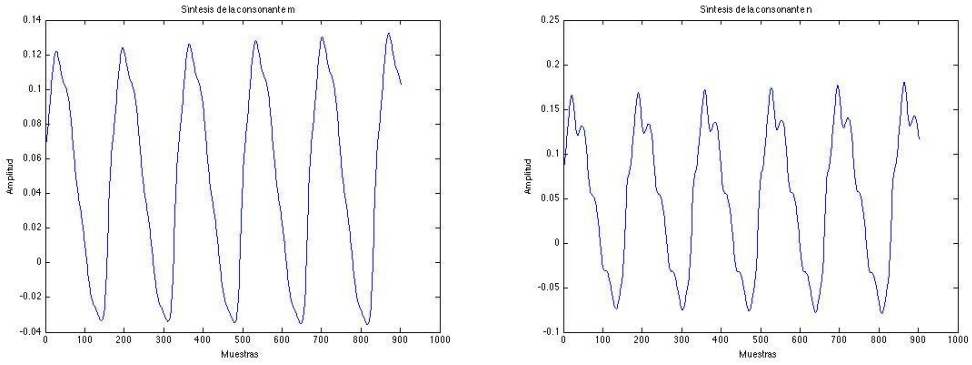


Figura 5.1: Ejemplos de formas de onda de la síntesis de la consonante $m$ (izda.) y de la consonante $n$ (dcha.).

Podemos ver que realmente el algoritmo funciona: la señal resultante es una señal periódica y, además, tienen diferente forma de onda por efecto del diferente filtrado de formantes que estamos realizando.

Como podemos observar en la figura 5.2, las ondas reales no son iguales a las sintetizadas, aunque sí que guardan cierta similitud (los picos más altos de las parejas de señales real-sintetizada sí que tienen una misma forma). Sin embargo, cabe destacar que no se podía pretender obtener una forma de onda sintetizada igual a la original porque, en definitiva,

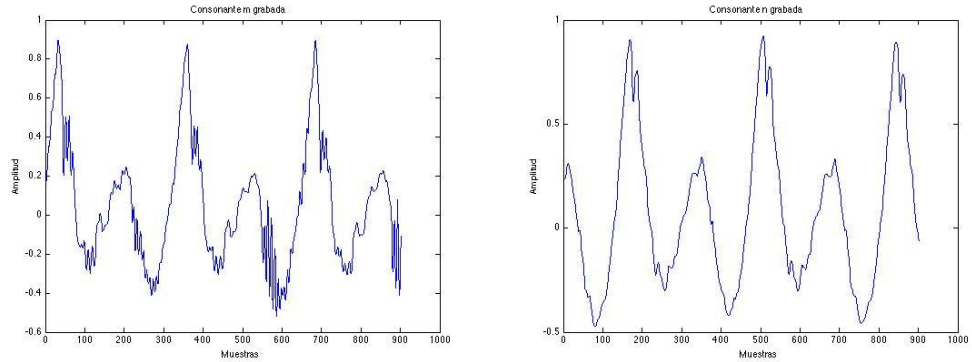


Figura 5.2: Tramos grabados de la consonante $m$ (izda.) y de la consonante $n$ (dcha.).

estamos modelando la consonante con una señal periódica de banda ancha y cinco filtros de formantes. Algo que, aunque pueda describir la señal de una forma general, no puede describir en todo su detalle todas las características de la misma.

Una vez realizadas estas consonantes se ha intentado dar solución, de la misma forma, a otros sonidos como son la $l$ o la $j$. En la figura 5.3 podemos ver un ejemplo del sonido de la $l$ sintetizado y su comparación con el real.

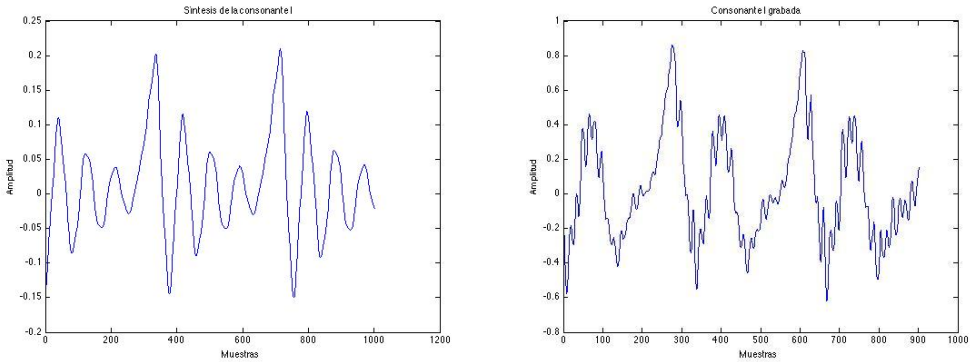


Figura 5.3: Consonante $l$: sintetizada (izda.) y grabada (dcha.).

También, en la figura 5.4 podemos ver el resultado de esto mismo con

la consonantes *j*, aunque auditivamente el resultado no sea tan bueno como en los anteriores.

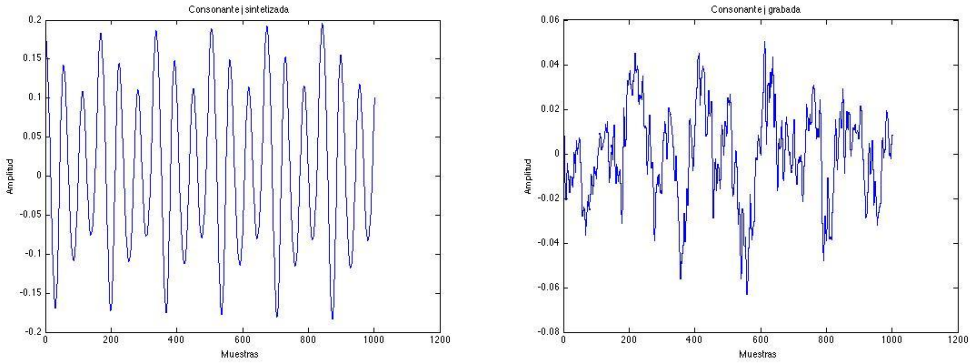


Figura 5.4: Consonante *j*: sintetizada (izda.) y grabada (dcha.).

- Síntesis de consonantes sordas
  Como hemos avanzado antes, tras la realización de pruebas con consonantes parecidas, en cierta forma, a las vocales (apartado anterior), se ha intentado dar solución al problema de la síntesis de consonantes sordas que, como ya sabemos, teóricamente se pueden modelar con ruido totalmente aleatorio.

  Para este caso se han realizado pruebas con distintos *opcodes* generadores de ruido, los cuales son, básicamente, **rand** (con sus variantes **randi** y **randh**) y **gauss**.

  El primer problema surgido al realizar estas pruebas es que, por acción de los filtros paso-banda utilizados para crear los formantes de la señal, el nivel obtenido de la misma ha sido muy alto. Por tanto se ha hecho necesario utilizar el *opcode* **balance** para conseguir que el nivel RMS de salida fuera el mismo que el de una señal sinusoidal de la amplitud que queremos de salida, la cual especificamos por medio de un parámetro al instrumento.

  Sin embargo, los resultados de esta técnica no han sido, para nada, satisfactorios debido, principalmente, a dos motivos:

1. El primero es que no se consigue distinguir una consonante bien definida tras realizar todo el proceso de síntesis sustractiva.
2. El segundo es que, al unir los sonidos sintetizados con las vocales ya sintetizadas, se nota un cambio muy brusco entre los tramos sordos y los sonoros, haciendo inviable un buen resultado.

En las figuras 5.5 y 5.6 podemos ver algunos de los resultados en la síntesis de algunas consonantes.

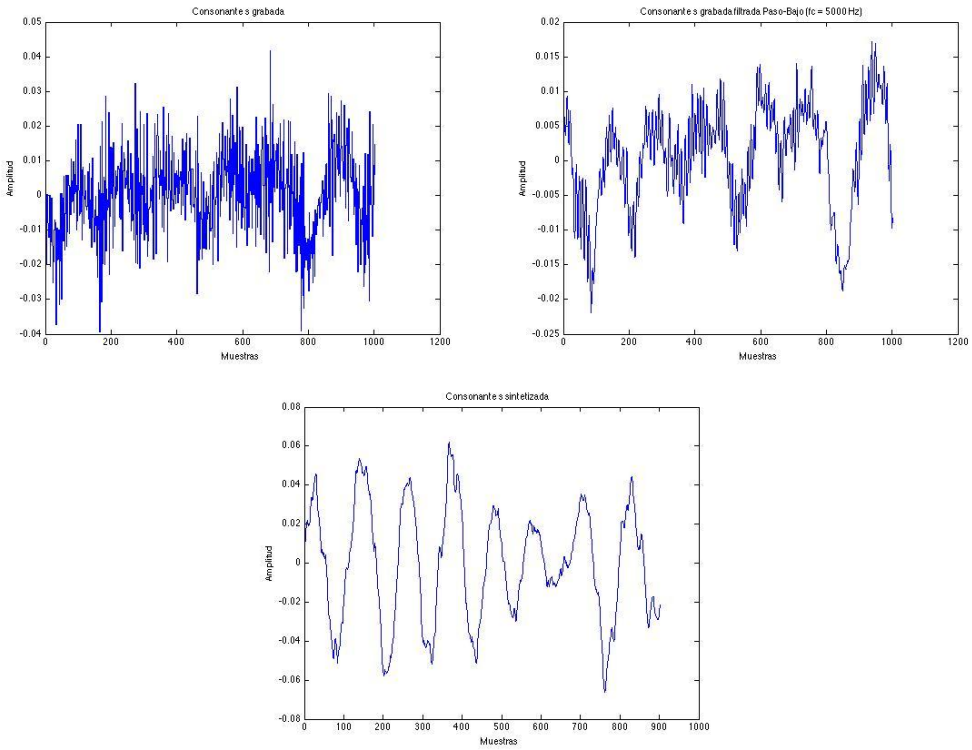


Figura 5.5: Consonante *s*: grabada (superior izda.), grabada con filtrado paso-bajo (superior dcha.) y sintetizada (inferior).

Como podemos observar, al haber utilizado un *opcode* de generación de ruido aleatorio (en este caso, el *opcode* **gauss** de generación de ruido blanco de distribución gaussiana), la señal sintetizada es una señal no periódica, que es precisamente la principal característica de la gran mayoría de consonantes.

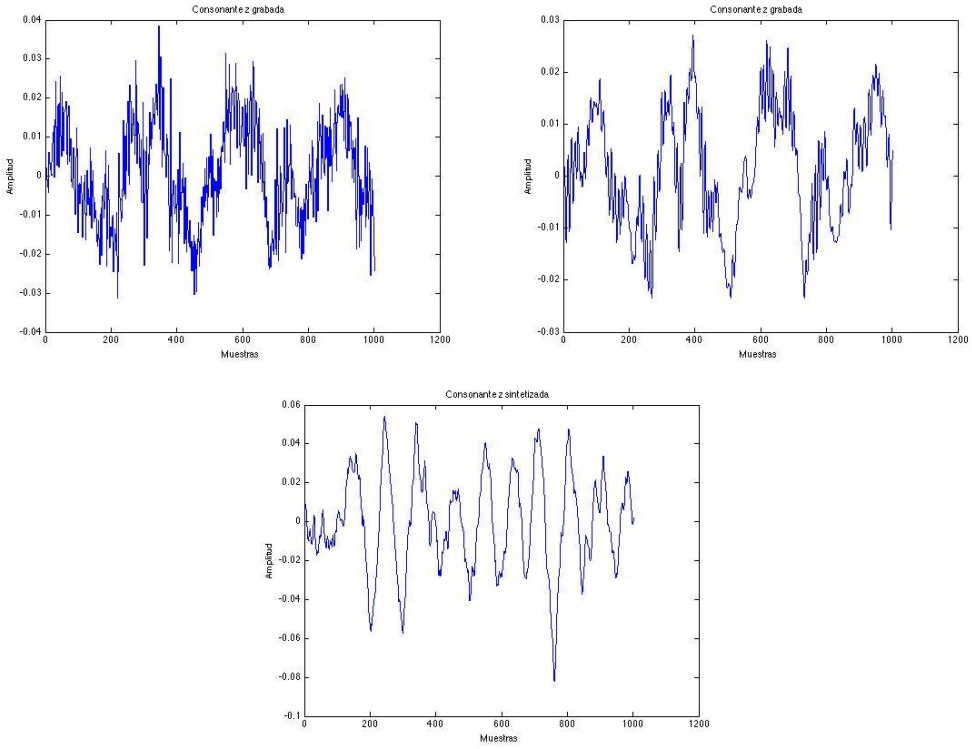


Figura 5.6: Consonante *z*: grabada (superior izda.), grabada con filtrado paso-bajo (superior dcha.) y sintetizada (inferior).

También, como modo de intentar eliminar esa cantidad de ruido excesiva, se ha llevado a cabo un filtrado de la señal, una vez ya con los formantes, con un filtro paso-bajo Butterworth (el *opcode* correspondiente es **butterlp**) y con una frecuencia de corte de 5000 Hz, por ser, más o menos, el límite superior al que llega el habla humana. Este proceso tampoco aporta gran resultado: la señal creada sigue siendo muy ruidosa aunque ahora carezca de tantas componentes de alta frecuencia. Como ejemplos de esto, podemos ver las figuras 5.7 y 5.8.

- Conclusiones
  Como principal idea, llegamos a la conclusión de que este tipo de síntesis funciona bien con señales de tipo periódico y, sobretodo, con señales cuyos formantes están muy bien definidos, lo que ocurre con señales con fase estacionaria (como las vocales o las consonantes de tipo nasal). Sin embargo, con señales de tipo ruidoso ocurre todo contrario: no se tiene información tan clara acerca de la estructura espectral de

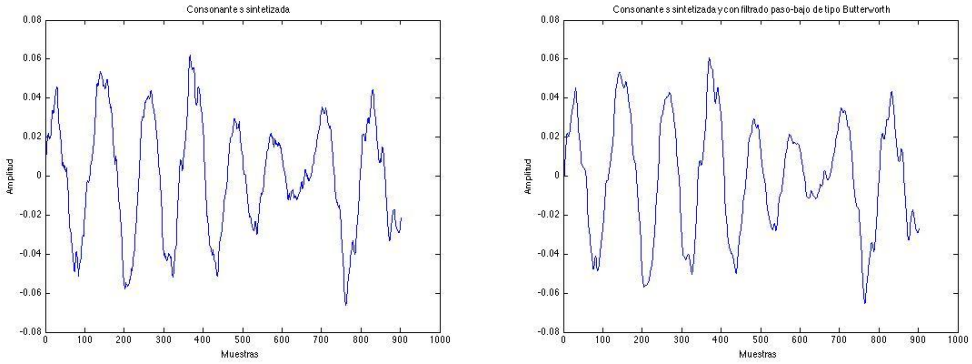


Figura 5.7: Consonante *s*: sintetizada normal (izda.) y sintetizada con filtrado paso-bajo (dcha.).

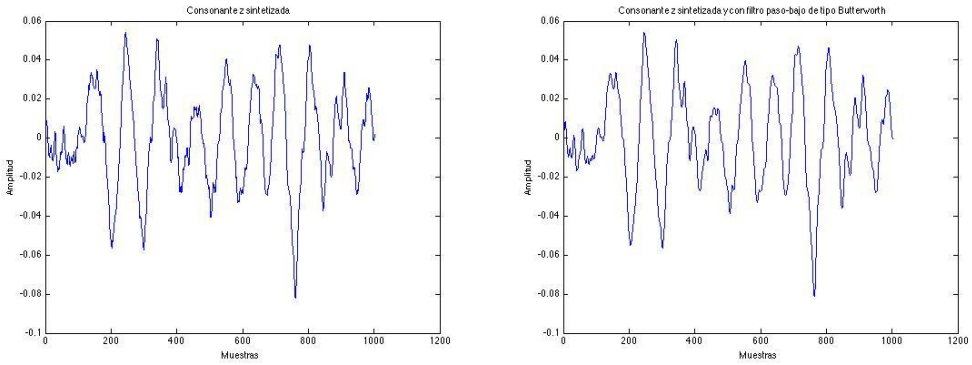


Figura 5.8: Consonante *z*: sintetizada normal (izda.) y sintetizada con filtrado paso-bajo (dcha.).

esa señal (es más difícil analizar los formantes de este tipo de señales al ser, por un lado, temporalmente muy cortos y, por otro lado, por tratarse, en general, de señales de tipo estocástico), con lo cual no se puede sintetizar algo de forma del todo fiable.

También cabe destacar que, en esta técnica de síntesis de sonido, la principal limitación es la señal de origen (la señal a filtrar) y, en este caso, las señales de ruido utilizadas dan bastantes problemas.

Por tanto, de cara a incluir elementos de este tipo en el sintetizador

final, se recomienda utilizar únicamente las consonantes de tipo nasal, las cuales, en cierta manera, dan un resultado aceptable.

## 5.2. Método LPC (Codificación Lineal Predictiva)

La siguiente opción a probar es la utilización de la codificación LPC, la cual es muy empleada en voz. Si bien se ha incluido un anexo[1] en el que se detalla bastante en qué consiste esta técnica y sobre cómo se realiza su desarrollo, para introducirlo diremos que, en líneas generales, la LPC es un método para la obtención de un filtro, variable en el tiempo, cuya función es imitar el aparato fonador de una persona (filtro del tracto vocal) y luego poder resintetizar la voz utilizando ese filtro con unas señales de excitación de banda ancha de tipo ruidoso o peridico. Precisamente por el hecho de que este método obtiene por sí mismo los coeficientes del filtro necesarios para sintetizar de nuevo la señal y no requiere mayor cantidad de datos, es un método bastante fiable. Para más información se recomienda la lectura del apéndice correspondiente.

Retomando el presente proyecto, CSound incluye todas las rutinas de procesado mediante LPC ya programadas como *opcodes*, los cuales son **lpanal**, **lpread** y **lpreson/lpfreson**, cuya explicación se incluye en el Apéndice A.

La forma en que se ha ido experimentado con este método ha sido realizando grabaciones de distintos fonemas y/o palabras para después realizar un análisis LPC con la función **lpanal** (guardando el fichero que devuelve, el cual es el que contiene todos los coeficientes del filtro que se ha obtenido) y más tarde, en el instrumento correspondiente, abrir ese fichero mediante la función **lpread** y, finalmente, procesar la señal de origen con el *opcode* **lpreson/lpfreson**.

Sin embargo, aunque puede que se mejorara algo en lo relacionado a los formantes de la señal (al calcularlos el propio algoritmo el resultado será más preciso que cualquier otro análisis), el problema sigue siendo el mismo que en el caso anterior, es decir, la señal a filtrar. Como la síntesis en este método vuelve a ser una síntesis sustractiva, si el método anterior fallaba precisamente por eso, este también lo hará.

[1] El anexo en cuestión es el Apéndice F.

En la figura 5.9 podemos ver algunos ejemplos de esta forma de síntesis.

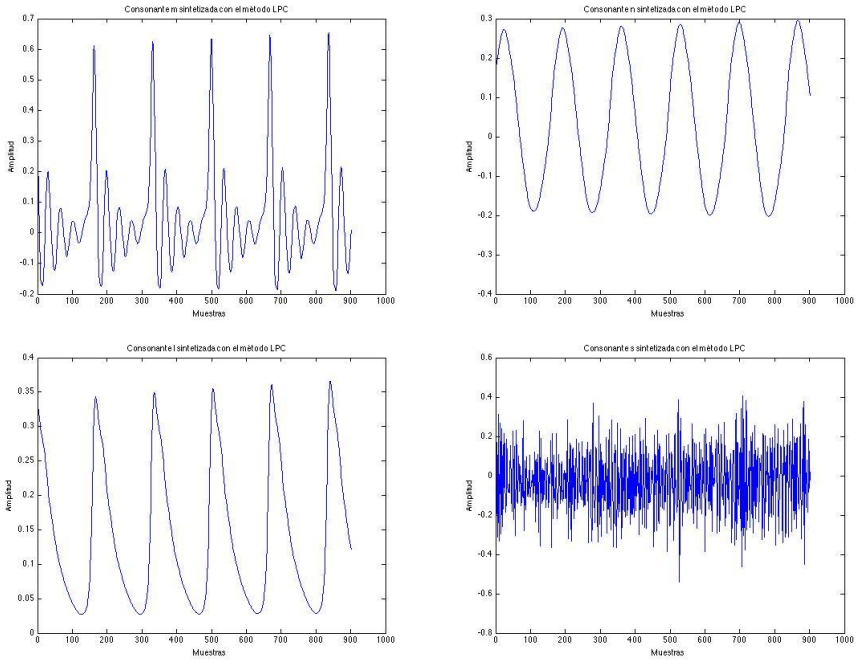


Figura 5.9: Síntesis, por el modelo LPC, de las consonantes $m$ (superior izda.), $n$ (superior dcha.), $l$ (inferior izda.), $s$ (inferior dcha.).

Como algo más a añadir a esta forma de síntesis podemos mostrar el resultado de este proceso sobre las diferentes vocales del castellano, cuyos resultados se encuentran en las figuras 5.10, 5.11, 5.12, 5.13 y 5.14.

En resumen, y como conclusión, esta técnica tiene una gran ventaja respecto a otras en el aspecto de que no tenemos que caracterizar nosotros la señal, es decir, no tenemos que buscar nosotros mismos los formantes a imitar, sino que de ello ya se encarga un algoritmo, que usualmente es más preciso en este tipo de tareas. Sin embargo todo esto tiene su precio y es que, olvidándonos de todo el proceso de grabar las muestras de sonido necesarias, el tiempo de cálculo que necesita el ordenador para compilar orquesta y partitura, por el hecho de ser filtros bastante complejos y de utilizar ficheros externos en los que estn almacenados los coeficientes para los mismos, es bastante superior a cualquiera de las técnicas citadas antes[2]. Por ello, si

[2]Al menos, en la implementación utilizada en el presente proyecto no se ha encontrado

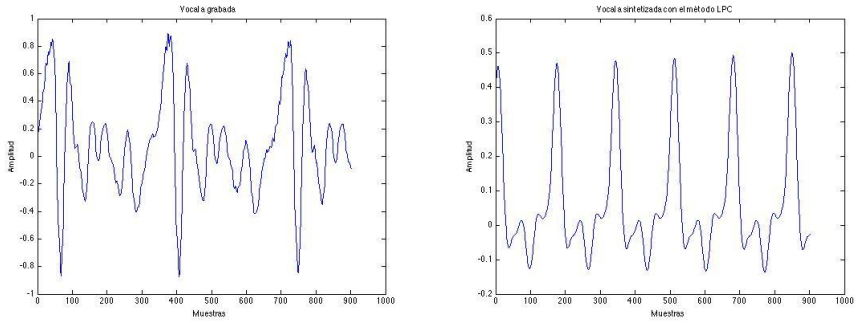


Figura 5.10: Vocal *a*: muestreada (izda.) y sintetizada (dcha.).

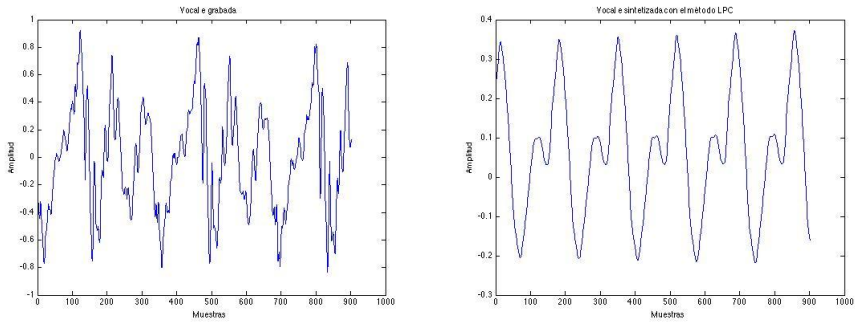


Figura 5.11: Vocal *e*: muestreada (izda.) y sintetizada (dcha.).

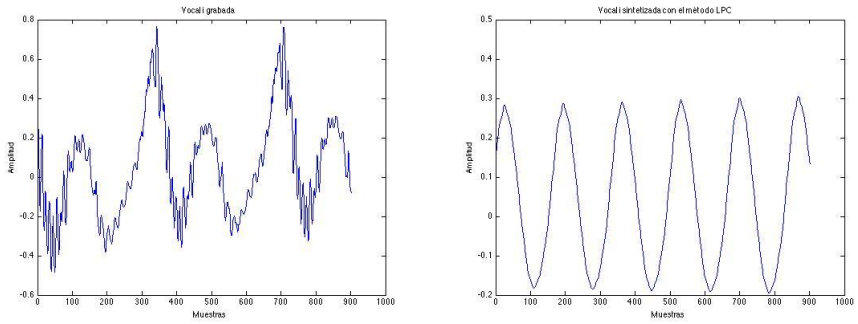


Figura 5.12: Vocal *i*: muestreada (izda.) y sintetizada (dcha.).

el resultado obtenido con esta técnica y con el método de los formantes es

forma de editar los archivos .lp ni para el examen detallado de los coeficientes ni para su edición manual.

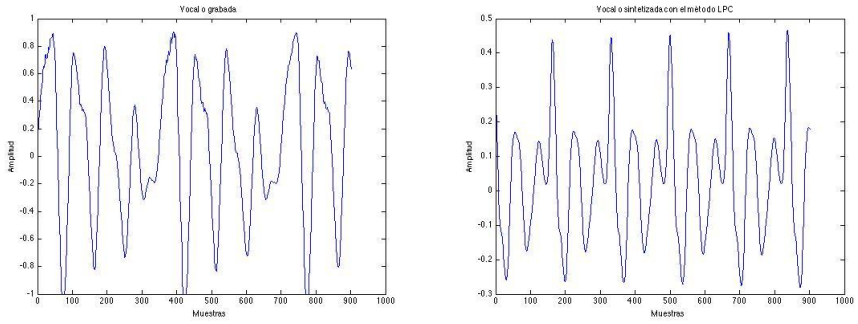


Figura 5.13: Vocal *o*: muestreada (izda.) y sintetizada (dcha.).

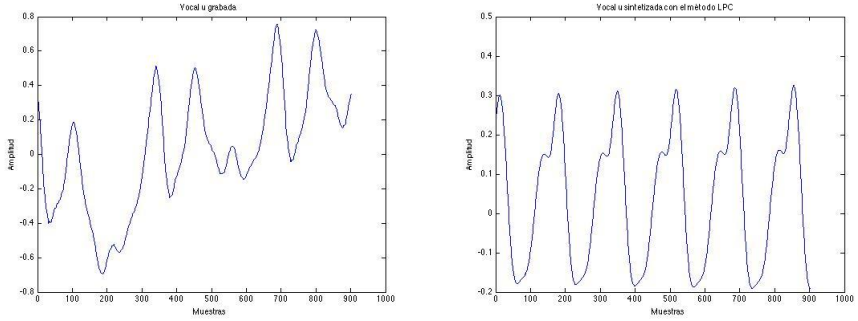


Figura 5.14: Vocal *u*: muestreada (izda.) y sintetizada (dcha.).

bastante similar (seguramente el filtrado por LPC obtenga un mejor resultado pero no demasiado audible, tanto en consonantes como en vocales), se optará por el segundo de ellos a la hora de realizar síntesis por realizar un sintetizador que sea computacionalmente eficiente.

## 5.3. Envolventes de amplitud

La utilización de envolventes de amplitud tiene, en el presente proyecto, tiene una doble finalidad:

- Conseguir la síntesis de consonantes de tipo impulsivo como son los fonemas oclusivos.
- Eliminación de ruidos en las señales creadas con síntesis concatenativa.

A pesar de esta doble finalidad, en este apartado sólo se tratará lo relativo a la síntesis de fonemas oclusivos por ser el segundo objetivo perteneciente a

otro apartado.

Retomando la idea de la síntesis de fonemas oclusivos, la idea es generar una seal (en principio, ruido aleatorio) y conferirle una envolvente, como la que podemos ver en la figura 5.15, que simule una señal prácticamente de pulso.

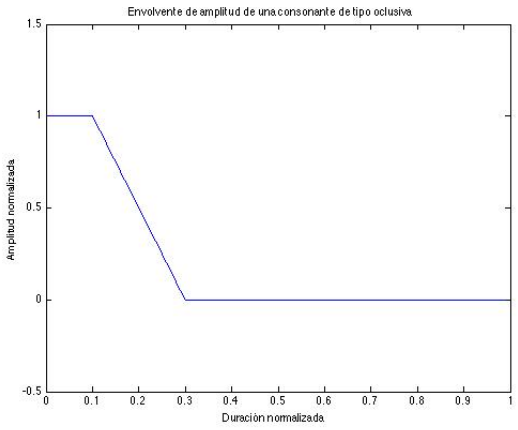


Figura 5.15: Envolvente de amplitud a aplicar a las señales que se creen con la técnica de envolventes de amplitud.

Los sonidos que sinteticemos en este apartado tendrán siempre una envolvente igual o muy parecida (puede que no exactamente igual por todo el hecho del filtrado y demás procesado de la señal), con lo cual no ser necesario especificar cuál será la forma de onda obtenida para cada caso.

A continuación mostramos los resultados obtenidos de realizar diversas pruebas modificando, sobre todo, la señal de origen.

- Señal de origen de tipo ruidoso
  En este caso hemos tomado, como señal a procesar, una señal de tipo ruido para intentar modelar consonantes, partiendo de la hipótesis de que su espectro es inarmónico y que, por tanto, carecen de altura.

  Los diferentes tipos de ruido utilizados han sido los generalmente utilizados para el modelado del habla, es decir, ruido blanco y ruido blanco gaussiano.

- Ruido blanco
  Para llevar a cabo esta serie de pruebas se ha utilizado el *opcode* **randi** y cabe destacar que, aunque se ha utilizado siempre la envolvente de amplitud antes comentada, también hemos hecho una implementación con y sin formantes para ver los diferentes resultados.

  1. Implementación sin formantes
     La señal obtenida de esta forma es la que encontramos en la figura 5.16.

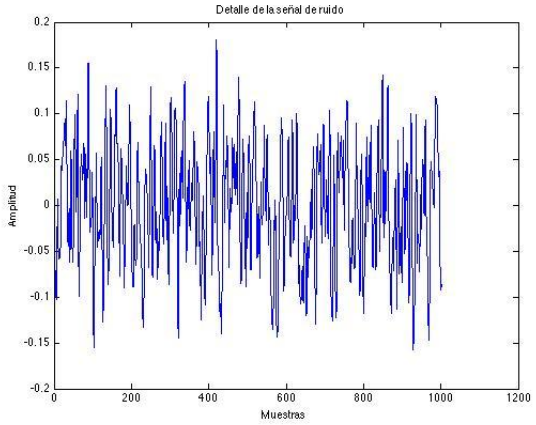


Figura 5.16: Detalle de la síntesis de la consonante *B*.

     En este caso, el resultado es el mismo que en el resto de casos en los que se ha utilizado señal aleatoria para sintetizar algo: un sonido demasiado ruidoso que no permite distinguir nada y que además, al unirlo a una vocal, no da un buen resultado.

  2. Implementación con formantes
     El resultado de esta forma de síntesis lo encontramos en la figura 5.17.

     En este caso, ya sea con el filtrado por formantes o con el filtrado LPC, el resultado sigue siendo demasiado ruidoso como para considerarlo una opción a incluir en el sintetizador.

     Lo único destacable de esta técnica es un cierto cambio en el timbre en el resultado de ambas técnicas. Sin embargo, esto

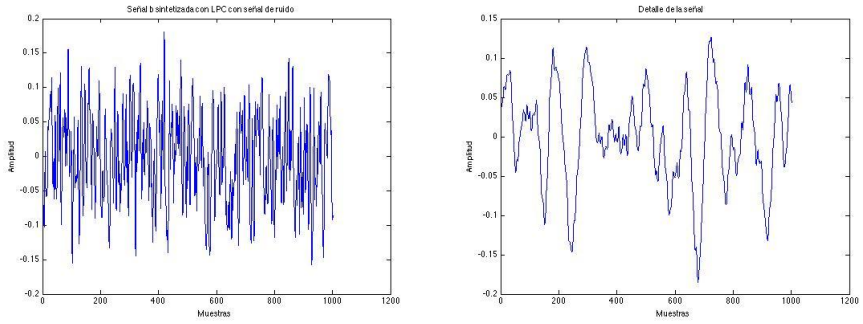


Figura 5.17: Detalle de la síntesis de la consonante *B* por medio de la LPC (izda.) y por medio del filtro por formantes (dcha.).

tampoco es destacable ya que es bastante obvio debido al filtrado implementado.

- Ruido blanco gaussiano
  En este caso se ha hecho exactamente lo mismo que en el caso anterior pero utilizando un ruido con distribución diferente, concretamente gaussiana (*opcode* **gauss**).

  Los resultados obtenidos mediante esta tcnica se comentan a continuación:

  1. Implementación sin formantes
     El resultado de esta técnica se puede ver en la figura 5.18.

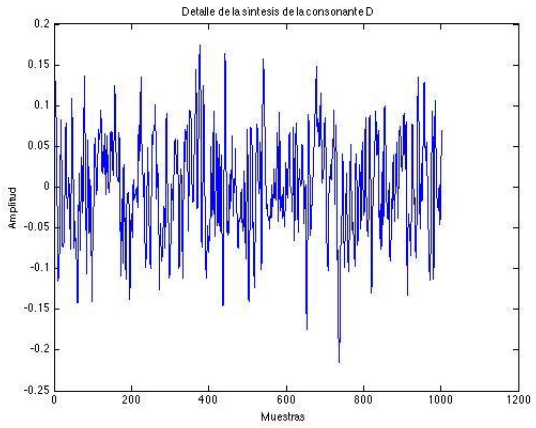


Figura 5.18: Detalle de la síntesis de la consonante *D*.

2. Implementación con formantes
   El resultado obtenido se puede ver en la figura 5.19.

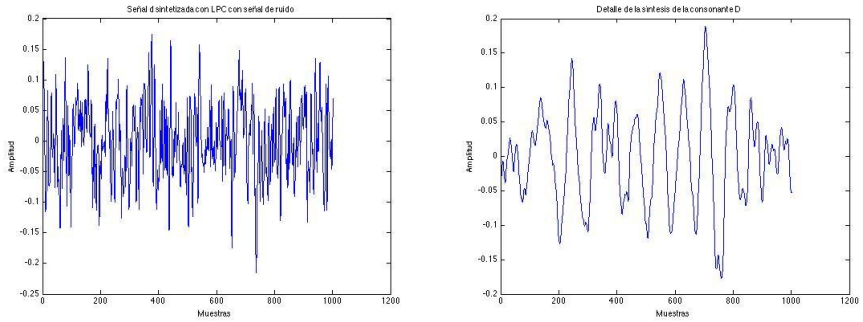


Figura 5.19: Detalle de la síntesis de la consonante *D* por medio de la LPC (izda.) y por medio del filtro por formantes (dcha.).

Como comentario general de esta prueba, cabe destacar que ciertamente hay un cambio de timbre respecto al sonido obtenido en el caso del ruido blanco. Sin embargo esto es algo obvio por el hecho de haber utilizado una señal con una distribución espectral distinta.

En lo relativo a la calidad de la señal, como ocurría en el caso anterior, el resultado sigue siendo demasiado ruidoso como para permitir distinguir qué consonante se est sintetizando.

- Señal de origen de tipo periódico
  Con este método se pretende utilizar señales de tipo periódico para la síntesis de fonemas consonánticos. En principio, como la gran parte de las consonantes son sordas, y mucho más las de tipo oclusivo a las cuales estamos intentando dar solución con este apartado en concreto, no es probable que su funcionamiento sea exitoso. Sin embargo, para estar seguros es necesario comprobarlo.

  Como en el caso de la implementacin con ruido, aquí se va a emplear como señal de origen tanto una señal generada mediante el *opcode* **buzz** (señal con espectro cuyo promedio es constante) como una generada con el *opcode* **gbuzz** (señal con espectro, en promedio, decreciente a medida que sube la frecuencia).

- Señal **buzz**

  1. Implementación sin formantes
     El resultado de esta operación se muestra en la figura 5.20.

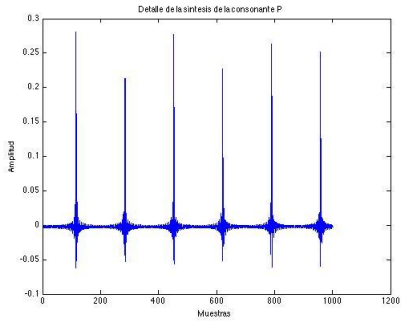


Figura 5.20: Detalle de la síntesis de la consonante *P*.

  2. Implementación con formantes
     El resultado de esta técnica se muestra en la figura 5.21.

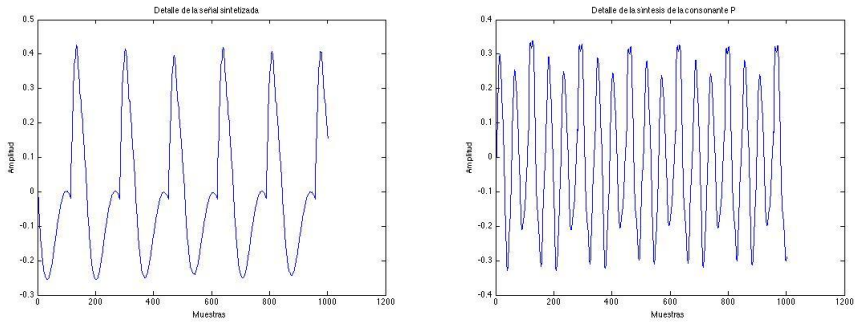


Figura 5.21: Detalle de la síntesis de la consonante *P* por medio de la LPC (izda.) y por medio del filtro por formantes (dcha.).

Como era de esperar por el *opcode* **buzz**, la señal resultante es periódica. Aunque aquí no se pueda ver, las diferentes señales ciertamente tienen la envolvente descrita al principio, lo que hace que, en cierta forma, parezcan señales de tipo impulsivo, que es justo lo que se pretendía conseguir. Sin embargo, ninguna da un resultado apropiado (no son inteligibles) como para su utilización posterior.

- Señal **gbuzz**

  1. Implementación sin formantes
     El resultado obtenido se muestra en la figura 5.22.

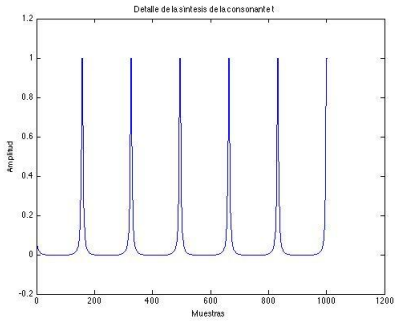


Figura 5.22: Detalle de la síntesis de la consonante $T$.

  2. Implementación con formantes
     El resultado de este proceso lo encontramos en la figura 5.23.

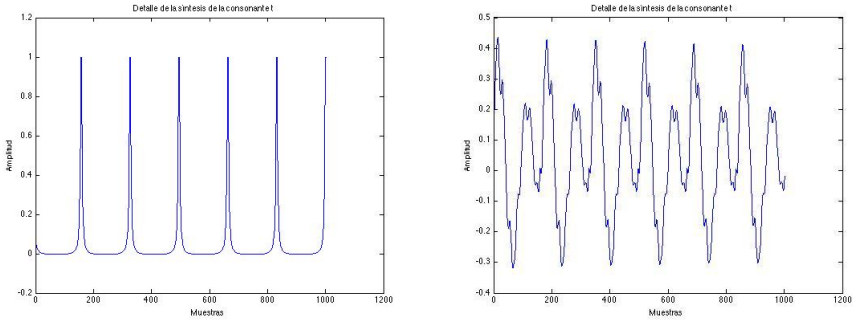


Figura 5.23: Detalle de la síntesis de la consonante $T$ por medio de la LPC (izda.) y por medio del filtro por formantes (dcha.).

Como en el caso anterior, el haber utilizado una señal periódica se ve corroborado por los resultados obtenidos en las formas de onda de las señales que podemos observar. También podemos apreciar un cambio de timbre (obviamente, más allá del diferente filtrado que se ha realizado) por el hecho de haber utilizado una señal como base para la síntesis.

Sin embargo, y como ocurría en el caso anterior, el resultado, a pesar de imitar bastante bien el impulso, la consonante sigue sin ser inteligible.

- Conclusiones
  Como conclusión final sobre este método podemos señalar que, a pesar de que las envolventes de las consonantes de tipo oclusivo se pueden imitar bastante bien, el resultado audible no es satisfactorio, ya sea por la gran cantidad de ruido que generan los *opcodes* de señales aleatorias o por la poca inteligibilidad de las consonantes sintetizadas con señales de tipo periódico.

## 5.4. Modelos sinusoidales

Con la implementación de este método se pretende utilizar la principal ventaja que ofrece el modelo de síntesis aditiva: su tendencia a la periodicidad de la onda resultante, lo que influye de forma positiva en la síntesis de la voz cantada.

Para la implementación de este método se han utilizado prácticamente los mismos datos que en el caso del método de los formantes (necesitamos tantos los formantes de la señal como su amplitud relativa) sólo que, en lugar de utilizar una señal de banda ancha para filtrarla, hemos utilizado ondas sinusoidales (en teoría, deltas de Dirac en el dominio de la frecuencia) con la amplitud y frecuencia del formante, para luego sumarlas todas y así obtener el espectro deseado, quedando así definida la señal.

- Vocales
  Como primer experimento en este tipo de síntesis, en la figura 5.24 se muestra el resultado de sintetizar con este método las cinco vocales del castellano. Cabe destacar que, al tratarse de una prueba, se han utilizado las amplitudes y frecuencias correspondientes a la voz Soprano tal cual se especifica en el Apéndice B.

  El resultado obtenido con esta forma de síntesis es un sonido que no es del todo inteligible (cuesta distinguir, en algunos casos, qué vocal es) pero tiene ventajas como el hecho de dotar de una periodicidad a la señal sintetizada[3] y que, además, es muy eficiente desde el punto

[3]Esto proporciona un mejor control de la entonación a pesar de la baja inteligibilidad del sistema, algo que en síntesis de voz cantada puede ser beneficioso ([12]).

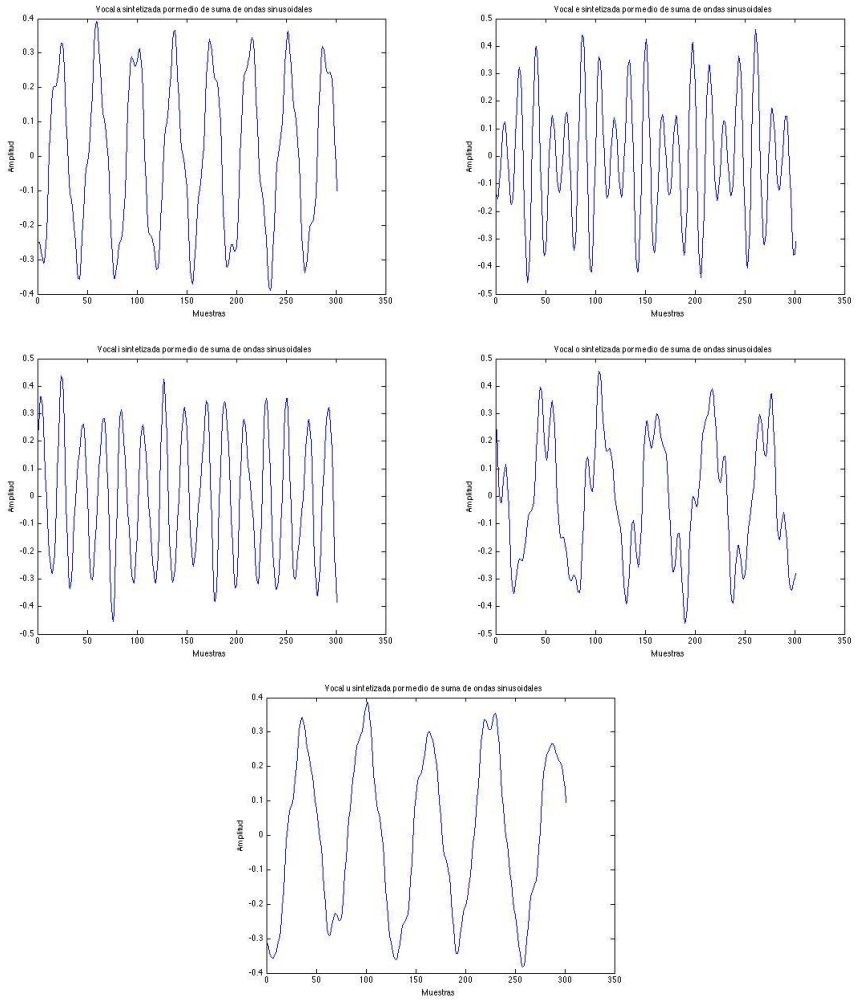


Figura 5.24: Síntesis de las cinco vocales: *a* (superior izda.), *e* (superior dcha.), *i* (centro izda.), *o* (centro dcha.), *u* (inferior).

de vista computacional ya que requiere muy poca cantidad de operaciones y nada de memoria física, siempre que tengamos pocos parciales. Volviendo al caso realizado sobre las vocales, en cada una de ellas se han utilizado cinco osciladores sinusoidales modelando, cada uno de ellos, uno de los formantes de la señal (se recuerda que los datos numéricos acerca de los formantes se encuentran en el apartado Soprano del Apéndice B), más además una envolvente de amplitud de tipo amplitud-sostenimiento-relajación (ASR) para evitar ruidos.

- Consonantes
  Para comenzar con este apartado vamos a intentar sintetizar, en base a los formantes que hemos utilizado en el método de la síntesis sustractiva por formantes, las consonantes que se pudieron modelar con una señal de tipo periódico como señal de excitación. El resultado obtenido se puede ver en la figura 5.25. Como ocurría con las vocales, el resul-

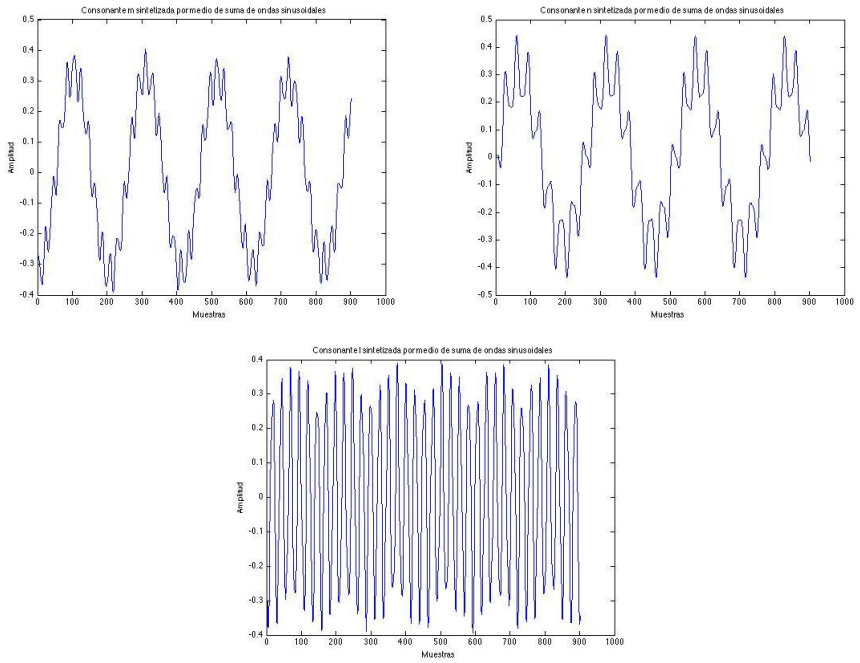


Figura 5.25: Síntesis de las consonantes $m$ (superior izda.), $n$ (superior dcha.), $l$ (inferior).

tado no es inteligible. Sin embargo, su concatenación con las vocales, sobretodo las creadas también con este método, es bastante suave por ser ambas señales de tipo periódico, lo cual elimina una de nuestros problemas, es decir, la unión entre dos sonidos sintetizados.

En lo relativo a síntesis de consonantes de tipo sordo no se puede ser tan optimista: como ya hemos comentado antes, los formantes en este tipo de señales no están tan claros y, por tanto, el crear una suma de señales que modelen ese espectro no es tan fácil como antes. Además, el hecho de forzar a una señal de tipo sordo a sonar como sonora no beneficia demasiado a nuestro sistema porque no se consigue un gran

resultado.

En las figuras 5.26 y 5.27 podemos ver el resultado de la síntesis de las consonantes *s* y *r* mediante este método.

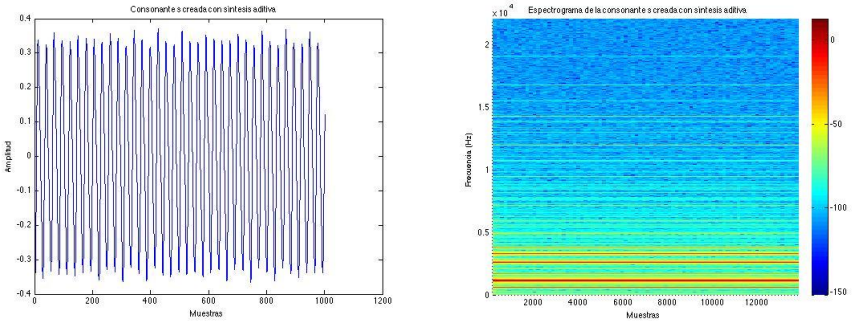


Figura 5.26: Consonante *s* con síntesis aditiva (izda.) y espectrograma de la misma señal (dcha.).

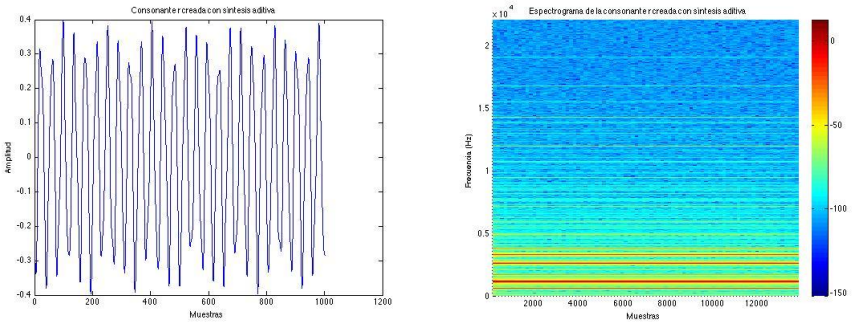


Figura 5.27: Consonante *r* con síntesis aditiva (izda.) y espectrograma de la misma señal (dcha.).

- Conclusiones
  En lo relativo a este tipo de síntesis, el resultado en señales de tipo periódico es aceptable teniendo en cuenta que, como mucho, se está realizando la suma de cinco ondas sinusoidales. Sin embargo, al conseguir un resultado mejor con la síntesis de tipo sustractiva (tanto para vocales como para consonantes de tipo periódico), a pesar de que sea un poco más lenta (aunque en realidad, con la potencia actual de los ordenadores, el coste parece el mismo en relación al tiempo de ejecución), se prefiere el uso del primer método explicado para intentar conseguir una mayor inteligibilidad.

  En lo relativo a consonantes de tipo sordo, al no ser el resultado de este método nada satisfactorio, no se incluirá finalmente en el sintetizador nada relacionado con este tipo de síntesis.

## 5.5. Tablas de onda

La idea básica de este método es, en lugar de utilizar los *opcodes* para generar las señales que necesitamos (ruido blanco, por ejemplo), utilizar funciones de generación de tablas de onda para intentar conseguir algún cambio en la señal resultante.

Aunque la idea básica es la anteriormente citada, la finalidad de este método es la de intentar crear señal de tipo ruido, distinta a la generada por medio de los *opcodes*, para poder realizar la síntesis de consonantes sordas que no se ha podido realizar con el método de los formantes por las razones allí especificadas.

- Consonante *s*
  El resultado obtenido se muestra en la figura 5.28.

  En este caso, la señal creada por medio de la función **GEN** 21 con distribución gaussiana tiene un resultado parecido a la señal creada con el *opcode* **gauss**, es decir, un gran nivel de ruido que contrasta con la periodicidad de las vocales haciendo, por tanto, inviable el unirlas de forma coherente. Sí que cabe destacar que la tabla aquí probada da un ligero matiz distinto al *opcode*: el sonido tiene un timbre más metálico, lo que hace que el sonido parezca mucho más sintético y menos realista.

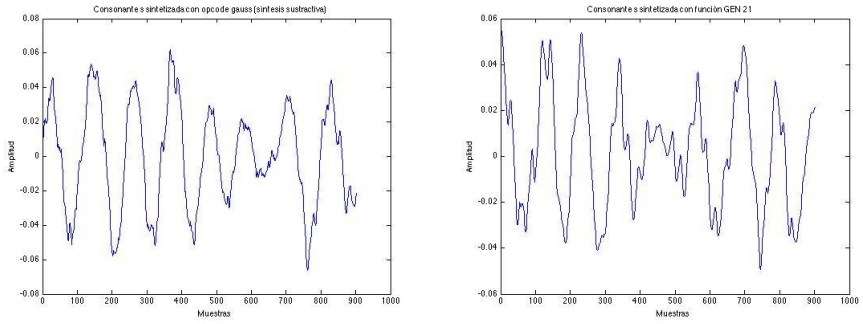


Figura 5.28: Síntesis de la consonante *s*: *opcode* **gauss** (izda.), función **GEN** 21 (dcha.).

- Consonante *z*
  El resultado obtenido se muestra en la figura 5.29.

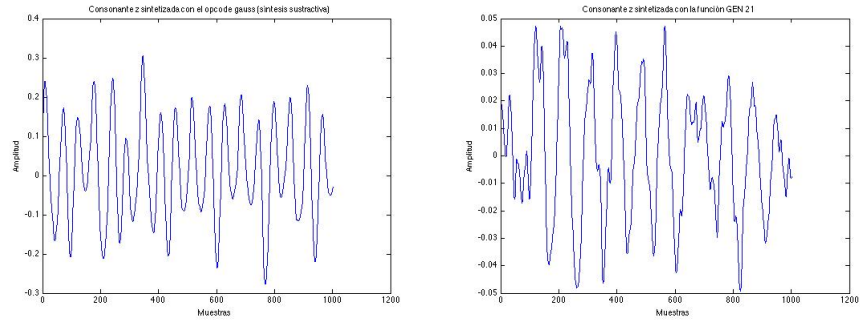


Figura 5.29: Síntesis de la consonante *z*: *opcode* **gauss** (izda.), función **GEN** 21 (dcha.).

  En este caso ocurre exactamente lo mismo que en la síntesis de la consonante *s*: el resultado obtenido con la tabla de ondas es parecido, en términos de ruido, al resultado obtenido con el *opcode* **gauss**. Sin embargo, y como también ocurría en el caso anterior, con la función **GEN** 21 obtenemos un timbre mucho más metálico que en el caso del *opcode*.

- Conclusiones
  El resultado para este tipo de síntesis no ofrece una solución viable

para la síntesis de ningún tipo de sonidos sordos, por lo que se desecha su inclusión en el sintetizador final.

## 5.6. Método del Vocoder (Phase–Vocoder)

Al igual que en el caso de la LPC, CSound incorpora ya las rutinas de análisis-resíntesis de un vocoder de fase. El vocoder de fase es un algoritmo que, además de actuar como un vocoder normal (vocoder por canales) y realizar el proceso análisis-resíntesis del sonido, también permite la cambiar su altura sin variar su duración (proceso llamado *pitch-shifting*) o cambiar su duración sin variar la altura (*time-stretching*). Para lograr todo esto utiliza la llamada transformada de Fourier a corto plazo (STFT, *Short-Time Fourier Transform*), que es un caso especial de transformada de Fourier que, en lugar de aplicarse de una vez a toda la señal, se aplica con una ventana sobre tramos de la señal, además de con un solapamiento entre tramos, normalmente.

Los *opcodes* que nos permiten realizar esta función son **pvanal**, **pvadd** y **pvoc** cuya sintaxis se incluye en el Apéndice A. Cabe destacar que los dos últimos son *opcodes* que permiten realizar la síntesis de la señal, con lo cual utilizaremos ambos con la finalidad de ver cuál nos devuelve un mejor resultado. La diferencia básica entre estos dos *opcodes*, de forma sencilla, es que mientras que **pvoc** tiene ya una serie de osciladores definidos de serie para realizar la síntesis, con **pvadd** podemos elegir qué señal utilizar como base en el oscilador, la cual, en este caso, ha sido una señal sinusoidal.

- Síntesis de vocales
  Para comenzar con un caso relativamente sencillo, vamos a intentar realizar la síntesis de algunas vocales. Los resultados se muestran en las figuras 5.30 y 5.31.

  En este caso podemos ver que la señal generada con el *opcode* **pvadd** se asemeja menos a ruido que la generada con **pvoc**, a pesar de que ambas son funciones claramente periódicas. Sin embargo, y seguramente por efecto de esa gran cantidad de componentes de alta frecuencia, las señal de la derecha resulta, al oído, mucho más sintética y poco inteligible que su correspondiente situada a la izquierda.

- Síntesis de consonantes
  A continuación procederemos a realizar la síntesis de consonantes con este método. Mientras que en el caso anterior el *opcode* **pvadd** sólo

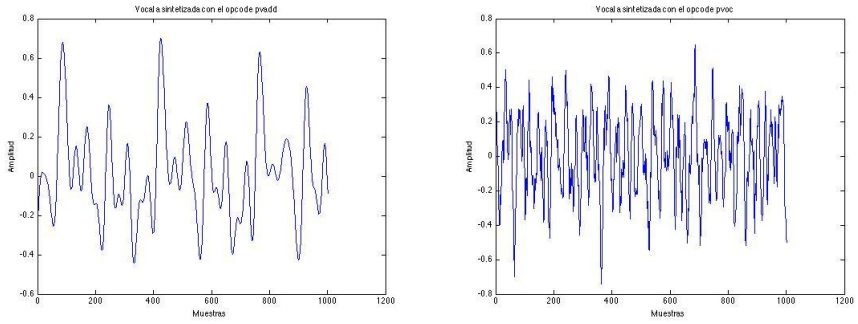


Figura 5.30: Vocal *i* sintetizada por medio de **pvadd** (izda.) y por medio de **pvoc** (dcha.).

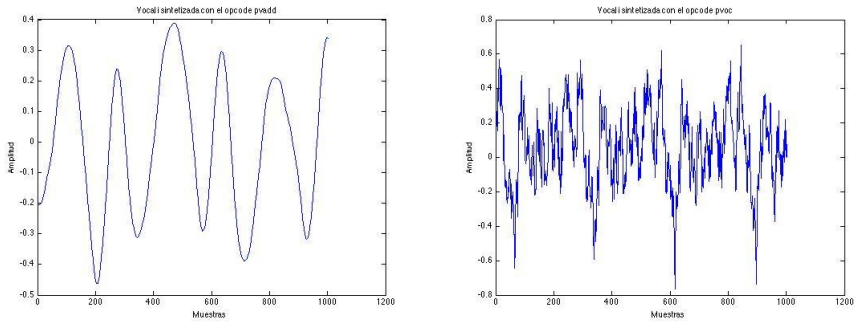


Figura 5.31: Vocal *a* sintetizada por medio de **pvadd** (izda.) y por medio de **pvoc** (dcha.).

recibía como señal base una sinusoide, aqu hemos creado, además de ésta, una tabla de onda con ruido distribuido de forma gaussiana.

- Consonante *s*
  El resultado obtenido es el que muestra la figura 5.32.

  Como podemos observar, tanto cuando utilizamos el *opcode* **pvoc** como el **pvadd** con una sinusoide como señal para realizar la síntesis, las señales tienen un comportamiento, en cierta forma periódico, mientras que con la tabla de onda de distribución gaussiana obtenemos una señal de ruido, la cual debería ser la que mejor resultado proporcionara para la síntesis de las consonantes. Sin embargo, y aún tratándose de la síntesis de la *s*, el resultado no es correcto: como en otras ocasiones, el nivel de ruido a la hora de

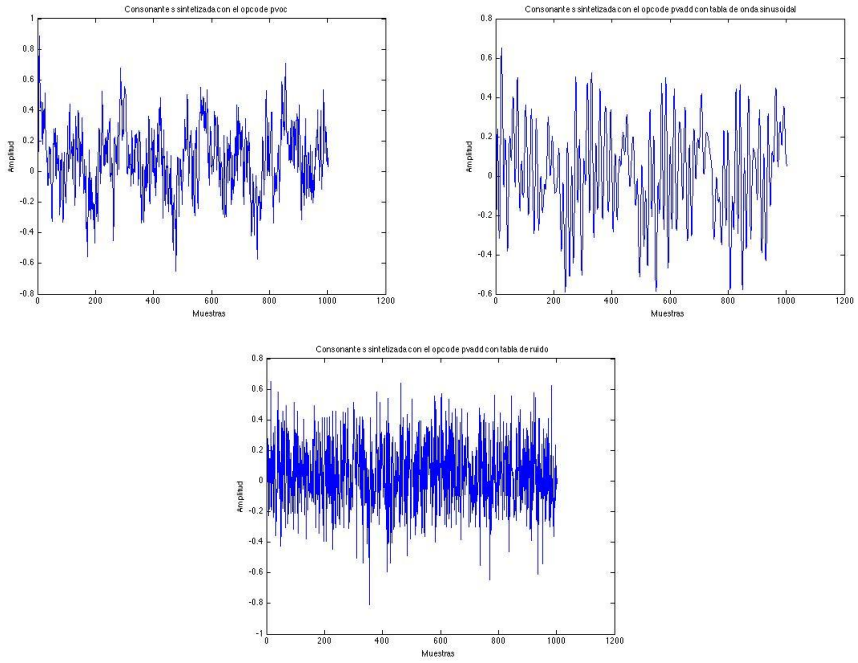


Figura 5.32: Síntesis de la consonante *s*: **pvoc** (superior izda.), **pvadd** con sinusoide (superior dcha.) y **pvadd** con ruido (inferior).

la síntesis es demasiado grande, incluso con un filtrado paso-bajo. En lo relativo a los otros dos sonidos sintetizados, ninguno de los dos consigue ninguna inteligibilidad, además de sonar demasiado artificiales, como metálicos.

- Consonante *l*
  El resultado de esta síntesis se encuentra en la figura 5.33.

  En este caso, la síntesis de la consonante *l* tampoco ha llevado a un buen resultado. Ocurre exactamente lo mismo que en el caso anterior: las dos señales que resultan periódicas no son inteligibles y la generada con ruido, además de tener un nivel muy alto, tampoco es, para nada, identificable.

- Consonante *k*
  El resultado obtenido se encuentra en la figura 5.34.

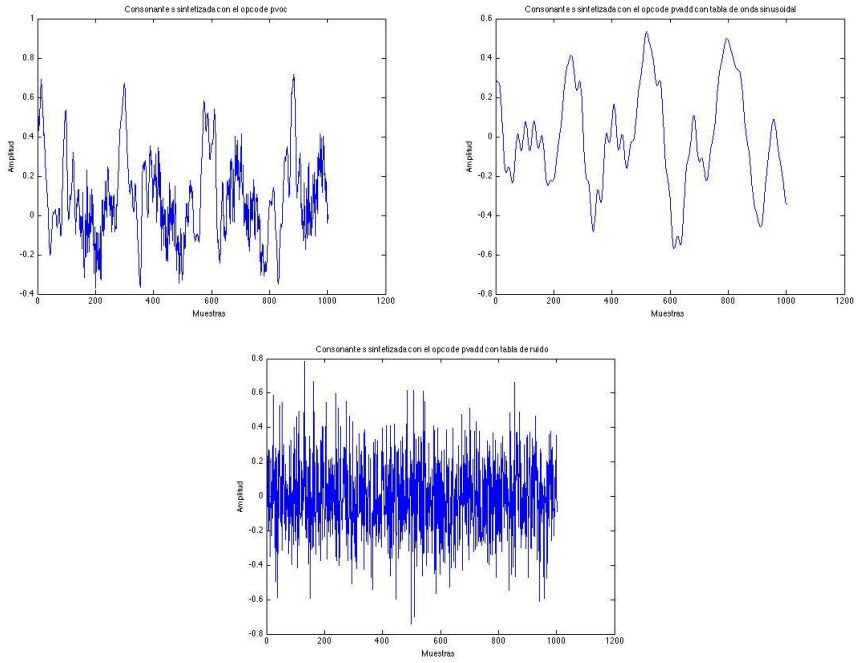


Figura 5.33: Síntesis de la consonante *l*: **pvoc** (superior izda.), **pvadd** con sinusoide (superior dcha.) y **pvadd** con ruido (inferior).

Finalmente, en la síntesis de la consonante *k*, aún haciendo uso de una envolvente de tipo consonante oclusiva, no se ha conseguido un resultado óptimo con ninguna de las tres variantes por las mismas razones que en los dos casos anteriores.

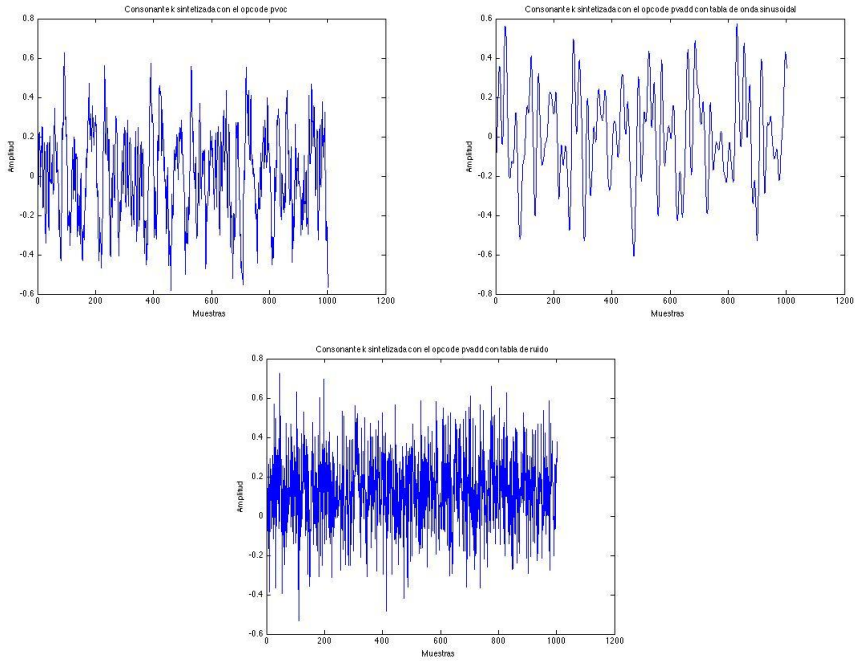


Figura 5.34: Síntesis de la consonante *k*: **pvoc** (superior izda.), **pvadd** con sinusoide (superior dcha.) y **pvadd** con ruido (inferior).

- Conclusiones
  Como conclusión sobre este método, podemos destacar que al no conseguir un resultado óptimo en la síntesis ni de consonantes ni de vocales, desecharemos la implementación de este método en el sintetizador.

## 5.7. Síntesis FOF (Formas de ondas formantes)

CSound incluye diversas rutinas ya programadas sobre la síntesis de sonidos mediante la técnica de síntesis de formas de onda formantes, las cuales utilizaremos, a continuación, para dar solución al problema de la síntesis consonántica.

- Síntesis de vocales
  Antes de comenzar con la síntesis de las consonantes, y con la finalidad de comprobar si se obtiene el mismo resultado que en la síntesis sustractiva por formantes, vamos a comprobar las formas de onda de

algunas vocales creadas con ambos métodos que se muestran en las figuras 5.35 y 5.36.

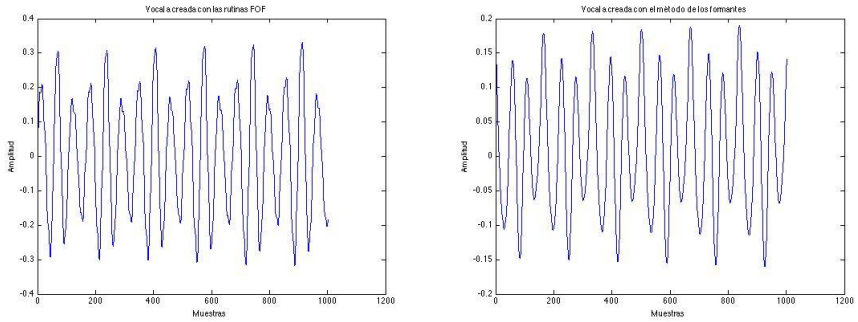


Figura 5.35: Síntesis de la vocal *a* mediante rutinas FOF (izda.) y mediante método de los formantes (dcha.).

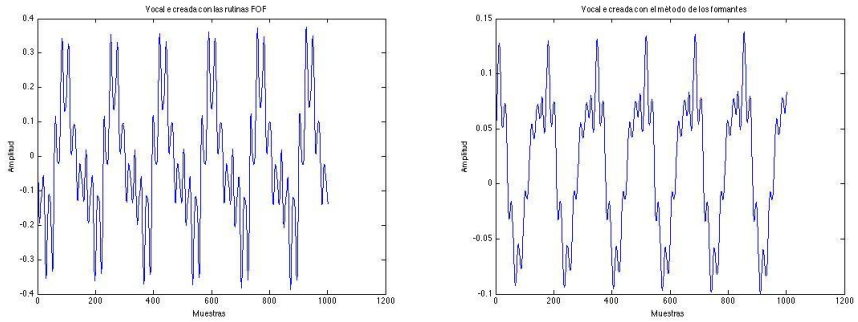


Figura 5.36: Síntesis de la vocal *e* mediante rutinas FOF (izda.) y mediante método de los formantes (dcha.).

- Síntesis de consonantes
  En lo relativo a la síntesis de consonantes mediante FOF, como en el caso de la síntesis sustractiva por formantes, comenzaremos por aquéllas que sí que han dado un resultado positivo anteriormente (las consonantes que se han podido modelar como segmentos sonoros) para después pasar a las que, por ahora, no se ha conseguido dar solución (consonantes sordas).

  1. Consonantes sonoras
     Como resultados de esta forma de síntesis se pueden observar en

las figuras 5.37, 5.38 y 5.39.

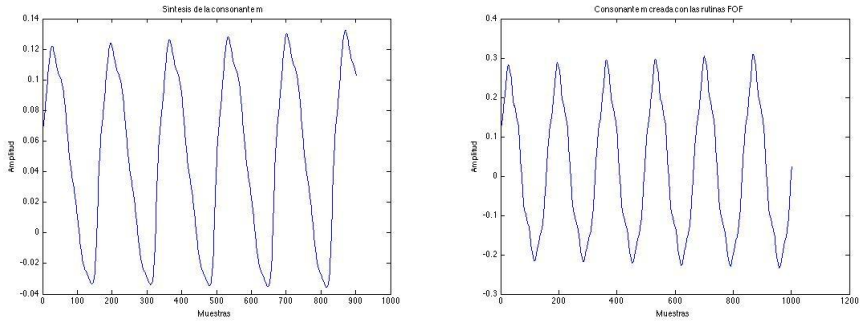


Figura 5.37: Síntesis de la consonante *m* mediante rutinas FOF (izda.) y mediante método de los formantes (dcha.).

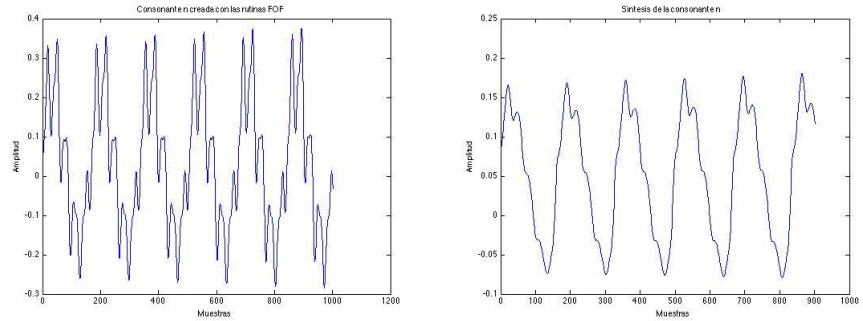


Figura 5.38: Síntesis de la consonante *n* mediante rutinas FOF (izda.) y mediante método de los formantes (dcha.).

Como podemos observar, en este caso sí que se puede notar más diferencia entre las consonantes sintetizadas por el método de los formantes y por las formas de ondas formantes. Como se han utilizado los mismo parámetros respecto a los formantes en ambas formas de síntesis (ancho de banda, frecuencia y amplitud de cada uno), la razón de la diferencia entre ambas señales es los diferentes parámetros elegidos en el *opcode* **fof** que éste necesita. Cabe destacar que estos parámetros han sido escogidos en base al criterio de imitar de la forma más clara posible el habla humana[4].

[4]Para síntesis vocal, los valores *kris*, *kdur* y *kdec* han de valer 0.003, 0.02 y 0.007, respectivamente. Se recomienda la lectura del Apéndice A.

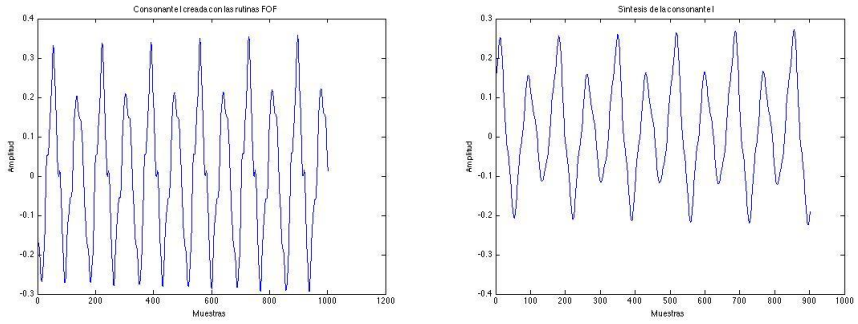


Figura 5.39: Síntesis de la consonante *l* mediante rutinas FOF (izda.) y mediante método de los formantes (dcha.).

En cuanto a la inteligibilidad, ambas formas suenan bastante correctas, incluso pudiendo llegar a ser esta forma de síntesis mejor que cualquiera de las anteriores. Esto se corroborar a la hora de realizar pruebas una vez se implemente el sintetizador definitivo.

2. Consonantes sordas
   Finalmente pasamos al apartado que más problemas está dando en la resolución del presente proyecto: la síntesis de consonantes de tipo sordo.

   Este apartado será resuelto con los mismos valores que los utilizados para la síntesis de consonantes de tipo sordo en el método de la síntesis sustractiva por formantes, con la finalidad de ver alguna mejora.

   Para intentar llegar a una conclusión más amplia, utilizaremos tanto sinusoides amortiguadas como ruido blanco gaussiano para conformar los gránulos de esta síntesis y comprobar los resultados.

   - Sinusoides amortiguadas
     Los resultados obtenidos se muestran en las figuras 5.40 y 5.41.

     Una vez realizadas estas pruebas, y salvando el hecho de si se parecen o no las formas de onda, el resultado obtenido no es del todo satisfactorio. Aunque al modelar las consonantes

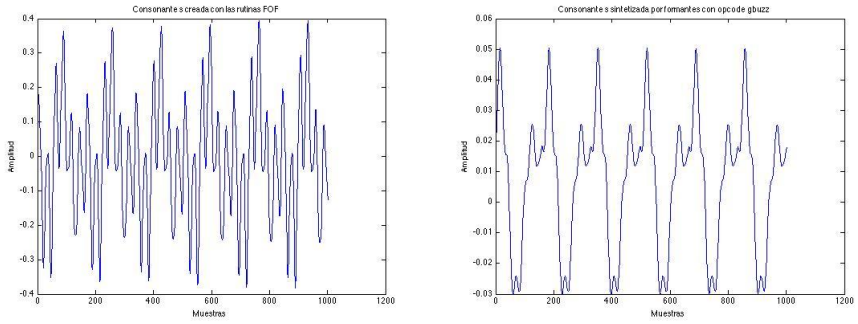


Figura 5.40: Síntesis de la consonante *s* mediante rutinas FOF (izda.) y mediante formantes (dcha.).

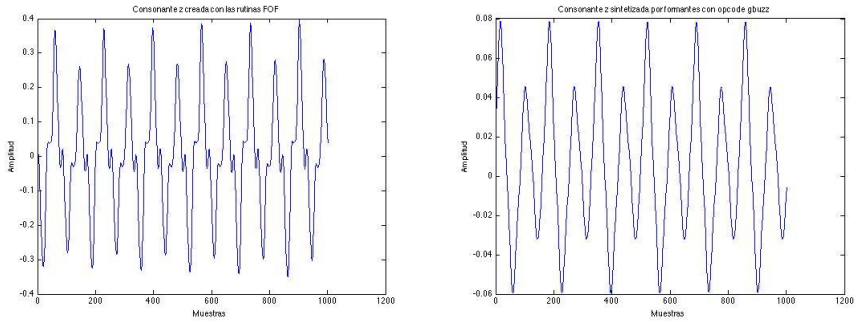


Figura 5.41: Síntesis de la consonante *z* mediante rutinas FOF (izda.) y mediante formantes (dcha.).

con señales periódicas sí que hay una transición suave entre la misma y una vocal, el sonido no resulta inteligible, de forma parecida a lo que ocurría en los modelos sinusoidales.

- Ruido blanco gaussiano
  Los resultados obtenidos se muestran en las figuras 5.42 y 5.43.

  Como resultado de estos experimentos podemos afirmar que, en cierta forma, el resultado ha sido mucho mejor que con la utilización de los *opcodes* típicos de ruido, ya que se obtenía una señal más clara. Sin embargo, y a pesar incluso de realizar filtrados de señal para eliminar otras componentes frecuenciales indeseadas, el resultado era inteligible.

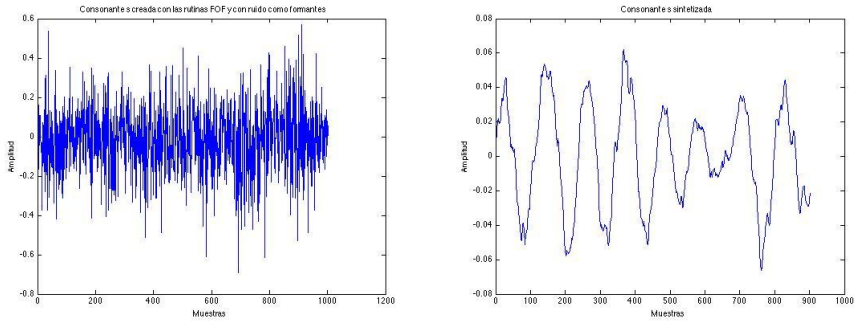


Figura 5.42: Síntesis de la consonante *s* mediante rutinas FOF (izda.) y mediante método de los formantes (dcha.).

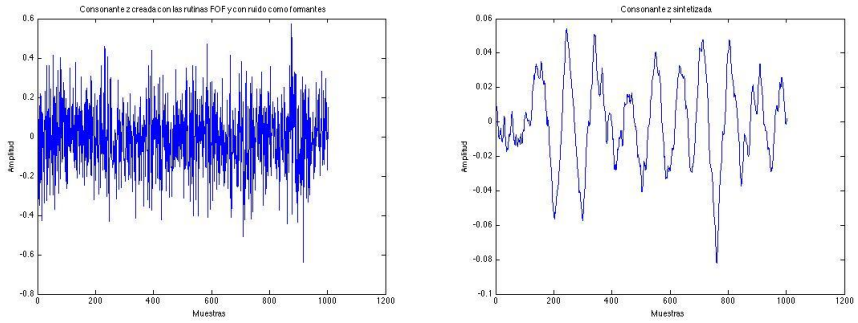


Figura 5.43: Síntesis de la consonante *z* mediante rutinas FOF (izda.) y mediante método de los formantes (dcha.).

- Algoritmo de *The CSound Book*
  En en libro de Richard Boulanger ([8]), concretamente en el apartado *Procesado de muestras con el opcode fof de CSound*[5], se describe la aplicación de este tipo de síntesis para la creación de un coro a partir de una señal externa de audio.

  Básicamente, este algoritmo tiene el siguiente funcionamiento:

  - Primero crea una serie de valores de *jitter* de forma aleatoria que utilizará para modular la frecuencia fundamental de la señal.
  - Tras ello crea otra serie de valores, dinámicos, que modificarán, junto al *jitter* anterior, la frecuencia fundamental la señal a crear.

[5]Capítulo 15, apartado denominado *Ejemplo de coro* (página 300).

- Una vez hecho esto, crea las envolventes tanto de los formantes como de la señal en general.
- En este punto, el algoritmo obtiene otra serie de valores que esta vez serán para modificar la frecuencia fundamental de los formantes de la señal.
- Por último añade una señal para controlar el campo *koct* de **fof** y así permitir la transposición de altura.
- Finalmente crea dos señales mediante el *opcode* **fof**, que son bastante parecidas (las diferencias en cada uno de los valores de los parámetros es mínima), devolviendo cada señal por un canal (estéreo) para dar así mayor realismo al sistema.

Cabe destacar que el sonido que nosotros queremos procesar se pasa incluído en una tabla desde la partitura (realmente, se hacen dos lecturas de la señal: una que lee desde el primer instante del fichero y otra que no comienza por el principio del fichero para, de esta forma, hacer que no todo suene perfecto y dar mayor sensación de realismo).

En la figura 5.44 podemos encontrar un ejemplo de síntesis mediante este método.

Tras la aplicación del algoritmo, las vocales y consonantes sintetizadas, a pesar de poseer un cierto efecto de profundidad e incluso dar un efecto de polifonía, el resultado no es óptimo ya que, además de lo citado antes, las señales sintetizadas poseían un cierto tono metálico, lo cual lo aleja mucho del realismo esperado, y poca inteligibilidad.

- Conclusiones
  Como conclusión acerca de este método de síntesis podemos afirmar que, a pesar de no haber obtenido consonantes de tipo sordo, sí que hemos podido encontrar una técnica que puede equipararse en calidad a la de los formantes (incluso puede que dé más inteligibilidad), por lo que, a la hora de implementar el sistema, se deberán realizar diferentes pruebas para escoger el método más adecuado.

## 5.8. Síntesis concatenativa

Otra de las técnicas que se ha utilizado para llevar a cabo nuestra tarea ha sido utilizar el método conocido como síntesis concatenativa o por

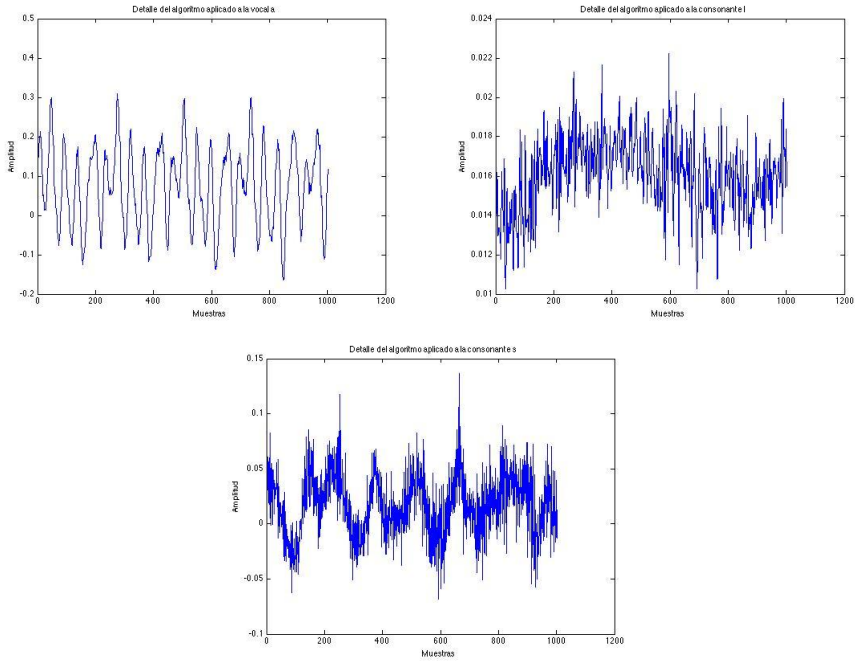


Figura 5.44: Síntesis de la vocal *a* (superior izda.) y las consonantes *l* (superior dcha.) y *s* (inferior).

sonidos muestreados. Para implementar esto ha sido necesario realizar grabaciones de sonidos de consonantes[6], los cuales después han sido procesados para adaptarlos a nuestra necesidad (eliminación de ruidos de fondo, filtrado de algunas señales para eliminar frecuencias indeseadas, reducción de la duración) y, utilizando las rutinas propias de CSound para esta forma de síntesis, se han conseguido unir al resto de la base de datos, tanto de consonantes como de vocales.

La obtención de estas muestras ha sido llevada a cabo por un cantante de coro barítono en una sala de grabación perteneciente al Grupo de Reconocimiento de Formas e Inteligencia Artificial (GRFIA) del Departamento de Lenguajes y Sistemas Informáticos (DLSI). Las características de esta grabación son las que siguen:

- Sala no anecoica: hay reverberación[7].

[6]Cabe destacar que antes de proceder a su grabación se intentó encontrar ejemplos de sonidos consonánticos muestreados en las bases de datos Freesound Project (http://www.freesound.org/) y RWC Music Database, cuyo resultado no fue satisfactorio.

[7]Se estima un $tr_{60}$ de 0.3724 segundos, obtenido con la grabación de una serie de señales

- Micrófono de grabación: Micrófono de condensador Behringer B-1.
- Mesa de mezclas: Mackie Onyx 1220 con módulo FireWire.
- Ordenador: Apple Mac G8.
- Software: Steinberg Cubase SX 3
- La grabación se realizó a una frecuencia de muestreo de 48 kHz y 16 bits de resolución quedándose, después de todo el procesado[8], en una frecuencia de muestreo de 44.1 kHz y 16 bits de resolución.

Las grabaciones llevadas a cabo son todo sonidos consonánticos sordos debido a que no se han podido sintetizar de una forma inteligible y convincente de ninguna otra manera. En las figuras 5.45, 5.46 y 5.47 podemos ver algunos ejemplos de las señales grabadas.

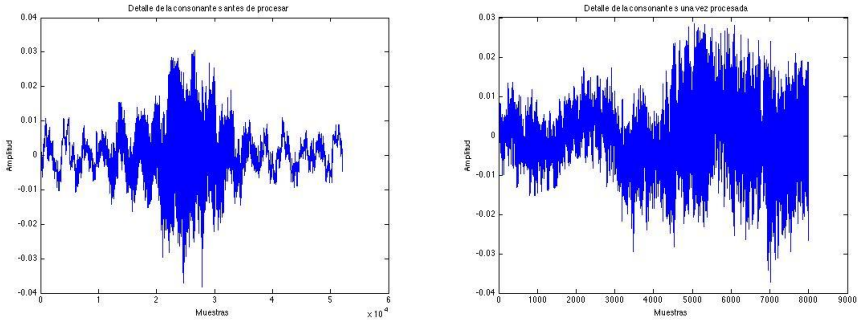


Figura 5.45: Muestra de la consonante *s* grabada (izda.) y tras la eliminación de partes innecesarias (dcha.).

Respecto a lo aquí realizado, cabe destacar que el procesado que se ha realizado tras la grabación de la muestra[9] ha consistido, con la finalidad de obtener una muestra lo más fiel a la realidad pero sin ningún tipo de procesado anterior (por ejemplo, reverberación), en lo siguiente:

- Eliminación de las partes de silencio presentes en la muestra, para así obtener la longitud adecuada. Esto se llevó a cabo tanto en el editor

impulso (en este caso, palmadas de mano) y procesándolas posteriormente para obtener el $tr_{20}$ y, con él, el primer parámetro comentado.

[8]El procesado realizado se especifica más adelante.

[9]En realidad no se grabó una muestra de cada consonante, sino que se realizaron varias tomas de cada una, tanto seguidas como no de vocales.

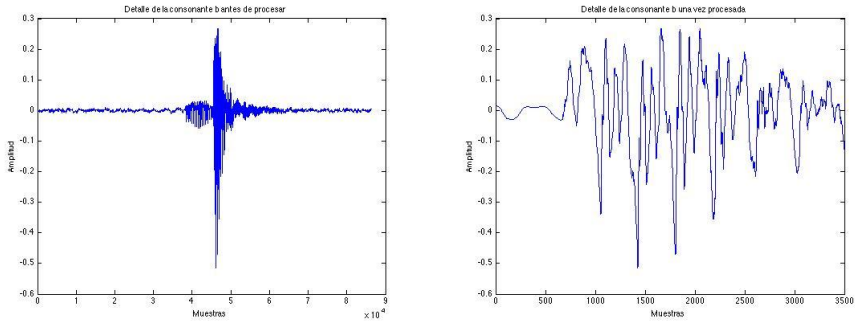


Figura 5.46: Muestra de la consonante *b* grabada (izda.) y tras la eliminación de partes innecesarias (dcha.).

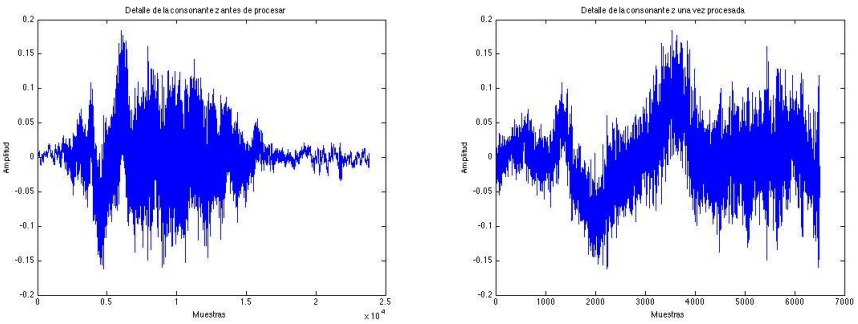


Figura 5.47: Muestra de la consonante *z* grabada (izda.) y tras la eliminación de partes innecesarias (dcha.).

de audio que se estaba utilizando[10] como posteriormente en el entorno Matlab.

- Eliminación de, en el caso de haber, la vocal con la que se grabó la consonante (entorno Matlab).
- Eliminación de la reverberación presente en la muestra (entorno Matlab).
- Establecimiento, tanto del inicio de la muestra como del final, de un valor entorno a cero para así evitar ruidos (entorno Matlab).

También cabe destacar que en la implementación de consonantes con gran nivel de ruido (como la *s* o la *z*), se ha hecho necesario un filtrado paso-bajo

[10]Apple Logic Pro 8.

en CSound para eliminar algunas componentes de alta frecuencia que no permitían la correcta concatenación entre consonante y vocal. Además, en algunos casos se ha hecho necesario el uso de envolventes de amplitud para paliar el efecto de ciertos ruidos (como *clics*) en las muestras, algo que ya se avanzaba en el apartado de *Síntesis por Envolventes de amplitud*.

Por último comentar que las principales dificultades de este método, al menos aquí experimentadas, han sido, primero, el conseguir encontrar la longitud óptima de la muestra de sonido y, por otro lado, y aunque también se ha dado en el resto de métodos, conseguir el solapado idóneo entre consonante y vocal.

Como conclusión de este apartado, el método aquí estudiado ha sido el que mejor resultado ha proporcionado en la síntesis de consonantes de tipo sordo, por lo que será el que se incluya en el sintetizador final.

## 5.9. Opcodes de CSound específicos para voz

Como alternativa a los métodos anteriores se han utilizado rutinas (*opcodes*) ya existentes sobre síntesis de voz en el propio lenguaje CSound, como son **voice** o **fmvoice** con la finalidad de comprobar si existe la síntesis de algún tipo de consonante mediante ellas.

1. **voice**
   Es un *opcode* de emulación de voz humana que utiliza una serie de ficheros externos para crear la señal de salida por medio de transformaciones sobre los mismos. Estos ficheros se pueden descargar de un servidor ftp cuya dirección viene proporcionada en el manual de CSound[11].

   En la figura 5.48 podemos observar algunos ejemplos de sonidos sintetizados con este método.

   En lo referente al resultado obtenido, simplemente viendo la forma de onda se puede deducir que no es un buen método que asegure la inteligibilidad de la señal. Además, una vez escuchadas las señales creadas, se corrobora esta impresión.

2. **vosim**
   Este opcode es la implementación en CSound del algoritmo del mismo

[11] El enlace es ftp://ftp.cs.bath.ac.uk/pub/dream/documentation/sounds/modelling/

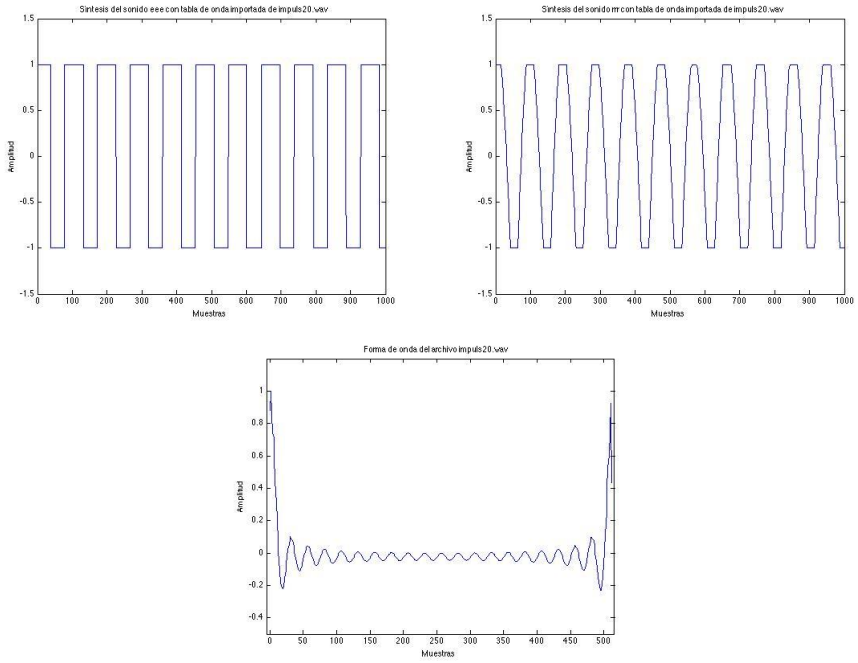


Figura 5.48: Fonema *eee* sintetizado en base a la señal *impuls20.wav* (superior izda.), fonema *rrr* sintetizado en base a la señal *impuls20.wav* (superior dcha.) y forma de onda de *impusl20.wav* (inferior).

nombre, desarrollado en los años 70, que permite crear sonidos de tipo hablado y cantado por medio de la repetición de ráfagas de sinusoides de duración y retardo variable. Estas ráfagas que crea el algoritmo pueden ser interpretadas como una aproximación a los pulsos glotales del habla humana.

En la figura 5.49 podemos ver un ejemplo de síntesis mediante este método.

El resultado obtenido en esta forma de síntesis no es de suficiente calidad como para incluirlo en el sintetizador final.

3. **fmvoice**
   El *opcode* **fmvoice** permite la síntesis de canto vocal mediante la técnica FM. Aunque en un principio se incluyó por el hecho de que se pensaba que era capaz de sintetizar todo tipo de sonidos, al consultar el manual se vio que la implementación era para vocales. Sin embargo, como ru-

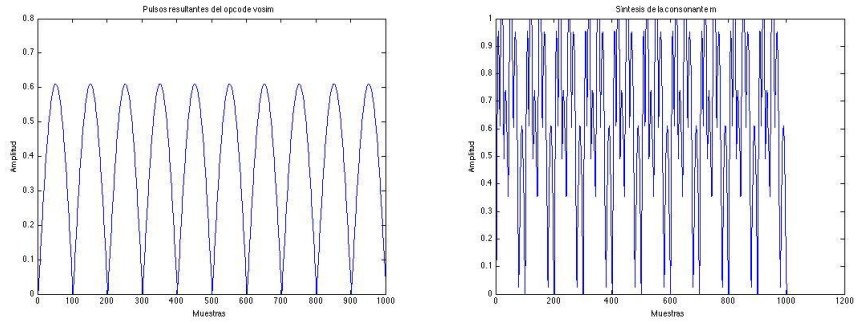


Figura 5.49: Ejemplo de pulsos glotales (izda.) y de consonante *m* sintetizada (dcha.).

tina ya presente en CSound, no viene mal conocerla.

El *opcode* tiene un rango de 65 sonidos vocálicos, eligiendo el que queremos mediante un parámetro de control.

En la figura 5.50 podemos ver un ejemplo de síntesis mediante este método.

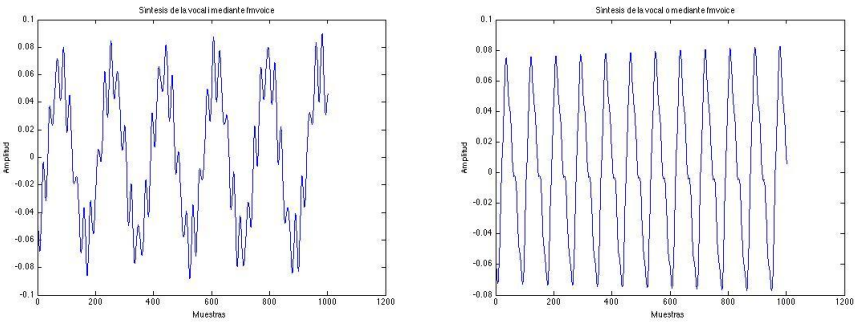


Figura 5.50: Síntesis de la vocal *i* (izda.) y de la vocal *o* (dcha.).

Cabe destacar que, a pesar de haber especificado qué vocales son las que se han sintetizado por el método FM, en realidad no se ha encontrado un sitio en el que consultar qué número corresponde a cada vocal. Por ello, y por la falta de inteligibilidad presente en este método, quizás otro oyente, al escuchar las muestras, pueda juzgarlas de diferente forma y

afirmar que, en realidad, son otras vocales. De hecho, y para ver más aún la ambigüedad que se puede producir, si nos fijamos en la forma de onda obtenida, podemos ver que es bastante parecida a la obtenida en la figura 5.25, que representa la consonante *m* en síntesis aditiva.

4. Conclusiones
   En definitiva, ninguna de las rutinas ya existentes en CSound nos ha ayudado a dar solución al problema de la síntesis de fonemas consonánticos. En lo que sí puede dar una solución aceptable es en la síntesis de vocales, pero tampoco es un resultado de gran calidad.

# Capítulo 6

# Adaptación de los resultados de la síntesis consonántica al sintetizador

El presente capítulo de la memoria tiene diferentes objetivos, los cuales son:

- Por un lado, establecer cuáles serán las técnicas de síntesis de sonido a utilizar para la construcción de nuestro sintetizador.
- Por otra parte, se establecerá una nueva forma de organización del fichero de orquesta para poder incluir todos los nuevos instrumentos necesarios.
- Se mostrará el funcionamiento del nuevo sistema de transposición MIDI–CSound que se deberá emplear para utilización del nuevo sintetizador.
- Por último, se mostrará un pequeño ejemplo de funcionamiento del nuevo sistema.

Cabe destacar que es en este capítulo en el que se dará aplicación a todo lo estudiado y creado a lo largo de la memoria, desde el sintetizador de vocales original y el sistema de transposición MIDI–CSound para vocales hasta las diferentes técnicas empleadas para la síntesis de sonido. Es por ello que destaca la gran importancia de este capítulo en la presente memoria.

## 6.1. Elección de los métodos de síntesis de sonido a emplear

Tras la exposición de los diferentes métodos de síntesis de sonido del Capítulo 5 se hace necesario, con ayuda de las conclusiones de cada método, decidir qué técnica o técnicas se utilizarán para dar solución a nuestro problema de síntesis de fonemas, especialmente los consonánticos. Para ello, estudiaremos los resultados en síntesis vocálica y consonántica por separado.

- Fonemas vocálicos
  En cuanto a fonemas vocálicos, los métodos que mejor resultado han dado han sido la síntesis sustractiva, la síntesis por LPC, el vocoder y la síntesis por formas de onda formantes.

  De todos estos métodos se implementará la síntesis sustractiva por las siguientes razones:

  1. La síntesis sustractiva da un buen resultado en poco tiempo de proceso, además de ya estar implementada.
  2. La síntesis por LPC, a pesar de dar un buen resultado en síntesis vocálica, es bastante lenta, además de que, en calidad final, la síntesis sustractiva es capaz de superarla.
  3. El método del vocoder no proporciona la suficiente calidad en comparación con la síntesis sustractiva.
  4. La síntesis por formas de onda formantes, a pesar de ser rápida y de una calidad comparable a la síntesis sustractiva, se desecha por estar ya implementada la segunda.

- Fonemas consonánticos
  En lo relativo a síntesis de fonemas consonánticos, como se deduce del Capítulo 5, no hay gran cantidad de técnicas para la resolución de nuestro problema. Sin embargo, cabe destacar dos situaciones diferentes:

  1. Consonantes semivocálicas
     Las consonantes de este tipo, al ser bastante similares a las vocales, sí que son susceptibles de ser sintetizadas por los mismos métodos que las propias vocales. Sin embargo, estos fonemas también serán creados por medio de la síntesis sustractiva.

Las razones para esta decisión son bastante parecidas a las de los fonemas vocálicos: la síntesis sustractiva proporciona un buen resultado en cuanto a calidad y rapidez, además de que también es más sencillo modificar la síntesis vocálica para esta finalidad que no implementar sintetizadores con otras formas de síntesis desde cero.

2. Consonantes sordas
   En lo relativo a síntesis de fonemas consonánticos sordos, hemos de decantarnos por la única técnica que ha proporcionado un resultado aceptable con este tipo de fonemas, la síntesis concatenativa.

## 6.2. Modificación del fichero de orquesta

Para la inclusión de los nuevos instrumentos en la orquesta, y con la finalidad de hacer un código eficiente, se ha modificado la forma de organización de este fichero para quedar así:

1. Instrumentos vocálicos
   Para respetar la anterior implementación de este sintetizador, se ha mantenido el símil de síntesis sustractiva que implicaba que, por cada vocal a generar, tuviéramos que ejecutar dos instrumentos (el instrumento excitador y el instrumento filtro). Sin embargo, la nueva numeración es la que sigue:

   - Instrumentos excitadores: Estos instrumentos tendrán la numeración 1001, 1002, 1003 ó 1004 en función de la voz que generen (soprano – contralto – tenor – bajo).
   - Instrumentos filtros: Irán numerados según la distancia que tenga la vocal a sintetizar, en ASCII, a la vocal 'a' más una unidad, además de ir precedidos del número de pista que les corresponde, lo que establece qué voz es. Es decir, los instrumentos serán x01, x05, x09, x15 y x21 (representando 'a', 'e', 'i', 'o', 'u') y siendo x el número de pista (1, 2, 3 ó 4).

2. Instrumentos consonánticos
   Los instrumentos consonánticos, al no utilizar todos la síntesis sustractiva como en el caso anterior, se han implementado como un único instrumento cada consonante.

La numeración de estos instrumentos consiste, como en las vocales, en la distancia a la que se encuentran de la vocal 'a', más una unidad, pero sin ir precedidos de la pista porque, partiendo de la hipótesis de que las consonantes no tienen altura, no es necesario diferenciar en función de la voz a sintetizar. Cabe destacar que ciertos sonidos como son la 'ñ' o la 'rr' no siguen la regla de la distancia por no encontrarse en ASCII como una consonante de por sí, por lo que se les da un número de instrumento mayor al de la mayor distancia que se pueda tener, concretamente mayor que 26 que corresponde a la distancia de la consonante 'z' con la 'a' añadiendo una unidad. Los fonemas incluidos por ahora y que siguen esta regla son:

| Fonema | Letra | Número de instrumento |
|---|---|---|
| r | rr | 27 |
| ɲ | ñ | 28 |

3. Instrumento de reverberación
   Con la finalidad de que siempre sea el intrumento con el número mayor de todos, se le asigna, en esta orquesta, el número de instrumento 9999.

Si se desea conocer la numeración exacta de cada una de las consonantes en el sintetizador, se recomienda la lectura del Apéndice C.

## 6.3. Modificación del sistema de control del sintetizador

En general, la forma de control del sintetizador con consonantes sigue exactamente igual que en el sintetizador de vocales pero sustituyendo básicamente las funciones de asignación de instrumento, que ahora deberán considerar que puede haber más de una letra por cada metadato de texto (en definitiva, por cada nota) y que, además, dentro de cada metadato del tipo citado anteriormente, podemos encontrar cualquier letra del abecedario, teniendo también que considerar casos especiales de ciertas consonantes (por ejemplo, la letra 'c' no se comporta igual si precede a una 'e' o a una 'u').

Cabe destacar que, al incluir consonantes y más de una vocal en cada nota, se pueden dar situaciones gramaticales bastante complejas, las cuales se han limitado por ser un problema inabarcable en el presente proyecto. Las situaciones que se han resuelto son:

1. Vocal individual

2. Diptongo
3. Consonante + Vocal
4. Consonante + Vocal + Consonante
5. Consonante + Vocal + Vocal
6. Consonante + Vocal + Vocal + Consonante

El diagrama de funcionamiento general del *back-end* sigue siendo exactamente el mismo que antes, por lo que se hace innecesario el repetirlo aquí. El verdadero cambio viene en la función *poner_instrumentos* de asignación de instrumento en función del metaevento de texto, cuyo diagrama de bloques ahora es el que se muestra en la figura 6.1.

Los diagramas de bloques de las partes catalogadas como *Caso 1* y *Caso 2* son los que se muestran en las figuras 6.2 y 6.3.

En este diagrama de bloques lo que se está realizando es constantemente una evaluación de cada una de las letras para saber cómo actuar a la hora de, por ejemplo, establecer la duración de cada letra, su tiempo de inicio... Cabe destacar que este algoritmo funcionará de forma correcta siempre que los casos que se encuentren dentro de la sílaba sean los comentados anteriormente, no pudiendo asegurar un funcionamiento correcto en cualquier otro caso.

Antes de continuar destacaremos ciertos detalles del algoritmo anterior que no se entienden a primera vista pero que son fundamentales para la comprensión del mismo:

- La variable *letra* sirve de iterador para el bucle por número de letras.
- La variable *voc_junta* y el correspondiente caso del condicional sirven para el caso de tener un diptongo.
- La variable *eliminar* y el correspondiente caso del condicional sirven para eliminar ciertas vocales o consonantes que no han de sonar.
- La variable *dur_menos* sirve para quitar parte de duración de algunos sonidos y así poder introducir otros como, por ejemplo, en el caso de tener una consonante tras una vocal en la que quitamos parte de la duración de la vocal para dársela a la consonante.

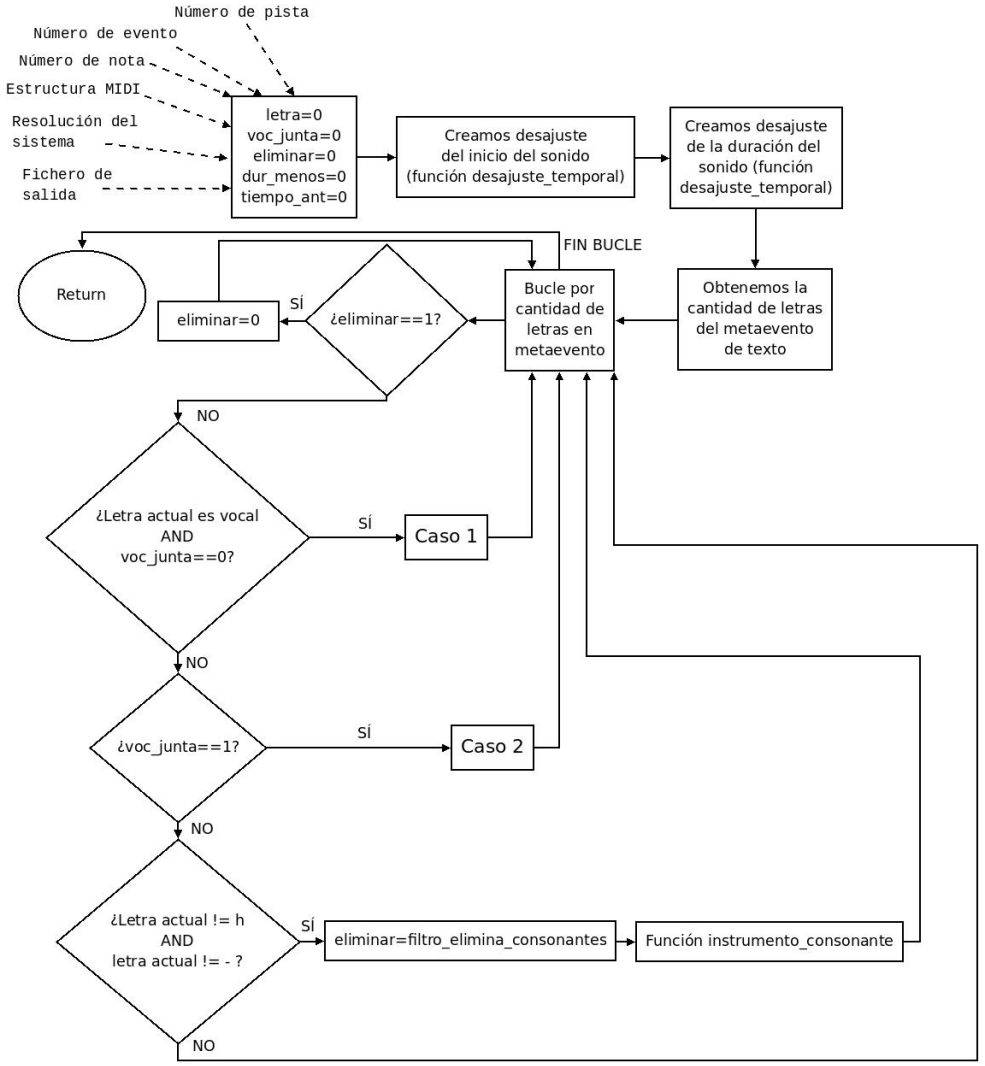


Figura 6.1: Diagrama de bloque de la función *poner_instrumentos* con consonantes.

- La variable *tiempo_ant* sirve para especificar cuándo ha de comenzar cierto sonido como, por ejemplo, el instante en que comienza la segunda consonante en un caso de Consonante + Vocal + Consonate.
- También cabe destacar que, en el caso de poner consonantes, tenemos dos particularidades:
  1. Cuando aparece una *h* la ignoramos y vamos a la siguiente letra.
  2. Cuando aparece una *r* precedida del signo '–' es porque queremos

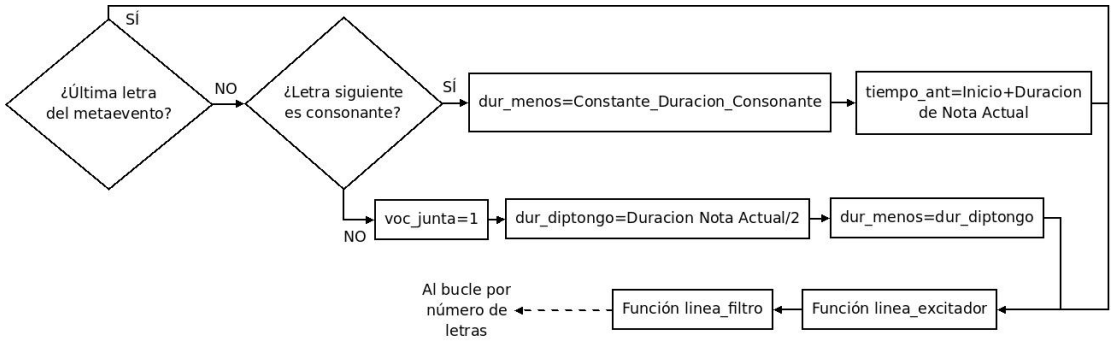


Figura 6.2: Diagrama de bloques del *Caso 1*.

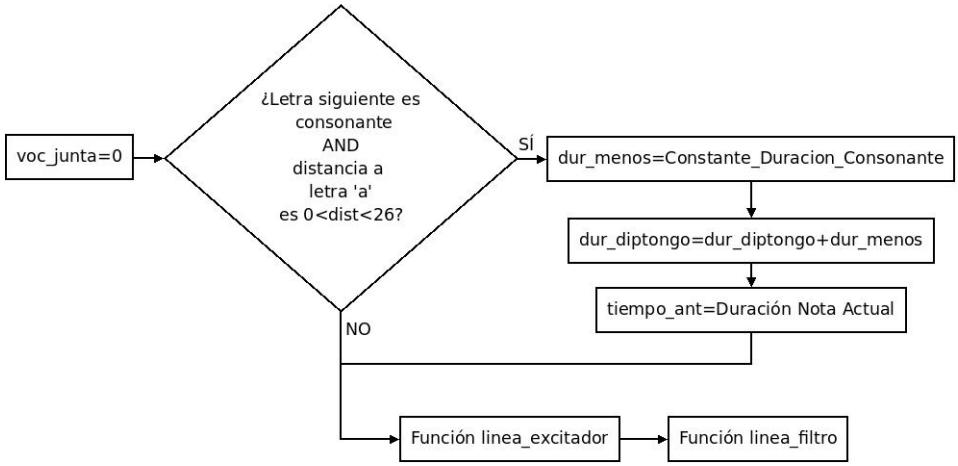


Figura 6.3: Diagrama de bloques del *Caso 2*.

que suene como ɾ (es decir, la *r* de "cara") mientras que si va sin nada, el sonido conseguido será r (es decir, la *r* de "rata").

- Lo que aparece como *Constante_Duracion_Consonante* es una constante del programa (declarada como *define*) que establece la duración de las consonantes, quedando por tanto la duración de las consonantes sujeto al criterio del usuario[1].

Una vez explicados estos detalles acerca del algoritmo pasamos a comentar el resto de funciones que se incluyen en los diferentes diagramas de bloques

[1]Esto sólo tiene efecto en consonantes que realmente se sinteticen puesto que las que se crean por medio de samplers tendrán siempre la misma duración por mucho que cambiemos ese valor.

presentados:

1. Función *vocaloconsonante*
   Esta función tiene la única finalidad de devolver un 0 si la letra que le mandamos es una vocal o un 1 si no lo es[2].

   El funcionamiento de esta función viene definido por el pseudocódigo 8.

**Algoritmo 8** Pseudocódigo de la función *vocaloconsonante*.

**Entrada:** Variable char *letra* con la letra a evaluar.
**Salida:** Variable *salida*.

```
1: Si letra==a OR letra==e OR letra==i OR letra==o OR letra==u
   Entonces
2:    salida = 0
3: Else
4:    salida = 1
5: Fin Si
6: Return salida
```

   Esta función es la que nos permite saber si una letra es vocal o consonante para así saber qué tratamiento le tiene que dar el algoritmo.

2. Función *linea_excitador*
   Esta función viene definida por el diagrama de bloques de la figura 6.4.

   Como podemos observar, esta función es prácticamente la misma que en el caso de la función de creación de las llamadas a los instrumentos de excitación del Capítulo 4 salvo por estas diferencias:

   *a*) En lugar de poner como intrumento la pista en la que nos encontramos, ponemos 100x, siendo $x$ la pista en la que estamos.

   *b*) En lugar de tener la duración de la nota más el desajuste aleatorio que producimos, también tenemos la variable *dur_menos* que quita parte de la duración si después de la vocal que se está tratando hay alguna consonante.

[2]Cabe destacar que no se puede afirmar que si devuelve un 1 tenemos una consonante porque puede ser cualquier carácter ASCII.

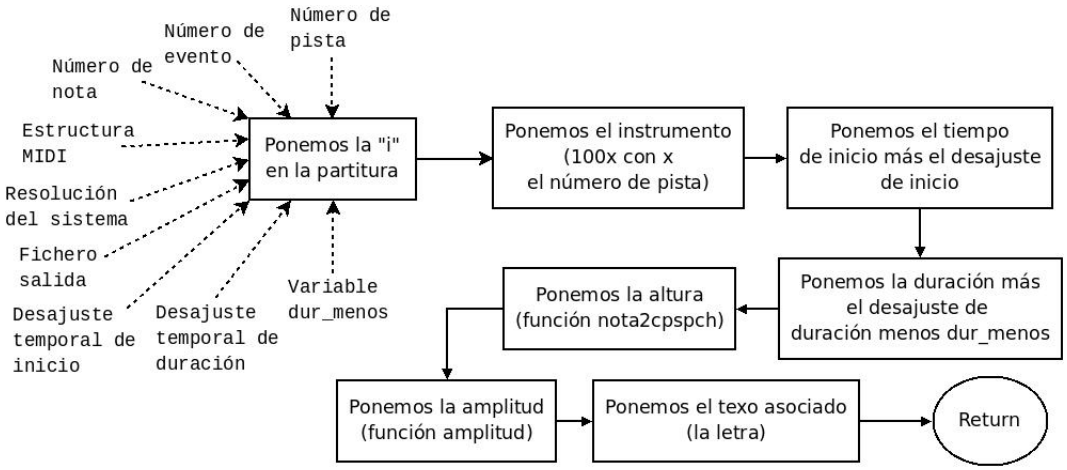


Figura 6.4: Diagrama de bloques de la función encargada de crear el instrumento excitador con consonantes en la partitura.

3. Función *linea_filtro*
   Esta función viene definida por el diagrama de bloques de la figura 6.5.

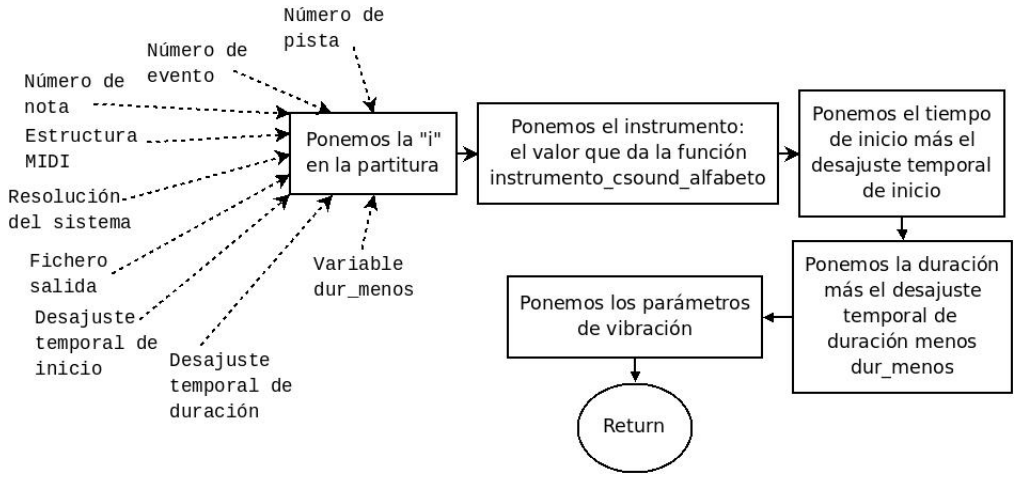


Figura 6.5: Diagrama de bloques de la función encargada de crear el instrumento filtro con consonantes en la partitura.

Como ocurre en el caso de la función *linea_excitador*, en ésta no hay grandes cambios respecto a la función del Capítulo 4, salvo los que enumeramos a continuación:

*a*) Cambio de la función *instrumento_csound* de asignación de instrumentos por la función *instrumento_csound_alfabeto* que incluye tratamiento de consonantes.

*b*) Como en la función *linea_excitador*, también se incluye la variable *dur_menos* por si después de la vocal que se está tratando tenemos una consonante.

Como último apunte de este apartado, la función *instrumento_csound_alfabeto* viene descrita por el pseudocódigo 9.

**Algoritmo 9** Pseudocódigo de la función *instrumento_csound_alfabeto*.

**Entrada:** Variable int *pista* con la pista, variable char *letra* con la letra y archivo de salida.

**Salida:** Nada.

```
1: letra_a = (int)'a'
2: dist = letra - letra_a+1
3: Si pista == 0 Entonces
4:    Si dist<10 Entonces
5:       ponemos instrumento como pista0dist
6:    Else
7:       ponemos instrumento como pistadist
8:    Fin Si
9: Else
10:   Si dist<10 Entonces
11:      ponemos instrumento como 0dist
12:   Else
13:      ponemos instrumento como dist
14:   Fin Si
15: Fin Si
16: Return
```

Como podemos observar, con esta función lo que implementamos es que el instrumento tenga siempre una estructura de tres números, de los cuales el más a la izquierda sea la pista en la que nos encontramos y los otros dos la distancia a la vocal 'a' más una unidad. Cabe destacar que, cuando la distancia calculada tiene menos de dos cifras, hemos de insertar un cero entre la pista y la distancia para así cumplir la estructura propuesta.

4. Función *filtro_elimina_consonantes*
   La finalidad de esta función es la de hacer que, en la siguiente iteración, la función *poner_instrumentos* descarte poner en la partitura la consonante que sigue en la siguiente posición del puntero al metaevento de texto. Aunque puede parecer que esta función no tenga utilidad, se ha utilizado principalmente para eliminar la vocal 'u' en casos de sílabas del tipo "gue", "que", "gui" o "qui".

   Esta función viene definida por el pseudocódigo 10.

**Algoritmo 10** Pseudocódigo de la función *filtro_elimina_consonantes*.

**Entrada:** Puntero **ptr* a metaevento de texto y variable entera *letra* con la posición de la letra actual.
**Salida:** Variable *eliminar*.

```
1: eliminar = 0
2: Si ((ptr[letra]==q OR ptr[letra]==g) AND ptr[letra+1]==u) AND
   (ptr[letra+2]==e OR ptr[letra+2]==i) Entonces
3:    eliminar = 1
4: Fin Si
5: Si ptr[letra]==r AND ptr[letra+1]==r Entonces
6:    eliminar = 1
7: Fin Si
8: Si ptr[letra]==n AND ptr[letra+1]==y Entonces
9:    eliminar = 1
10: Fin Si
11: Si ptr[letra]==l AND ptr[letra+1]==l Entonces
12:    eliminar = 1
13: Fin Si
14: Return eliminar
```

Como podemos observar, para ciertos casos como es una 'u' entre una 'g' o una 'q' y una 'e' o 'i' u otros casos como 'ny' (la 'ñ' se ha tenido que modelar de esta forma por no estar presente en el código ASCII) o la 'll', la segunda consonante o bien no ha de sonar o bien causa un efecto sobre la primera cambiando su forma de sonar. Sin embargo, lo que sí que está claro es que esas letras no han de sonar, y de ello es de lo que se encarga esta función.

5. Función *instrumento_consonante*
   La función *instrumento_consonante* se puede representar con el diagrama de bloques de la figura 6.6.

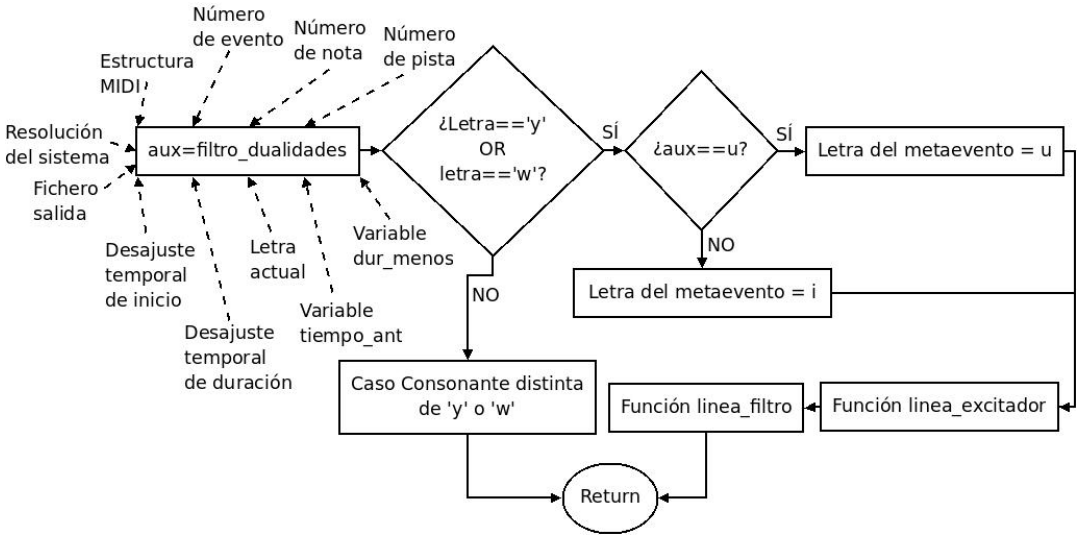


Figura 6.6: Diagrama de bloques de la función *instrumento_consonante*.

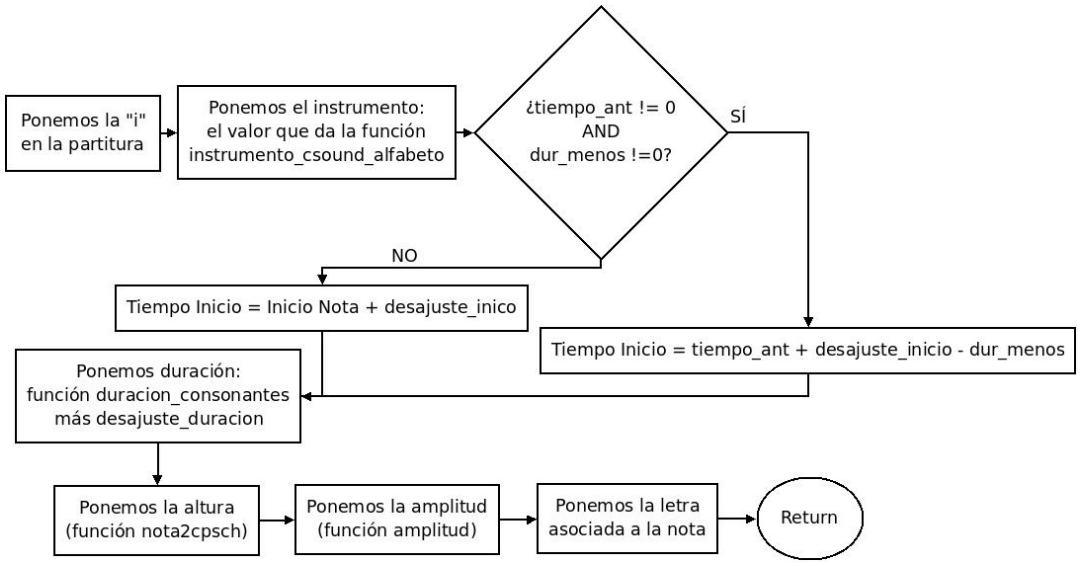


Figura 6.7: Diagrama de bloques del caso en que la consonante no es 'w' ni 'y'.

Como podemos observar, lo primero que realizamos al entrar en esta función es llamar a la función *filtro_dualidades*, cual se encarga de decidir qué consonante es la que realmente ha de sonar[3]. Tras ello el algoritmo discrimina entre dos casos, los cuales comentaremos a continuación:

*a*) Caso en el que estamos con la consonante 'w' o 'y', las cuales vamos a modelar como las vocales 'u' e 'i' respectivamente por medio de la síntesis sustractiva[4], pero con una duración de consonante[5].

*b*) Cualquier otra consonante, las cuales se modelarán como un único instrumento.

Una vez realizada esa distinción, si la consonante no es ni la 'w' ni la 'y', lo que el algoritmo realiza lo que manda el algoritmo de la figura 6.7, que en definitiva es una mezcla de las funciones *linea_filtro* y *linea_excitador*. Cabe destacar la importancia del condicional que se puede ver en el diagrama de bloques: gracias a él establecemos el momento de inicio de la consonante en cuestión, mientras que la duración de la misma viene dada por la función *duracion_consonantes*, que, de acuerdo con el pseudocódigo de la misma, simplemente establece que las consonantes realmente sintetizadas tengan la duración establecida por la constante *Constante_Duracion_Consonantes* mientras que las otras pueden tener la duración de toda la nota por ser simplemente muestras de sonido[6].

Finalmente, las funciones *duracion_consonantes* y *filtro_dualidades* vienen definidas por los pseudocódigos 11 y 12, respectivamente.

[3]Recordemos que, por ejemplo, una 'c' no ha de sonar igual delante de una 'a' que de una 'e'.

[4]Utilizando tanto la función *linea_excitador* como *linea_filtro*.

[5]Definida por *Constante_Duracion_Consonante*.

[6]Si no realizáramos esa distinción, las consonantes sintetizadas de cero tendrían toda la duración de la nota, con lo que no obtendríamos un resultado correcto.

**Algoritmo 11** Pseudocódigo de la función *duracion_consonantes*.

**Entrada:** Variable flotante con la duración inicial y variable char *consonante* con la letra a evaluar.

**Salida:** Variable *salida*.

```
1: Si consonante == m OR consonante == n OR consonante == l Entonces
2:    salida = Constante_Duracion_Consonante
3: Else
4:    salida = Duración Inicial
5: Fin Si
6: Return salida
```

## 6.4. Ejemplo de funcionamiento

Para finalizar, y como ya se hizo en el Capítulo 4, vamos a mostrar un ejemplo de funcionamiento del sistema de transposición MIDI-CSound con la capacidad de tratar consonantes ya incluida.

La partitura que se va a utilizar en este caso es la que se muestra en la figura 6.8.

Por la limitación de nuestro sintetizador consonántico[7], en realidad la sílaba *blet* aparecerá en la partitura de CSound como *bet*.

Este extracto de partitura, en un secuenciador, tendría la forma mostrada en la figura 6.9.

Los diferentes diagramas de pianola de las diferentes pistas los podemos ver en la figura 6.10.

Finalmente, el resultado de la transposición de la partitura secuenciada es el que sigue:

```
;Datos presentes en el archivo midi acerca del contenido del mismo:

;Partitura extraida del archivo
;Cambios de tempo:
t 0 96 ; Hay 1 tempo(s)

f1 0 8192 10 1
```

[7]Debemos recordar que la implementación realizada no tiene la capacidad de sintetizar dos fonemas consonánticos juntos.

**Algoritmo 12** Pseudocódigo de la función *filtro_dualidades*.

**Entrada:** Variable int *letra* con la posición y variable puntero char **texto* con el metaevento de texto.

**Salida:** Variable *salida*.

```
1: Switch texto[letra]
2:   Case c:
3:     Si texto[letra+1]==e OR texto[letra+1]==i
4:       salida = z
5:     Else
6:       salida = k
7:     Fin Si
8:   Case g:
9:     Si texto[letra+1]==e OR texto[letra+1]==i
10:      salida = j
11:    Else
12:      salida = g
13:    Fin Si
14:  Case l:
15:    Si texto[letra+1]==l
16:      salida = y
17:    Else
18:      salida = l
19:    Fin Si
20:  Case n:
21:    Si texto[letra+1]==y
22:      salida = (char)((int)z+2)
23:    Else
24:      salida = n
25:    Fin Si
26:  Case q:
27:    salida = k
28:  Case r:
29:    Si texto[letra+1]==r OR (texto[letra-1] es consonante AND
       texto[letra-1]!='-')
30:      salida = (char)((int)z+1)
31:    Else
32:      salida = r
33:    Fin Si
34:  Default:
35:    salida = texto[letra]
36: Fin Switch
37: Return salida
```

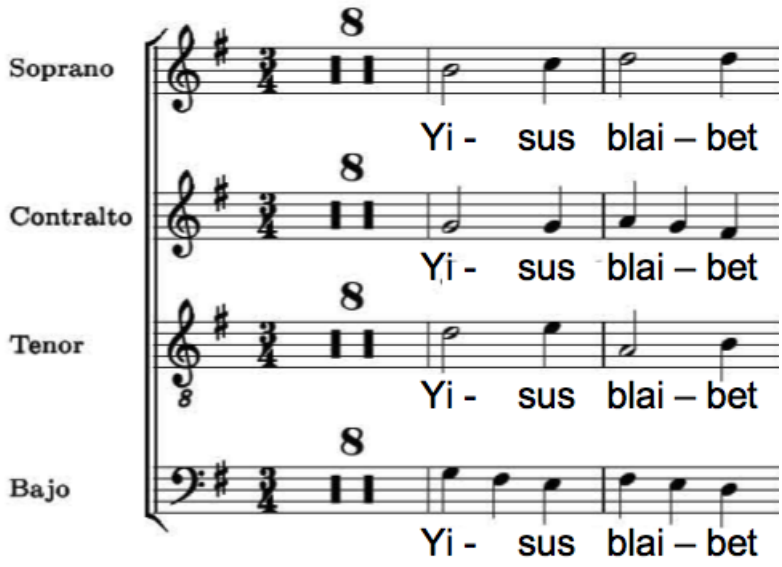


Figura 6.8: Extracto del 10º Movimiento de la obra *Herz und Mund und Tat und Leben* (BWV 147) de Johann Sebastian Bach, conocido como *Jesu, Joy of Man's Desiring*.


Figura 6.9: Detalle del extracto de partitura de la figura 6.8 en el secuenciador Apple Logic Pro 8.

```
f2 0 8192 9 1 1 .25
i9999 0 51 0.95 0.95 0.8 2000 1

;Soprano
;Instrumento    Tinicio      Duracion     Frecuencia    Amplitud    Letra
i        1001   36.000000    0.300000     8.110000      75.83       ;/i/
i        109    36.000000    0.300000     1             10          3.000000
i        1001   36.000000    3.000000     8.110000      77.28       ;/i/
i        109    36.000000    3.000000     1             10          3.000000
```

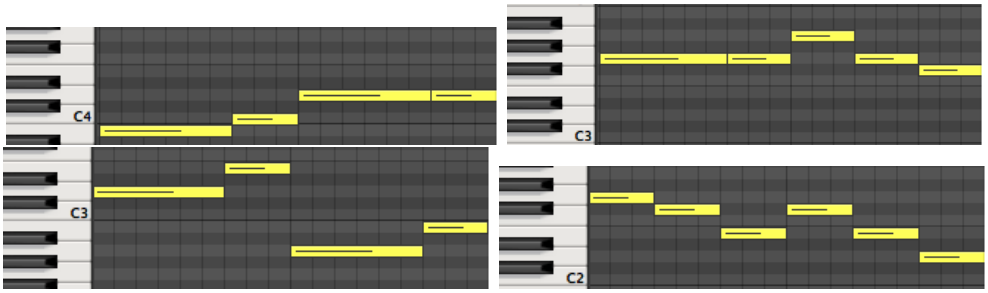


Figura 6.10: Detalle los diagramas de pianola de las diferentes pistas: soprano (arriba izda.), contralto (arriba dcha.), tenor (abajo izda.) y bajo (abajo dcha.).

```
i       19      39.000000   1.500000    9.000000    72.81       ;/s/
i       1001    39.000000   1.200000    9.000000    71.22       ;/u/
i       121     39.000000   1.200000    1           10          1.500000
i       19      40.200000   1.500000    9.000000    69.09       ;/s/

i       02      40.500000   3.000000    9.020000    67.86       ;/b/
i       1001    40.500000   1.500000    9.020000    67.67       ;/a/
i       101     40.500000   1.500000    1           10          3.000000
i       1001    42.000000   1.500000    9.020000    68.33       ;/i/
i       109     42.000000   1.500000    1           10          3.000000

i       02      43.500000   1.500000    9.020000    69.01       ;/b/
i       1001    43.500000   1.200000    9.020000    70.31       ;/e/
i       105     43.500000   1.200000    1           10          1.500000
i       20      44.700000   1.500000    9.020000    67.43       ;/t/

;Contralto
;Instrumento    Tinicio     Duracion    Frecuencia  Amplitud    Letra
i       1002    36.000000   0.300000    8.070000    72.91       ;/i/
i       209     36.000000   0.300000    1           10          3.002083
i       1002    36.000000   3.002083    8.070000    76.81       ;/i/
i       209     36.000000   3.002083    1           10          3.002083

i       19      36.000000   3.002083    8.070000    76.21       ;/s/
i       1002    36.000000   2.702083    8.070000    76.94       ;/u/
i       221     36.000000   2.702083    1           10          3.002083
i       19      38.702083   3.002083    8.070000    75.20       ;/s/

i       02      40.500000   1.502083    8.090000    72.37       ;/b/
i       1002    40.500000   1.502083    8.090000    76.26       ;/a/
i       201     40.500000   1.502083    1           10          1.502083
```

```
i       1002    42.000000    1.502083    8.070000      66.63       ;/i/
i       209     42.000000    1.502083    1             10          1.502083

i       02      43.500000    1.502083    8.060000      71.79       ;/b/
i       1002    43.500000    1.202083    8.060000      68.11       ;/e/
i       205     43.500000    1.202083    1             10          1.502083
i       20      44.702083    1.502083    8.060000      71.47       ;/t/

;Tenor
;Instrumento    Tinicio      Duracion    Frecuencia    Amplitud    Letra
i       1003    36.000000    0.300000    8.020000      74.78       ;/i/
i       309     36.000000    0.300000    1             10          3.010417
i       1003    36.000000    3.010417    8.020000      75.85       ;/i/
i       309     36.000000    3.010417    1             10          3.010417

i       19      40.500000    3.010417    7.090000      71.96       ;/s/
i       1003    40.500000    2.710417    7.090000      67.79       ;/u/
i       321     40.500000    2.710417    1             10          3.010417
i       19      43.210418    3.010417    7.090000      67.52       ;/s/

i       02      40.500000    3.010417    7.090000      67.61       ;/b/
i       1003    40.500000    1.505208    7.090000      67.60       ;/a/
i       301     40.500000    1.505208    1             10          3.010417
i       1003    42.005208    1.505208    7.090000      69.13       ;/i/
i       309     42.005208    1.505208    1             10          3.010417

i       02      43.500000    1.468750    7.110000      71.26       ;/b/
i       1003    43.500000    1.168750    7.110000      70.11       ;/e/
i       305     43.500000    1.168750    1             10          1.468750
i       20      44.668750    1.468750    7.110000      68.38       ;/t/

;Bajo
;Instrumento    Tinicio      Duracion    Frecuencia    Amplitud    Letra
i       1004    36.000000    0.300000    7.070000      75.43       ;/i/
i       409     36.000000    0.300000    1             10          1.500000
i       1004    36.000000    1.500000    7.070000      74.84       ;/i/
i       409     36.000000    1.500000    1             10          1.500000

i       1004    37.500000    1.500000    7.060000      67.28       ;/i/
i       409     37.500000    1.500000    1             10          1.500000

i       19      39.000000    1.500000    7.040000      67.64       ;/s/
i       1004    39.000000    1.200000    7.040000      66.09       ;/u/
i       421     39.000000    1.200000    1             10          1.500000
i       19      40.200000    1.500000    7.040000      71.19       ;/s/

i       02      40.500000    1.500000    7.060000      66.06       ;/b/
i       1004    40.500000    1.500000    7.060000      69.27       ;/a/
i       401     40.500000    1.500000    1             10          1.500000
```

```
i       1004    42.000000   1.500000    7.040000    65.59       ;/i/
i       409     42.000000   1.500000    1           10          1.500000

i       02      43.500000   1.500000    7.020000    69.77       ;/b/
i       1004    43.500000   1.200000    7.020000    71.06       ;/e/
i       405     43.500000   1.200000    1           10          1.500000
i       20      44.700000   1.500000    7.020000    69.08       ;/t/
```

Cabe destacar que en este caso, al igual que en el ejemplo del Capítulo 4, los desajustes temporales han sido puestos a cero para una mejor comprensión de la propia transformación[8].

Por último, algo que no se ha comentado durante el diseño del *back-end*, es que las consonantes creadas por samplers, a pesar de que lo único que realizan los instrumentos encargados de introducirlas en la composición es leer un fichero de sonido, realizar un pequeño tratado de ese archivo (filtrado y ajuste de volumen, con ocasional utilización de envolvente de amplitud), la línea encargada de realizar la llamada al instrumento desde la partitura incluye los mismos parámetros que si de una consonante que se fuera a sintetizar desde cero (como, por ejemplo, la *m*). El no realizar una distinción de llamadas entre estos dos tipos de instrumentos es por simple comodidad, ya que así el instrumento que necesite esos parámetros los podrá obtener, mientras que al instrumento que no le hagan falta, los ignorará por completo.

[8]Los números con gran cantidad de decimales aparecen por pequeños fallos a la hora de secuenciar la partitura y no haber realizado una cuantización.

# Capítulo 7

# Conclusiones

En este capítulo se pretende llevar a cabo una valoración de los resultados obtenidos tras el desarrollo del presente proyecto, junto con el grado de consecución de los objetivos planteados al inicio del mismo. También, por ser de interés para la valoración del resultado final, se adjuntarán datos acerca del desarrollo del proyecto que no tienen cabida en ningún otro punto de la memoria.

## 7.1. Revisión de los objetivos iniciales

Para realizar esta revisión primero es recomendable recordar, aunque sea de forma esquemática, los objetivos iniciales del presente proyecto:

- Creación de un software de control del sintetizador mediante el estándar MIDI.
- Síntesis de sonidos vocálicos.
- Estudio de las diferentes técnicas de síntesis de sonido aplicado a la síntesis de consonantes.
- Síntesis de sonidos consonánticos.

Comparando lo que se ha descrito en la presente memoria (los diferentes capítulos) con los objetivos (tanto los resumidos ahora como los comentados al comienzo de la memoria), se puede afirmar que sí que se ha llevado a cabo todo aquello que se proponía realizar.

Sin embargo, sí que es importante el realizar una comparación entre resultados esperados, al menos personalmente, con los obtenidos finalmente de cada uno de estos puntos:

- Sistema de control mediante MIDI.
  El sistema de control del sintetizador, al menos en un principio, se suponía que iba a resultar un sistema de conversión de notación MIDI a notación de CSound que simplemente implementara una transposición de los datos presentes en la secuencia MIDI a la partitura, sin añadir nada más. En cierta forma, esto sigue siendo así (sin ir más lejos, siempre estamos buscando metadatos de tipo letra asociados a una nota para extraer su momento de inicio, su duración, su altura, la letra a la que hace referencia para así definir el instrumento necesario...) pero cabe destacar que también se han incluido ciertos elementos que, al menos por mi parte personal, no había considerado en un momento inicial, los cuales son las diferentes funciones que intentan dar cierta naturalidad a la composición como los desajustes temporales (tanto de momento de inicio como de duración de nota), las variaciones aleatorias de amplitud de cada nota o la interpretación de la métrica de cada composición para realizar, aunque sea con una primera aproximación bastante simple, un pequeño acercamiento a lo que un músico puede extraer al leer una partitura.

- Síntesis de sonidos vocálicos.
  En lo relativo a síntesis de sonidos de tipo vocálico, con la existencia del sintetizador de vocales del que se parte en el presente proyecto, no ha sido necesaria una implementación del mismo, sino simplemente un estudio de su funcionamiento (algo básico para poder manipularlo después) y una pequeña modificación del comentado sintetizador para permitir posteriormente la incorporación de los sonidos de tipo consonántico. Por tanto, en este caso, los resultados esperados coinciden con los resultados obtenidos, puesto que no ha habido ninguna modificación.

- Estudio de las diferentes técnicas de síntesis de sonido.
  El estudio de las diferentes técnicas de síntesis de sonido puede considerarse suficiente o no en función de cómo se valoren las conclusiones obtenidas: de todas las formas de síntesis estudiadas, las dos únicas que han proporcionado un resultado positivo han sido la síntesis sustractiva para ciertas consonantes (la $m$, la $n$ y la $l$, además de la $y$ y la $w$ considerando que se han modelado como las vocales $i$ y $u$), mientras que para el resto se ha llegado a la conclusión de que lo mejor es la síntesis por sonidos muestreados. Esta conclusión, correcta o no, es válida para este caso en particular y quizás en otras circunstacias no sea para nada válida (por ejemplo, utilizando otra forma de síntesis que no sea

CSound o considerando otro final que no sea la síntesis de voz coral).

- Síntesis de sonidos consonánticos.
  Aunque parezca que hace referencia a lo mismo que el apartado anterior, estamos refiriéndonos a algo distinto ya que, mientras que anteriormente considerábamos la síntesis pura de cada uno de los sonidos consonánticos, en este apartado nos referimos, además de a eso, a cómo se relacionan con el resto de sonidos, tanto con otras consonantes como con otras vocales.

  En lo referente a este aspecto, también se ha conseguido llevar a cabo esta tarea, realizando la unión de síntesis de consonantes y vocales con cierto grado de calidad. Sin embargo, los problemas más destacables que se ha tenido y que, aunque en cierta forma se han paliado, no se han conseguido solucionar son los que siguen:

  1. La inteligibilidad del sistema, aún no siendo lo prioritario al ser voz cantada, no es muy exitosa y no permite el buen entendimiento de lo que se está tratando de transmitir en la mayoría de casos. Esto puede deberse al problema del solapado entre consonantes y vocales, además de la propia calidad individual de cada uno de los fonemas sintetizados. También es destacable en este sentido el hecho de que, al menos según parece tras la realización de gran cantidad de pruebas, los cambios de altura entre sílabas de una palabra (y entre palabras) no ayudan a la inteligibilidad en general, lo cual refuerza la teoría de que en voz coral, la inteligibilidad es más compleja que en voz hablada, en la cual los cambios de altura no son tan acusados.
  2. Las consonantes sintetizadas por samplers producen ciertos ruidos que, a pesar de la utilización de diferentes técnicas (filtrado de la muestra, aplicación de envolventes de amplitud...), no se consiguen eliminar del todo.
  3. El volumen de las consonantes parece tener que ser distinto para cada caso ya que, mientras que en algunas obras el volumen es el correcto y las consonantes se pueden distinguir, en otras parecen quedar o muy altas o muy bajas, no obteniendo, por tanto, el resultado óptimo. Esto, en cierta forma, también se puede deber a los desajustes aleatorios de inicio de cada una de las notas, haciendo que, si en un instante las cuatro voces cantan exactamente la misma sílaba, en ocasiones el desajuste propiciará que las formas de onda coincidan de una forma constructiva (aumentando,

por tanto, el volumen individual de la consonante) o de una forma destructiva (disminuyendo ese volumen), pero ocurriendo esto siempre de una forma aleatoria.

## 7.2. Evolución del proyecto

En lo referente a este apartado, podemos ver la evolución del proyecto en dos bloques diferenciados por mayor sencillez, a pesar de que en la práctica están unidos:

1. Creación del software de control.
   La creación del software de control, en un primer desarrollo para el sintetizador de vocales, y en parte gracias al diseño restringido que se propuso, se pudo implementar de forma relativamente sencilla. Sin embargo, tras la inclusión de las consonantes en el sistema, debido en gran medida a la ambigüedad que conllevan[1], además de que a partir de este momento cada metadato de texto ya podía contener más de una letra, el sistema pasó a ser bastante más complejo por la gran cantidad de elementos a considerar. Finalmente, la inclusión de elementos de no simplemente transposición de datos, como la interpretación de la métrica o la inclusión de variaciones aleatorias de la amplitud y los tiempos de inicio y duración de cada nota, añadieron, además, un grado de dificultad importante al sistema.

   Como último apunte en el desarrollo de este software cabe destacar que se han tenido que utilizar dos secuenciadores distintos por ciertos problemas: el primer secuenciador utilizado, MusE, funcionaba de forma totalmente correcta hasta la implementación de la interpretación de la métrica, cuando se vio que no secuenciaba correctamente esos datos al poner métricas como -1/-1 ó 0/255. Ante eso se decidió migrar al sistema Logic, el cual, a pesar de haber dado ciertos problemas con la secuenciación de ciertas notas (al realizar la transposición de MIDI a partitura, algunas notas variaban su posición), consiguió corregir los problemas relacionados con la métrica, además de permitir realizar otras operaciones de forma mucho más sencilla, como son el cambio de tempo o el cambio de métricas durante la composición.

[1] Refiriéndonos, como ya se comentó anteriormente, a que no todas las consonantes actúan de la misma forma, por ejemplo, delante de las mismas vocales.

2. Creación del software de síntesis.
   En cuanto al software de síntesis de canto coral, tras llegar a la conclusión de que la mejor respuesta iba a venir dada por la síntesis por samplers, se realizaron dos sesiones de grabación en las que se obtuvieron las diferentes consonantes incluidas en el sintetizador actualmente. Para el procesado previo de las muestras (eliminación de la reverberación de fondo, ajuste óptimo de la medida de la muestra...) se hizo uso del entorno Matlab, así como otros procesados posteriores (filtrados de señal, aplicación de envolventes...) se realizaron dentro de CSound.

   Para las consonantes sintetizadas desde cero simplemente se utilizó la misma estructura que la de la síntesis de vocales[2], pero uniendo ambos instrumentos en uno para simplificar el sistema.

## 7.3. Futuras líneas de trabajo

En lo referente a futuras ampliaciones a este proyecto, se proponen las siguientes posibles mejoras:

- Implementación de este sistema en un formato tipo plugin de audio, pudiendo llegar incluso a la creación de un instrumento virtual.
- Aumentar la cantidad/tipo de voces para dar cobertura a un mayor rango de composiciones.
- Mejora del tratamiento de las combinaciones de letras en las sílabas como, por ejemplo, dar en los diptongos duración que necesita cada letra, ya que por ahora ambas son igual de largas.
- Creación de sonidos consonánticos combinados (gr, cr, br, gl, cl, bl, dr...). Por ahora, no se considera el caso de que hayan dos consonantes seguidas.
- Mejora de las muestras en los samplers, utilizando una cámara anecoica, mejor microfonía, varios locutores...
- Mejora de la inteligibilidad del sistema.
- Implementación del sistema en, al menos, otro idioma (en teoría, sólo habría que modificar las vocales e incluir algunas consonantes).

[2]Síntesis sustractiva con instrumentos excitador y filtro.

- Estudio de una posible mejora en la síntesis de los sonidos vocálicos porque, aunque se tienen datos de los formantes para las diferentes alturas (soprano, contralto, tenor y bajo), en la práctica se está considerando que los formantes se mantienen estáticos en toda la tesitura de la voz cuando, en la práctica, varían. Se propone, por tanto, realizar una mejora en este aspecto.

- Utilización de otro sistema de control que permita codificar, además de la información necesaria para la creación del coro (alturas y letras), información propia de interpretación de partituras (por ejemplo *vibratos*, *staccatos*, *crescendos*...) para poder transportarla a la partitura de CSound y aumentar la naturalidad del sistema.

## 7.4. Conclusiones finales

Como ya se avanzaba en la propuesta de proyecto, el resultado no se esperaba que tuviera una *calidad de una voz solista* sino simplemente que fuera *capaz de articular sonidos reconocibles en el contexto del canto coral*. En gran parte, esto sí que se ha cumplido: la presencia de las consonantes en el sintetizador, aunque sea con volúmenes poco adecuados, ayuda a la inteligibilidad del sistema, además de ampliar el rango de posibilidades sonoras con respecto a una voz coral únicamente vocálica; además, el sistema de control ayuda mucho a la composición puesto que, junto con la liberación de la necesidad de conocer un lenguaje de programación como es CSound, nos permite trabajar en un ambiente más intuitivo, al menos musicalmente, como es un secuenciador, con respecto a un editor de textos, escribiendo cifras sobre inicios, fines, alturas y amplitudes de nota e intrumentos.

Sin embargo, es precisamente esa falta en el acierto de los volúmenes de cada consonante y la presencia de ciertos ruidos provocan una falta de calidad en el propio sistema que, en cierta forma, *deja con mal sabor de boca*, pero que posteriormente, si se retoma este proyecto, podrá ser mejorado.

# Apéndice A

# Elementos de CSound utilizados

En este anexo adjuntaremos cada uno de los elementos de este lenguaje que han sido necesarios para llevar a cabo la correcta implementación de este proyecto.

## A.1. Opcodes

En este apartado se pretende describir cada uno de los *opcodes* de CSound que se han utilizado en este proyecto, tanto desde un punto de vista de lo que hace (sintaxis y tipo de aplicación que realiza) como su uso particular dentro de los programas que se han creado o analizado.

1. **ampdb**
   Es un *opcode* convertidor de valor: se encarga de transformar un valor dado en decibelios a su equivalente en escala lineal, es decir, valor PCM. Su sintaxis es la que sigue:

   amplitud_lineal = **ampdb**(amplitud_db)

   Este *opcode* es bastante utilizado durante todo el programa debido a que dar un valor de amplitud en decibelios es más intuitivo que dar el valor en amplitud PCM.

2. **balance**
   Es un *opcode* modificador de señal: este *opcode* modifica el valor RMS de una señal para que se iguale al nivel RMS de otra señal. Su sintaxis es la que sigue:

asal **balance** asen1, asen2

siendo:

- asal – Señal procesada.
- asen1 – Señal a modificar su valor RMS.
- asen2 – Señal cuyo valor RMS es la referencia.

Este *opcode* lo hemos utilizado bastante para conseguir un nivel de amplitud determinado en señales que, tras ser procesadas, su nivel es muy grande. Esto por ejemplo se nos daba, en algunos instrumentos, al tratar nuestras señales de excitación con los filtros de los formantes o también cuando hemos trabajado con *opcodes* de generación de ruido cuyo nivel enmascaraba el resto de las señales sintetizadas.

3. **butterlp**
   Es un *opcode* de modificación de la señal: se trata de un filtro paso-bajo de tipo Butterworth. Su sintaxis es la que sigue:

   asal **butterlp** asen, kfrec

   siendo:

   - asal – Señal procesada.
   - asen – Señal a filtrar.
   - kfrec – Frecuencia de corte del filtro.

   Este *opcode* ha sido utilizado, por ejemplo, para filtrar altas frecuencias generadores de ruido.

4. **buzz**
   Es un *opcode* de generación de señal: genera una serie de parciales coseno armónicamente relacionados. Su sintaxis es la que sigue:

   asal **buzz** xamp, xfrec, knarm, ifn

   siendo:

   - asal – Señal sintetizada.
   - xamp – Amplitud de la señal.
   - xfrec – Frecuencia fundamental de la señal a crear.

- knarm – Número de armónicos.
- ifn – Número de tabla que contiene una onda sinusoidal.

Este *opcode* ha sido utilizado para realizar pruebas en la síntesis de sonidos de tipo sonoro.

5. **cpspch**
   Es un *opcode* conversor de altura: transforma una altura dada al formato *octava.nota* (octave point pitch-class) a hertzios. Su sintaxis es la que sigue:

   Valor_Hz = **cpspch** (Valor_pch)

   Este *opcode* ha sido ampliamente utilizado por la misma razón que el *opcode* **ampdb**: es más sencillo recordar un número que represente una octava y otro que represente la nota que el recordar la frecuencia exacta de la nota que se quiere procesar.

6. **delayr**
   Es un *opcode* de modificación de señal: permite leer una línea digital de retardo en la que la señal ha estado guardada un tiempo. Su sintaxis es la que sigue:

   ar **delayr** idlt

   siendo:

   - ar – Valor retardado.
   - idlt – Tiempo a retardar.

   Este *opcode* ha sido utilizado para crear líneas de retardo cuyo fin ha sido, sobretodo, el crear efectos de tipo vibrato.

7. **delayw**
   Es un *opcode* de modificación de señal: permite escribir la variable a la que hace referencia en una línea de retardo creada antes por el *opcode* **delayr** que siempre le precede. Su sintaxis es la que sigue:

   **delayw** asig

   siendo:

   - asig – Señal a guardar.

Junto con **delayr**, muy utilizado para la creación de líneas de retardo.

8. **diskin**
   Es un *opcode* de entrada de señal: permite introducir audio de fichero externos a CSound, dando opción, además, a transponer su altura. Su sintaxis es la que sigue:

   ar **diskin** ifilcod, kpitch

   siendo:

   - ar – Señal introducida en la orquesta.
   - ifilcod – Fichero externo en el que se encuentra la señal a introducir.
   - kpitch – Factor por el cual se incrementa la altura.

   Este *opcode*, junto con **soundin**, ha sido utilizado para la introducción de muestras de sonido dentro de la orquesta en la síntesis por sonidos muestreado.

9. **deltapi**
   Es un *opcode* de modificación de señal: permite extraer un valor de una línea de retardo a la que se está haciendo referencia (normalmente mediante **delayr** o alguna variante). Utiliza interpolación para la obtención del valor. Su sintaxis es la que sigue:

   ar **deltapi** xdlr

   siendo:

   - ar – Valor devuelto.
   - xdlt – Tiempo que ha de esperar para sacar la muestra (coge la muestra retrasada xdlt unidades de tiempo). Va en segundos.

   Junto con **delayr** y **delayw**, utilizado para la extracción de valores de las líneas de retardo en la creación de vibratos.

10. **fof**
    Es un *opcode* generador de señal: implementa la síntesis de formas de onda formantes (FOF) mediante la síntesis granular. Su sintaxis es la que sigue:

ar **fof** xamp, xfund, xform, koct, kband, kris, kdur, kdec, iolaps, ifna, ifnb, itotdur

siendo:

- ar – Señal de salida.
- xamp – Amplitud de pico.
- xfund – Frecuencia fundamental.
- xform – Frecuencia del formante.
- koct – Índice de octavación (normalmente 0).
- kband – Ancho de banda del formante.
- kris – Parámetro de tiempo de ataque de una envolvente de tipo ASR que se aplica al formante. En voz típicamente vale 0.003.
- kdur – Parámetro duración total de una envolvente de tipo ASR que se aplica al formante. En voz típicamente vale 0.02.
- kdec – Parámetro de tiempo de caída de una envolvente de tipo ASR que se aplica al formante. En voz típicamente vale 0.007.
- iolaps – Número de espacios preasignados, necesarios para alojar los datos del solapamiento de las explosiones.
- ifna – Tabla de onda utilizada para sintetizar explosiones sinusoidales (típicamente una tabla de tipo **GEN** 10).
- ifnb – Tabla para dibujar el ataque y la caída de las explosiones sinusoidales (típicamente una tabla **GEN** 9 ó 19).
- itotdur – Tiempo que el *opcode* **fof** estará activo.

Este *opcode* ha sido utilizado en la síntesis mediante las rutinas **fof** de CSound.

11. **fmvoice**
    Es un *opcode* generador de señal que permite sintetizar canto vocal mediante síntesis FM. Su sintaxis es la que sigue:

    asen **fmvoice** kamp, kfreq, kvowel, ktilt, kvibamt, kvibrate, ifn1, ifn2, ifn3, ifn4, ivibfn

    siendo:

    - asen – Señal sintetizada.

- kamp – Amplitud de la señal.
- kfreq – Frecuencia de la nota.
- kvowel – Vocal a sintetizar (rango 0  64).
- ktilt – Ajuste espectral del sonido (rango 0  99).
- kvibamt – Profundidad del vibrato.
- kvibrate – Frecuencia del vibrato.
- ifn1 – Tabla de ondas 1.
- ifn2 – Tabla de ondas 2.
- ifn3 – Tabla de ondas 3.
- ifn4 – Tabla de ondas 4.
- ivibfn – Tabla de onda para producir el vibrato (típicamente una sinusoide).

Este *opcode* ha sido utilizado para realizar diferentes pruebas en la síntesis de sonidos de tipo vocal.

12. **gauss**
Es un *opcode* generador de ruido blanco de distribución gaussiana. Su sintaxis es la que sigue:

xsen **gauss** krango

siendo:

- xsen – Señal generada.
- krango – Rango de la señal a generar (de krango a +krango).

Este *opcode* ha sido utilizado para generar ruido para la síntesis de sonidos sordos. Se prefiere esta distribución (gaussiana) por adecuarse mejor a la de la voz humana.

13. **gbuzz**
Es un *opcode* generador de señal: genera una serie de parciales coseno armónicamente relacionados. Es muy parecido al *opcode* **buzz**, pero el que estamos tratando aquí varía la amplitud de cada parcial según nosotros digamos. Su sintaxis es la que sigue:

asal **gbuzz** xamp, xfrec, knarm, kmarm, kr, ifn

siendo:

- asal – Señal sintetizada.
- xamp – Amplitud de la señal.
- xfrec – Frecuencia fundamental de la señal a crear.
- knarm – Número de armónicos.
- kmarm – Amplitud del armónico más bajo.
- kr – Factor de múltiplo en la amplitud de la serie de coeficientes.
- ifn – Número de tabla que contiene una onda sinusoidal.

Este *opcode* ha sido utilizado en la síntesis de sonidos sonoros, preferente a **buzz**, por hacer que no todos los armónicos tengan la misma amplitud.

14. **linen**
Este *opcode* es un modificador de señal: permite aplicar una envolvente de tipo ASR a nuestra señal. Su sintaxis es la que sigue:

kenv **linen** xamp, itataq, idur, itrel

siendo:

- kenv – Envolvente ya creada.
- xamp – Amplitud máxima de la envolvente.
- itataq – Tiempo de ataque.
- idur – Duración total de la envolvente.
- itrel – Tiempo de relajación.

Este *opcode* ha sido ampliamente utilizado para crear envolventes a las señales sintetizadas desde cero.

15. **linseg**
Es un *opcode* generador de señal: permite generar una señal a base de tramos de rectas. Su sintaxis es la que sigue:

kr **linseg** ia, idur1, ib[, idur2, ic[, idur3, id...]]

siendo:

- kr – Señal generada.

- ia, ib... – Pareja de valores que definen la amplitud inicial y la final de cada tramo.
- idur1, idur2... – Duración de cada tramo. Este *opcode* ha sido utilizado para crear envolventes de tipo impulsivo en la síntesis de consonantes por medio de envolventes de amplitud.

16. **lpanal**
Es una utilidad de sonido que permite realizar un análisis LPC a un fichero externo de sonido y que luego puede ser recogido por los operadores lp de CSound. Para más información consultar el manual.

17. **lpread**
Es un *opcode* modificador de señal que permite abrir un archivo .lp que contiene un análisis LPC previamente realizado con **lpanal** sobre un sonido. Su sintaxis es la que sigue:

krmsr, krmso, kerr, kcps **lpread** ktimpnt, ifilcod

siendo:

- krmsr – Valor RMS del residuo.
- krmso – Valor RMS de la señal original.
- kerr – Señal de error normalizada.
- kcps – Altura calculada (Hz).

Este *opcode* ha sido utilizado en la técnica de síntesis mediante LPC para abrir los archivos con los coeficientes del filtro LPC.

18. **lpreson**
Es un *opcode* modificador de señal que permite filtrar una señal con un filtro LPC previamente obtenido por **lpread**. Su sintaxis es la que sigue:

afilt **lpreson** asen

siendo:

- afilt – Señal filtrada con el filtro LPC.
- asen – Señal original.

Este *opcode* ha sido utilizado para realizar el filtrado propio de la LPC a las señales de partida.

19. **multitap**
    Es un *opcode* modificador de señal que permite implementar un línea de retardo múltiple. Su sintaxis es la que sigue:

    ar **multitap** asig, itime1, igain1, itime2, igain2 ...

    siendo:

    - ar – Señal con los retardos.
    - itime1, itime2... – Tiempos de retardo de cada línea.
    - igain1, igain2... – Ganancias de retardo de cada línea.

    Utilizado en la creación de líneas de retardo múltiple en la implementación del algoritmo de reverberación de Stautner y Puckette.

20. **oscil**
    Es un *opcode* generador de señal que permite crear sonidos en base a la repetición de una cierta tabla de onda. Su sintaxis es la que sigue:

    asen **oscil** iamp, ifreq, ifn

    siendo:

    - asen – Señal sintetizada.
    - iamp – Define la amplitud de la señal a sintetizar.
    - ifreq – Frecuencia de la señal sintetizada.
    - ifn – Tabla que contiene un ciclo de la señal a crear.

    Este *opcode* es bastante útil. Una de nuestras aplicaciones ha sido la de crear una sencilla onda sinusoidal de una amplitud determinada para luego, con el *opcode* **balance**, realizar una igualación del valor RMS de una señal procesada con nuestra señal simple.

21. **oscili**
    Es un *opcode* generador de señal que permite crear sonidos en base a la repetición de una cierta tabla de ondas. Es igual que **oscil** pero con una interpolación entre muestras. Su sintaxis es la que sigue:

    asen **oscili** iamp, ifreq, ifn

    siendo:

- asen – Señal sintetizada.
- iamp – Define la amplitud de la señal a sintetizar.
- ifreq – Frecuencia de la señal sintetizada.
- ifn – Tabla que contiene un ciclo de la señal a crear.

Utilizado para crear ondas a partir de tablas pero con mayor precisión que con **oscil**.

22. **pvadd**
Es un *opcode* generador de señal utilizado para realizar un proceso de síntesis aditiva usando osciladores con interpolación. Su sintaxis es la que sigue:

ar **pvadd** ktimpnt, kfmod, ifile, ifn, ibins

siendo:

- ktimpnt – Instante en el fichero a utilizar.
- kfmod – Factor de transposición de la frecuencia.
- ifile – Archivo con los resultados del análisis del fichero de audio.
- ifn – Número de tabla con la onda del oscilador.
- ibins – Número de pistas usadas en la resíntesis.

Utilizado en el apartado de síntesis de consonantes por medio del vocoder en fase.

23. **pvanal**
Es una utilidad de sonido que permite realizar el análisis de Fourier a una señal para luego almacenarlo en un fichero y poder ser utilizado por utilidades como **pvoc** o similares. Para más información consultar el manual.

24. **pvoc**
Es un *opcode* generador de señal que se emplea para implementar un vocoder de fase por medio de la suma de diferentes ondas sinusoidales. Su sintaxis es la que sigue:

ar **pvoc** ktimpnt, kfmod, ifilcod

siendo:

- ar – Señal sintetizada.
- ktimpnt – Instante del fichero a utilizar.
- kfmod – Factor de transposición de altura.
- ifilcod – Fichero en el que se encuentra el análisis del sonido.

Este *opcode* ha sido utilizado en el apartado de síntesis de consonantes por medio del vocoder en fase.

25. **randi**
Es un *opcode* generador de señal que permite crear una serie de números aleatorios entre unos límites fijados. Produce una interpolación lineal entre dos valores obtenidos. Su sintaxis es la que sigue:

ar **randi** xamp, xfreq

siendo:

- ar – Señal de números aleatorios.
- xamp – Límites entre los que generar números, [-xamp, +xamp].
- xfreq – Frecuencia de generación de muestras.

Utilizado para crear ruido a la hora de modelar fonemas sordos en la síntesis sustractiva.

26. **reson**
Es un *opcode* que permite modificar la señal filtrándola paso-banda (filtro todo polos). Su sintaxis es la que sigue:

ar **reson** asig, kfc, kbw, inorm

siendo:

- ar – Señal filtrada.
- asig – Señal a filtrar.
- kfc – Frecuencia central del filtro.
- kbw – Ancho de banda del filtro.
- inorm – Factor de normalización de la amplitud de la señal.

Este *opcode* ha sido utilizado para crear el filtrado que genera los formantes en el sintetizador sustractivo.

27. **soundin**
    Es un *opcode* de entrada de señal: permite introducir ficheros de audio externos para utilizarlos en los instrumentos como cualquier otra señal. Su sintaxis es la que sigue:

    ar **soundin** ifilcod

    siendo:

    - ar – Señal introducida en CSound.
    - ifilcod – Fichero de audio externo.

    Este *opcode* ha sido utilizado en la síntesis por samplers para conseguir introducir dentro de la orquesta las diferentes muestras de sonido necesarias.

28. **resonz**
    Es un *opcode* que permite modificar la señal filtrándola paso-banda (filtro con ceros y polos). Su sintaxis es la que sigue:

    ar **resonz** asig, kfc, kbw

    siendo:

    - ar – Señal filtrada.
    - asig – Señal a filtrar.
    - kfc – Frecuencia central del filtro.
    - kbw – Ancho de banda del filtro.

    Este *opcode*, como **reson**, también ha sido utilizado para filtrar paso-banda la señal de excitación y crear así los formantes necesarios.

29. **tone**
    Es un *opcode* modificador de señal que permite filtrar una señal paso-bajo mediante un filtro todo polos. Su sintaxis es la que sigue:

    ar **tone** asig, kfc

    siendo:

    - ar – Señal filtrada.
    - asig – Señal a filtrar.

- kfc – Frecuencia de corte.

Este *opcode* ha sido utilizado en la unidad de reverberación para eliminar componentes de altas frecuencias y así dar un resultado más real al efecto.

30. **voice**
Es un *opcode* generador de señal que permite emular voz humana. Su sintaxis es la que sigue:

ar **voice** kamp, kfreq, kphoneme, kform, kvibf, kvamp, ifn, ivfn

siendo:

- ar – Señal creada.
- kamp – Amplitud de pico de la señal.
- kfreq – Altura de la señal.
- kphoneme – Un número del 0 al 16 que representa diferentes fonemas (ver manual original).
- kform – Ganancia del fonema (recomendado entre 0 y 1.2).
- kvibf – Frecuencia del vibrato en hertzios (recomendado entre 0 y 12).
- kvamp – Amplitud del vibrato.
- ifn – Tabla de la onda portadora.
- ivfn – Tabla de onda del vibrato.

Este *opcode* ha sido utilizado en las pruebas de síntesis de consonantes por medios de las rutinas propias de CSound.

31. **vosim**
Es un *opcode* generador de señal: crea señal de voz por medio de pulsos glotales con formantes. Su sintaxis es la que sigue:

ar **vosim** kamp, kFund, kForm, kDecay, kPulseCount, kPulseFactor, ifn

siendo:

- ar – Señal sintetizada.
- kamp – Amplitud de pico de la señal.

- kFund – Frecuencia fundamental de la señal creada.
- kForm – Frecuencia del formante.
- kDecay – Factor de decaimiento entre pulsos. La amplitud de un pulso es la amplitud del anterior multiplicada por este factor.
- kPulseCount – Número de pulsos en cada ráfaga.
- kPulseFactor – Factor que multiplica la anchura del pulso: si es mayor que 1, se incrementa la anchura; si es menor que 1, disminuye la anchura. Esto modificará la altura del formante, la cual vendrá dada por la fórmula $KForm = KPulseFactor^{KPulseCount}$.
- ifn – Tabla de onda con medio ciclo de onda sinusoidal.

Este *opcode* ha sido utilizado en síntesis sustractiva para intentar obtener mejores resultado que con una simple señal de pulsos periódica.

## A.2. Funciones de generación de tablas

Aquí vamos a describir las funciones de generación de tablas de onda utilizadas durante el desarrollo del presente proyecto.

- **GEN** 1
  Función de generación de tabla que permite incluir sonido de un fichero como muestras de una tabla de onda para luego poder ser utilizada. Su sintaxis es la que sigue:

  **f** num_tabla tiempo tamaño *1* filcod skiptime format channel

  siendo:

  - filcod – Indica el nombre del fichero fuente.
  - skiptime – Cantidad de tiempo, en segundos, que la lectura desecha contando desde el principio del fichero.
  - format – Formato del fichero. Si es 0, se asume el formato que marque la cabecera del mismo.
  - channel – Número de canales a leer. Si es 0, se leen todos.

  Esta función **GEN** ha sido utilizada principalmente en el *opcode* **voice**, el cual necesita una serie de ficheros externos para poder funcionar.

- **GEN** 9
  Función de generación de tablas de onda que permite crear formas de onda mediante una suma ponderada de armónicos sinusoidales, especificados por su amplitud y su fase. Su sintaxis es la que sigue:

  **f** num_tabla tiempo tamaño *9* pna stra phsa pnb strb phsb...

  siendo:

  - pna, pnb... – Número de parcial.
  - stra, strb... – Amplitud relativa de cada parcial.
  - phsa, phsb... – Fase inicial de cada parcial.

  La función de onda que se puede generar con esta función ha sido utilizada principalmente para el *opcode* **vosim**.

- **GEN** 10
  Función de generación de tablas de onda que permite crear formas de onda mediante una suma ponderada de armónicos sinusoidales. Su sintaxis es la que sigue:

  **f** num_tabla tiempo tamaño *10* arm1 arm2 arm3...

  siendo:

  - arm1, arm2, arm3... – Amplitud del armónico.

  La tabla generada por esta función **GEN** ha sido bastante utilizada: para utilizarla en la síntesis de sonidos vocálicos y para crear señales simples con las que poder hacer la igualación de valores RMS.

- **GEN** 17
  Función de generación de tabla de ondas que permite crear una función escalonada por medio de pares xy de datos. Su sintaxis es la que sigue:

  **f** num_tabla tiempo tamaño *17* x1 a x2 b x3 c...

  siendo:

  - x1, x2, x3... – Valor de abscisa comenzando por 0 y en orden ascendente.

- a, b, c... – Valor de ordenada correspondiente a la abscisa anterior y constante hasta la siguiente abscisa.

Con esta función **GEN** se intentó llevar a cabo una forma de síntesis de fonemas de tipo oclusivo.

- **GEN** 19
Función de generación de tablas de onda que permite crear ondas compuestas por medio de suma de ondas sinusoidales. Su sintaxis es la que sigue:

**f** num_tabla tiempo tamaño *19* pna stra pha dcoa pnb strb phb dcob...

siendo:

- pna, pnb... – Número de parcial. Puede estar en cualquier orden.
- stra, strb... – Amplitud del parcial.
- pha, phb... – Fase inicial del parcial.
- dcoa, dcob... – Nivel de offset del parcial.

Esta función **GEN** ha sido utilizada en la creación de las tablas necesarias para la síntesis mediante las rutinas FOF de CSound.

- **GEN** 21
Función de generación de tablas de onda que genera formas de onda con distribuciones aleatorias de distinto tipo. Su sintaxis es la que sigue:

**f** num_tabla tiempo tamaño *21* tipo lvl arg1 arg2

siendo:

- tipo – Tipo de distribución a emplear. Puede ser:
  1. Uniforme
  2. Lineal
  3. Triangular
  4. Exponencial
  5. Biexponencial
  6. Gaussiana
  7. Cauchy
  8. Cauchy Positiva

9. Beta
10. Weibull
11. Poison

- lvl – Parámetro de alguna distribución especial.
- arg1 – Parámetro de alguna distribución especial.
- arg2 – Parámetro de alguna distribución especial.

Esta función **GEN** fue utilizada para crear distribuciones aleatorias de tipo gaussiano para después crear señal de ruido (mediante cualquier variante del *opcode* **oscil**) y posteriormente filtrar esa señal con un banco de filtros para realizar los formantes.

## A.3. Tabla de equivalencias

- Equivalencia de amplitudes

| Nota | $DO_3$ | $DO\sharp_3$ | $RE_3$ | $RE\sharp_3$ | $MI_3$ | $FA_3$ | $FA\sharp_3$ | $SOL_3$ |
|---|---|---|---|---|---|---|---|---|
| Hz | 261.63 | 277.2 | 293.7 | 311.1 | 329.6 | 349.2 | 370.0 | 391.5 |
| Cpspch | 8.00 | 8.01 | 8.02 | 8.03 | 8.04 | 8.05 | 8.06 | 8.07 |
| MIDI | 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 |

| Nota | $SOL\sharp_3$ | $LA_3$ | $LA\sharp_3$ | $SI_3$ | $DO_4$ |
|---|---|---|---|---|---|
| Hz | 415.3 | 440.0 | 466.2 | 493.9 | 523.3 |
| Cpspch | 8.08 | 8.09 | 8.10 | 8.11 | 9.00 |
| MIDI | 68 | 69 | 70 | 71 | 72 |

- Equivalencia de alturas

| Decibelios (dB) | 42 | 48 | 54 | 60 | 66 | 72 | 78 |
|---|---|---|---|---|---|---|---|
| Lineal (PCM) | 125.9 | 251.2 | 501.2 | 1000 | 1995.3 | 3981.1 | 7943.3 |

| Decibelios (dB) | 84 | 90 | 96 |
|---|---|---|---|
| Lineal (PCM) | 15848.9 | 31622.9 | 63095.7 |

# Apéndice B

# Sobre formantes en los fonemas

En este anexo se incluyen los valores de los diferentes formantes de cada fonema tratado en el presente proyecto.

## B.1. Vocales

### B.1.1. Soprano

- **a**

| Número de formante | Frecuencia (Hz) | Amplitud relativa (dB) | Ancho de banda (Hz) |
|---|---|---|---|
| F1 | 800 | 0 | 80 |
| F2 | 1150 | -6 | 90 |
| F3 | 2900 | -32 | 120 |
| F4 | 3900 | -20 | 130 |
| F5 | 4950 | -50 | 140 |

- **e**

| Número de formante | Frecuencia (Hz) | Amplitud relativa (dB) | Ancho de banda (Hz) |
|---|---|---|---|
| F1 | 350 | 0 | 60 |
| F2 | 2000 | -20 | 100 |
| F3 | 2800 | -15 | 120 |
| F4 | 3600 | -40 | 150 |
| F5 | 4950 | -56 | 200 |

- **i**

| Número de formante | Frecuencia (Hz) | Amplitud relativa (dB) | Ancho de banda (Hz) |
|---|---|---|---|
| F1 | 270 | 0 | 60 |
| F2 | 2140 | -12 | 90 |
| F3 | 2950 | -26 | 100 |
| F4 | 3900 | -26 | 120 |
| F5 | 4950 | -40 | 120 |

- **o**

| Número de formante | Frecuencia (Hz) | Amplitud relativa (dB) | Ancho de banda (Hz) |
|---|---|---|---|
| F1 | 450 | 0 | 70 |
| F2 | 800 | -11 | 80 |
| F3 | 2830 | -22 | 100 |
| F4 | 3800 | -22 | 130 |
| F5 | 4950 | -50 | 135 |

- **u**

| Número de formante | Frecuencia (Hz) | Amplitud relativa (dB) | Ancho de banda (Hz) |
|---|---|---|---|
| F1 | 325 | 0 | 50 |
| F2 | 700 | -16 | 60 |
| F3 | 2700 | -35 | 170 |
| F4 | 3800 | -40 | 180 |
| F5 | 4950 | -60 | 200 |

### B.1.2. Contralto

- **a**

| Número de formante | Frecuencia (Hz) | Amplitud relativa (dB) | Ancho de banda (Hz) |
|---|---|---|---|
| F1 | 800 | 0 | 80 |
| F2 | 1150 | -4 | 90 |
| F3 | 2800 | -20 | 120 |
| F4 | 3500 | -36 | 130 |
| F5 | 4950 | -60 | 140 |

- **e**

| Número de formante | Frecuencia (Hz) | Amplitud relativa (dB) | Ancho de banda (Hz) |
|---|---|---|---|
| F1 | 400 | 0 | 60 |
| F2 | 1600 | -24 | 80 |
| F3 | 2700 | -30 | 120 |
| F4 | 3300 | -35 | 150 |
| F5 | 4950 | -60 | 200 |

- **i**

| Número de formante | Frecuencia (Hz) | Amplitud relativa (dB) | Ancho de banda (Hz) |
|---|---|---|---|
| F1 | 350 | 0 | 50 |
| F2 | 1700 | -20 | 100 |
| F3 | 2700 | -30 | 120 |
| F4 | 3700 | -36 | 150 |
| F5 | 4950 | -60 | 200 |

- **o**

| Número de formante | Frecuencia (Hz) | Amplitud relativa (dB) | Ancho de banda (Hz) |
|---|---|---|---|
| F1 | 450 | 0 | 70 |
| F2 | 800 | -9 | 80 |
| F3 | 2830 | -16 | 100 |
| F4 | 3500 | -28 | 130 |
| F5 | 4950 | -55 | 135 |

- **u**

| Número de formante | Frecuencia (Hz) | Amplitud relativa (dB) | Ancho de banda (Hz) |
|---|---|---|---|
| F1 | 325 | 0 | 50 |
| F2 | 700 | -12 | 60 |
| F3 | 2530 | -30 | 170 |
| F4 | 3500 | -40 | 180 |
| F5 | 4950 | -64 | 200 |

### B.1.3. Tenor

- **a**

| Número de formante | Frecuencia (Hz) | Amplitud relativa (dB) | Ancho de banda (Hz) |
|---|---|---|---|
| F1 | 650 | 0 | 80 |
| F2 | 1080 | -6 | 90 |
| F3 | 2650 | -7 | 120 |
| F4 | 2900 | -8 | 130 |
| F5 | 3250 | -22 | 140 |

- **e**

| Número de formante | Frecuencia (Hz) | Amplitud relativa (dB) | Ancho de banda (Hz) |
|---|---|---|---|
| F1 | 400 | 0 | 70 |
| F2 | 1700 | -14 | 80 |
| F3 | 2600 | -12 | 100 |
| F4 | 3200 | -14 | 120 |
| F5 | 3580 | -20 | 120 |

- **i**

| Número de formante | Frecuencia (Hz) | Amplitud relativa (dB) | Ancho de banda (Hz) |
|---|---|---|---|
| F1 | 290 | 0 | 40 |
| F2 | 1870 | -15 | 90 |
| F3 | 2800 | -18 | 100 |
| F4 | 3250 | -20 | 100 |
| F5 | 3540 | -30 | 120 |

- **o**

| Número de formante | Frecuencia (Hz) | Amplitud relativa (dB) | Ancho de banda (Hz) |
|---|---|---|---|
| F1 | 400 | 0 | 40 |
| F2 | 800 | -10 | 80 |
| F3 | 2600 | -12 | 100 |
| F4 | 2800 | -12 | 120 |
| F5 | 3000 | -26 | 120 |

- **u**

| Número de formante | Frecuencia (Hz) | Amplitud relativa (dB) | Ancho de banda (Hz) |
|---|---|---|---|
| F1 | 325 | 0 | 40 |
| F2 | 600 | -20 | 60 |
| F3 | 2700 | -17 | 100 |
| F4 | 2900 | -14 | 120 |
| F5 | 3300 | -26 | 120 |

### B.1.4. Bajo

- **a**

| Número de formante | Frecuencia (Hz) | Amplitud relativa (dB) | Ancho de banda (Hz) |
|---|---|---|---|
| F1 | 600 | 0 | 60 |
| F2 | 1040 | -7 | 70 |
| F3 | 2250 | -9 | 110 |
| F4 | 2450 | -9 | 120 |
| F5 | 2750 | -20 | 130 |

- **e**

| Número de formante | Frecuencia (Hz) | Amplitud relativa (dB) | Ancho de banda (Hz) |
|---|---|---|---|
| F1 | 400 | 0 | 40 |
| F2 | 1620 | -12 | 80 |
| F3 | 2400 | -9 | 100 |
| F4 | 2800 | -12 | 120 |
| F5 | 3100 | -18 | 120 |

- **i**

| Número de formante | Frecuencia (Hz) | Amplitud relativa (dB) | Ancho de banda (Hz) |
|---|---|---|---|
| F1 | 250 | 0 | 60 |
| F2 | 1750 | -30 | 90 |
| F3 | 2600 | -16 | 100 |
| F4 | 3050 | -22 | 120 |
| F5 | 3340 | -18 | 120 |

- **o**

| Número de formante | Frecuencia (Hz) | Amplitud relativa (dB) | Ancho de banda (Hz) |
|---|---|---|---|
| F1 | 400 | 0 | 40 |
| F2 | 750 | -11 | 80 |
| F3 | 2400 | -21 | 100 |
| F4 | 2600 | -20 | 120 |
| F5 | 2900 | -41 | 120 |

- **u**

| Número de formante | Frecuencia (Hz) | Amplitud relativa (dB) | Ancho de banda (Hz) |
|---|---|---|---|
| F1 | 350 | 0 | 40 |
| F2 | 600 | -20 | 80 |
| F3 | 2400 | -32 | 100 |
| F4 | 2675 | -28 | 120 |
| F5 | 2950 | -36 | 120 |

## B.2. Consonantes

En lo relativo a formantes sobre fonemas consonánticos, aquí los valores no son tan fiables como en el caso de las vocales porque, al no ser sonidos, en su mayoría, estacionarios y ser, en general, de corta duración, el análisis frecuencial se hace, al mismo tiempo, dificultoso e impreciso. Por ello se adjuntan en esta memoria una serie de espectrogramas de palabras que contienen las consonantes de las cuales ha sido necesario extraer los formantes para llevar a cabo su síntesis.

Estas palabras surgen de grabaciones que fueron llevadas a cabo por medio de un micrófono dinámico (concretamente, el modelo FAMAPREM MC–454) conectado a la tarjeta básica de sonido del ordenador[1].

Las consonantes son:

- Consonante *m*
  La palabra utilizada en este caso es *mango*, cuyo espectrograma podemos ver en la figura B.1.

[1]También cabe destacar la importancia de ciertas fuentes bibliográficas a la hora de establecer estos formantes, como son [1] y [15].

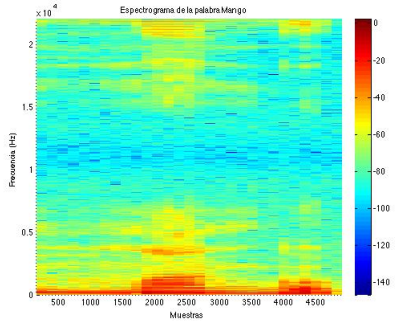


Figura B.1: Palabra *mango* para la extracción de los formantes de la consonante *m*.

Los formantes encontrados en este caso se sitúan a las frecuencias de 170 Hz, 200 Hz, 1200 Hz y 3500 Hz, en una primera aproximación.

- Consonante *n*
  La palabra utilizada en este caso es *nombre*, cuyo espectrograma podemos ver en la figura B.2.

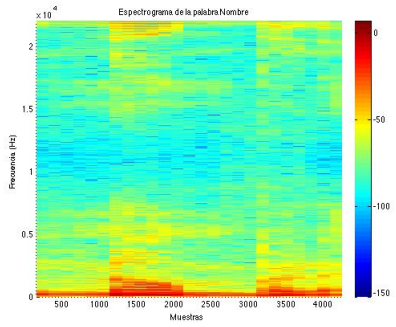


Figura B.2: Palabra *nombre* para la extracción de los formantes de la consonante *n*.

Para este caso encontramos los formantes situados en las frecuencias de 170 Hz, 1400 Hz y 2500 Hz, aproximadamente.

- Consonante *s*
  La palabra utilizada en este caso es *según*, cuyo espectrograma podemos ver en la figura B.3.

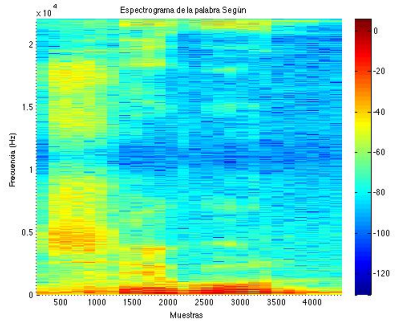


Figura B.3: Palabra *según* para la extracción de los formantes de la consonante *s*.

El valor de los formantes encontrados en este caso es 400 Hz, 1600 Hz, 2600 Hz y 3400 Hz.

- Consonante *b*
  La palabra utilizada en este caso es *bueno*, cuyo espectrograma podemos ver en la figura B.4.

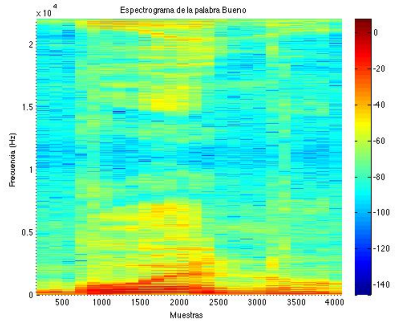


Figura B.4: Palabra *bueno* para la extracción de los formantes de la consonante *b*.

Los formantes, al ser su extracción a partir del espectrograma bastante dificultosa, se utilizan los encontrados en **??**, los cuales son 431 Hz, 904 Hz, 2584 Hz y 3790 Hz.

- Consonante *d*
  La palabra utilizada en este caso es *dado*, cuyo espectrograma podemos

ver en la figura B.5.

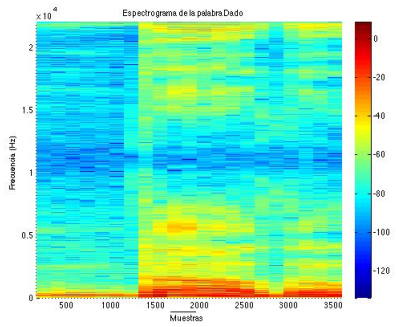


Figura B.5: Palabra *dado* para la extracción de los formantes de la consonante *d*.

Los formantes obtenidos en este caso son 550 Hz, 1600 Hz, 2800 Hz y 3700 Hz.

- Consonante *t*
  La palabra utilizada en este caso es *tocar*, cuyo espectrograma podemos ver en la figura B.6.

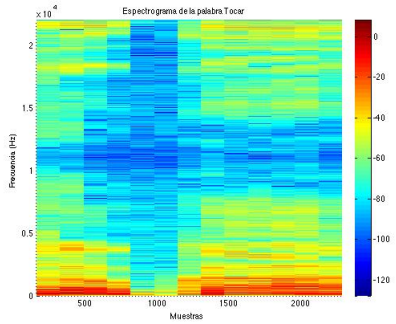


Figura B.6: Palabra *tocar* para la extracción de los formantes de la consonante *t*.

En este caso, los formantes encontrados son 450 Hz, 1900 Hz, 3300 Hz y 4450 Hz.

- Consonante *z*
  La palabra utilizada en este caso es *alza*, cuyo espectrograma podemos ver en la figura B.7.

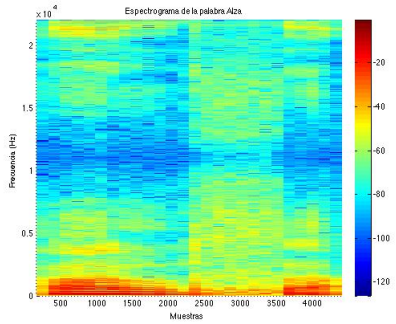


Figura B.7: Palabra *alza* para la extracción de los formantes de la consonante *z*.

Los formantes obtenidos son 500 Hz, 1600 Hz, 2400 Hz, 3200 Hz, y 4000 Hz.

- Consonante *r*
  La palabra utilizada en este caso es *casa*, cuyo espectrograma podemos ver en la figura B.8.

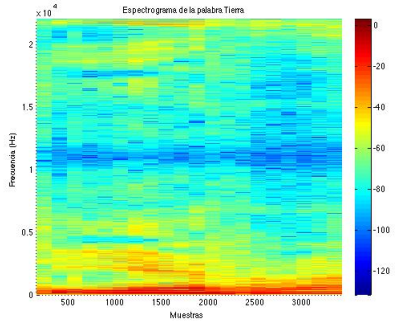


Figura B.8: Palabra *Tierra* para la extracción de los formantes de la consonante *r*.

Los formantes encontrados en este caso son 500 Hz, 1150 Hz, 2600 Hz, 3300 Hz y 3800 Hz.

- Consonante *p*
  La palabra utilizada en este caso es *Paco*, cuyo espectrograma podemos ver en la figura B.9.

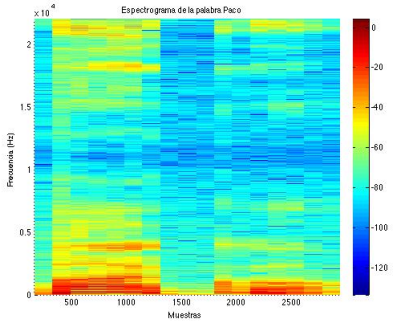


Figura B.9: Palabra *Paco* para la extracción de los formantes de la consonante *p*.

Los formantes encontrados para esta consonante son 800 Hz, 1100 Hz, 2650 Hz y 3650 Hz.

- Consonante *j*
  La palabra utilizada en este caso es *gente*, cuyo espectrograma podemos ver en la figura B.10.

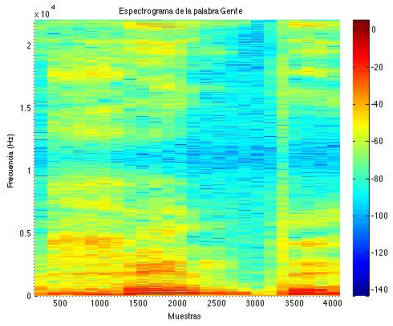


Figura B.10: Palabra *gente* para la extracción de los formantes de la consonante *g*.

Los formantes obtenidos son 790 Hz, 1400 Hz, 2800 Hz y 3800 Hz.

- Consonante *l*
  La palabra utilizada en este caso es *Lola*, cuyo espectrograma podemos ver en la figura B.11.

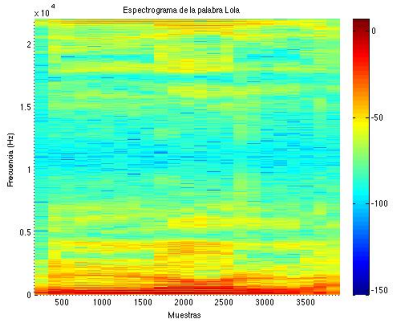


Figura B.11: Palabra *Lola* para la extracción de los formantes de la consonante *l*.

Los formantes encontrados en este caso son 517 Hz, 1723 Hz, 2756 Hz y 3747 Hz.

# Apéndice C

# Sobre numeración de instrumentos en el sintetizador

En este anexo se incluye la numeración de los diferentes instrumentos en los dos archivos de orquesta que han tenido lugar en el desarrollo del presente proyecto.

## C.1. Sintetizador de sonidos vocálicos

A continuación se muestra la numeración de los instrumentos del sintetizador de vocales[1] que se ha utilizado para el desarrollo del proyecto:

| Instrumento Excitador | | Instrumento Filtro | |
|---|---|---|---|
| Voz | Instrumento | Vocal | Instrumento |
| Soprano | 1 | a | x1 |
| Contralto | 2 | e | x2 |
| Tenor | 3 | i | x3 |
| Bajo | 4 | o | x4 |
| - | | u | x5 |
| Instrumento Reverberación | | | 99 |

Como último apunte, destacar que la $x$ que aparece en los instrumentos de filtrado se sustituye por el número de la voz que la produce, es decir, por el número del instrumento excitador que crea la señal a fitrar.

[1]Recordemos que este sintetizador funciona con síntesis sustractiva, por lo que tendremos un instrumento de excitación y otro de filtrado.

## C.2. Sintetizador de sonidos vocálicos y consonánticos

Para la numeración de los instrumentos en este sintetizador distinguimos dos situaciones para simplificar la clasificación.

### C.2.1. Vocales

Los sonidos vocálicos tienen la siguiente distribución:

| Instrumento Excitador | | Instrumento Filtro | |
|---|---|---|---|
| Voz | Instrumento | Vocal | Instrumento |
| Soprano | 1001 | a | x01 |
| Contralto | 1002 | e | x05 |
| Tenor | 1003 | i | x09 |
| Bajo | 1004 | o | x15 |
| - | | u | x21 |

Como en el caso del sintetizador de vocales, la $x$ de los instrumentos de excitación se sustituye por el número del excitador. En este caso, realmente se pone el último de los números del instrumento de excitación, que coincide con el instrumento excitador del sintetizador de vocales.

### C.2.2. Consonantes

La numeración de las consonantes es la que sigue a continuación:

| Letra | Fonema | Instrumento | Letra | Fonema | Instrumento |
|---|---|---|---|---|---|
| b | b | 02 | q | k | 11 |
| d | d̠ | 04 | r | ɾ | 18 |
| f | f | 06 | s | s | 19 |
| g | g | 07 | t | t | 20 |
| j | j | 10 | v | b | 22 |
| k, c | k | 11 | w | u | x21 |
| l | l | 12 | y, ll | ʎ | x09 |
| m | m | 13 | z, c | z | 26 |
| n | n | 14 | rr | r | 27 |
| p | p | 16 | ñ | ɲ | 28 |

Respecto a esta tabla cabe destacar una serie de puntos:

- Ciertos fonemas se han aproximado a otros a pesar de que no es así, como es el caso de modelar la consonante $v$ como una $b$.

- Algunas consonantes (la *w*, la *y* y la *ll*) se modelan como vocales, por lo que se necesita el instrumento excitador de la voz que crea la llamada y el filtro.
- Por último, algunos fonemas (los sonidos de la $\tilde{n}$ y la *rr*) tienen una numeración más allá de la *z* por no encontrarse como tales en el código ASCII.

### C.2.3. Reverberación

El instrumento de reverberación, en este caso, ha sido codificado con el número 9999 ya que estamos tratando siempre de que sea el instrumento con el mayor número de todos.

# Apéndice D

# El estándar MIDI

Con este anexo se pretende añadir información a la ya proporcionada en la introducción del estándar MIDI comentada en la presente memoria.

## D.1. Tipos de mensaje en el estándar MIDI

A continuación se añade cada uno de los casos posibles que pueden darse de mensajes en el estándar MIDI, tanto de canal como de sistema, así como de los parámetros que son necesarios.

1. Mensajes de canal de voz

| | Byte de estado | | Byte de datos 1 | Byte de datos 2 |
|---|---|---|---|---|
| | Binario | Hexadecimal | Valor | Valor |
| Nota Off | 1000cccc | 8c | Nota | Velocidad |
| Nota On | 1001cccc | 9c | Nota | Velocidad |
| Postpulsación polifónica | 1010cccc | Ac | Nota | Presión |
| Cambio de controlador | 1011cccc | Bc | Número de controlador | Valor numérico |
| Cambio de programa | 1100cccc | Cc | Número de programa | |
| Postpulsación de canal | 1101cccc | Dc | Presión | |
| Variación de altura | 1110cccc | Ec | Valor menos significativo | Valor más significativo |

2. Mensajes de canal de modo

| Mensaje | Hexadecimal | Binario | Valor 1 | Valor 2 |
|---|---|---|---|---|
| Restaurar controladores | Bc | 1011cccc | 79 | 0 |
| Control local on/off | Bc | 1011cccc | 7A | 7F=On 0=Off |
| Desactivar todas las notas | Bc | 1011cccc | 7B | 0 |
| Desactivar modo omni | Bc | 1011cccc | 7C | 0 |
| Activar modo omni | Bc | 1011cccc | 7D | 0 |
| Modo mono on | Bc | 1011cccc | 7E | 0 - 10 |
| Modo mono off | Bc | 1011cccc | 7F | 0 |

3. Mensajes de sistema comunes

| Mensaje | Hexadecimal | Binario | Valor 1 | Valor 2 |
|---|---|---|---|---|
| Trama temporal | F1 | 11110001 | | |
| Posición en canción | F2 | 11110010 | LSB | MSB |
| Selección de canción | F3 | 11110011 | 0–127 | – |
| Sin definir | F4 | 11110100 | | |
| Sin definir | F5 | 11110101 | | |
| Petición de afinación | F6 | 11110110 | | |

4. Mensajes de sistema de tiempo real

| Mensaje | Hexadecimal | Binario |
|---|---|---|
| Reloj MIDI | F8 | 11111000 |
| Sin definir | F9 | 11111001 |
| Inicio | FA | 11111010 |
| Continuación | FB | 11111011 |
| Parada | FC | 11111100 |
| Sin definir | FD | 11111101 |
| Espera activa | FE | 11111110 |
| Reset | FF | 11111111 |

5. Mensajes de sistema exclusivos
   Son mensajes que sirven para que activar ciertas características especiales de algunos dispositivos MIDI sin tener que modificar el estándar. Estos mensajes tienen siempre esta forma:

   **F0** <id_fabricante> <id_dispositivo> <id_modelo> ... **F7**

   En esta trama, el primer byte y el último son siempre los especificados en negrita (para así delimitar la trama) y los tres primero de cada trama hacen siempre referencia a estos tres datos:

- Mensaje dirigido a dispositivo según fabricante.
- Mensaje dirigido según el dispositivo MIDI al que nos queramos dirigir.
- Mensaje dirigido según el modelo de dispositivo.

## D.2. Ficheros MIDI

Finalmente cabe comentar un pequeño aspecto más acerca del estándar MIDI que es el cómo se guardan esas secuencias en ficheros.

Ciertamente, todo los mensajes explicados siguen siendo válidos (de hecho, eso el estándar de comunicación entre dispositivos MIDI), pero este tipo de intercambio de información está pensado para tiempo real, es decir, que alguien esté tocando un instrumento MIDI y que, cuando lo vaya haciendo, se vayan mandando mensajes cada cuanto se necesite. Se podría dar el caso de querer guardar en un fichero todas nuestras secuencias y, como los mensajes que se emiten no se tiene concepción del tiempo de duración de cada nota, se hace indispensable el incluir un mecanismo para conseguir codificar esa información.

En definitiva, en un fichero MIDI podremos encontrar estos tres bloques de información:

1. Mensajes MIDI
   Son los comandos que hemos explicados antes que harán que una nota suene, deje de sonar, cambie de timbre... Al ya estar explicados no haremos mayor referencia a los mismos.

2. Tiempos delta
   Es el método que se utilizará para conseguir codificar cuánto ha de durar cada nota y cada silencio.

   Los tiempos delta son unos mensajes que se adjuntan al principio de cada mensaje MIDI y que hacen referencia al número de pulsos, refiriéndonos a pulsos de resolución temporal, que ha de esperar el sintetizador para emitir (entendiendo por emitir el hecho de ejecutar) el mensaje al que precede ese tiempo delta.

Estos tiempos delta se codifican con una longitud variable que puede tener de 1 a 4 bytes. Para que el sintetizador pueda saber cuántos bytes tiene el tiempo delta se siguen estas simples reglas:

- El último byte siempre comienza por 0, por lo que ya sabemos qué byte es el último de todos.
- El resto de bytes (hayan 2, 3 ó 4) comienzan siempre por 1.
- Sea cual sea el byte, únicamente cuentan como valor de tiempo delta los 7 bits restantes que no tienen ningún valor impuesto de serie.

Por último cabe destacar que si un tiempo delta es 0 quiere decir que ese mensaje es simultáneo al anterior.

3. Metaeventos
   Los campos con metaeventos en el estándar MIDI son campos en los que se envía información complementaria que el sintetizador no necesita. En estos campos podemos introducir información como la métrica, el nombre de la pista, la letra de la canción, la tonalidad, el tempo...

   La estructura que permite enviar metaeventos es la que sigue:

   **FF** <id_metadato> <longitud_datos> <datos>

   Finalmente cabe destacar algunos de estos metaeventos:

| Identificador (Hexadecimal) | Longitud | Datos | Descripción |
|---|---|---|---|
| 01 | N | Texto | Texto genérico (N=longitud en bytes) |
| 02 | N | Texto | Copyright (N=longitud en bytes) |
| 03 | N | Texto | Nombre de la pista (N=longitud en bytes) |
| 04 | N | Texto | Nombre del instrumento de la pista (N=longitud en bytes) |
| 05 | N | Texto | Letra de la canción (N=longitud en bytes) |
| 2F | 00 | - | Marca de fin de pista |
| 51 | 03 | t1 t2 t3 | Tempo (microsegundos por tiempo) |
| 58 | 04 | nn dd cc bb | Métrica (nn=numerador; dd=exponente de la potencia de 2 del denominador); cc=mensajes de reloj por tiempo; bb=fusas por tiempo) |
| 59 | 02 | sf mm | Tonalidad. sf=número de alteraciones (positivo si sostenidos, negativo si bemoles); mm=modo (00 es mayor, 01 es menor) |

# Apéndice E

# Reverberación Feedback Delay Network de Stautner y Puckette

En 1982 J. Stautner y M. Puckette crearon un algoritmo de reverberación, actualmente conocido como Feedback Delay Network (FDN), basado en líneas de retardo retroalimentadas con la señal de entrada a través de una matriz. Esto, que a priori puede parecer bastante complejo, se resume en que nosotros, en lugar de implementar un efecto de reverberación con una única señal, realizamos copias del sonido original y, para cada una de ellas, creamos una unidad de reverberación simple, que no es del todo independiente de las demás, y el resultado lo volvemos a unir. En definitiva, la representación en diagrama de bloques puede verse en la figura E.1.

La idea, desde un punto de vista más matemático, es que en lugar una única línea de retardo de m muestras tengamos múltiples líneas de diferentes longitud y con ganancias también diferentes (representado por una matriz de ganancias de realimentación) para simular diferentes reflexiones de onda. La ecuación en la que se puede ver esto es la que sigue:

$$y(n) = x(n-m) + g \cdot y(n-m) \tag{E.1}$$

La matriz que se utilizó en el sintetizador original de coros que se comenta en el presente proyecto, y que además fue la que Stautner y Puckette propusieron, es la que sigue:

$$G = \frac{g}{\sqrt{2}} \cdot \begin{pmatrix} 0 & 1 & 1 & 0 \\ -1 & 0 & 0 & -1 \\ 1 & 0 & 0 & -1 \\ 0 & 1 & -1 & 0 \end{pmatrix} \text{ Siendo } g \text{ un valor de ganancia.} \tag{E.2}$$

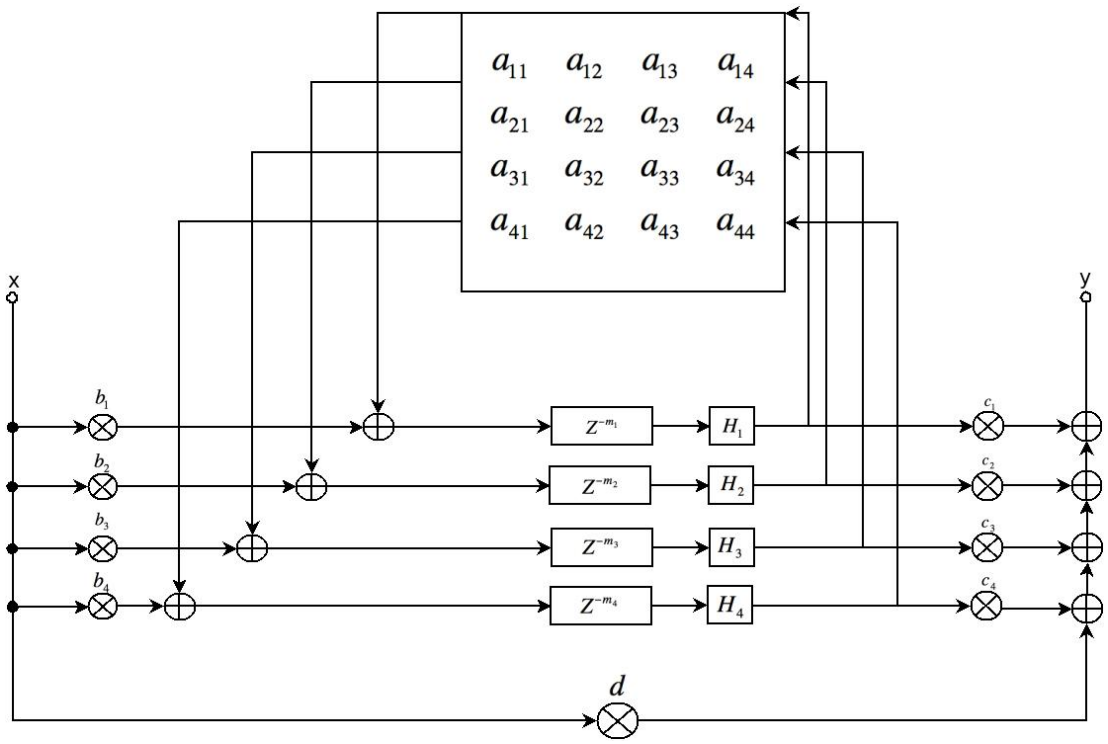


Figura E.1: Diagrama del algoritmo FDN

# Apéndice F

# Método de Codificación Lineal Predictiva (LPC)

La codificación lineal predictiva, también llamada análisis de predicción lineal, es una técnica muy utilizada en el procesado de voz porque permite parametrizar una señal con un número relativamente pequeño de patrones para luego poder ser reconstruida.

Lo que básicamente se intenta con este método es crear un filtro (de tipo IIR todo-polos) que vaya variando en el tiempo para que así, al introducir nosotros una señal y ser filtrada, consigamos un resultado lo más parecido a la señal sobre la que se diseñó el filtro (obviamente, la calidad final dependerá tanto de la precisión de realización del filtro como de la señal de entrada). Por tanto, lo que estamos modelando es una función de transferencia: en lugar de caracterizar de cualquier forma la propia voz (típicamente, mediante su espectro), nosotros lo que hacemos es crear un sistema que, al ser excitado con una señal (normalmente señales de banda ancha ruidosas o periódicas para imitar los sonidos sordos y sonoros respectivamente), devuelva la voz que nosotros queremos. La representación de esto se puede ver en la figura F.1.

El razonamiento que nos lleva a la forma de calcular los coeficientes del filtro es el que sigue:

1. Sea la señal $s(n)$, establecemos una muestra como una predicción lineal de P muestras anteriores:

$$\widetilde{s}\,(n) = \sum_{i=1}^{P} a_i \cdot s(n-i) \tag{F.1}$$


Figura F.1: Representación del esquema de modelado de voz según el método LPC

2. Calculamos el error de predicción y lo transformamos en cuadrático para obtener siempre un error positivo:

   - Error lineal:

$$e(n) = s(n) - \widetilde{s}\,(n) = s(n) - \sum_{i=1}^{P} a_i \cdot s(n-i) \tag{F.2}$$

   - Error cuadrático:

$$e^2(n) = [s(n) - \widetilde{s}\,(n)]^2 = [s(n) - \sum_{i=1}^{P} a_i \cdot s(n-i)]^2 \tag{F.3}$$

3. Error cuadrático de N muestras:

$$E = \sum_{n=0}^{N-1} e^2(n) = \sum_{n=0}^{N-1} [s(n) - \sum_{k=1}^{P} a_k \cdot s(n-k)]^2 \tag{F.4}$$

4. Minimizamos el error derivando por cada uno de los coeficientes $a_k$ que compondrón el filtro e igualando la expresión a cero:

   - Derivamos e igualamos a cero:

$$\begin{aligned} \frac{\partial E}{\partial a_j} &= \frac{\partial}{\partial a_j} \sum_{n=0}^{N-1} \left[ s(n) - \sum_{k=1}^{P} a_k \cdot s(n-k) \right]^2 = \\ &= \sum_{n=0}^{N-1} 2 \cdot \left[ s(n) - \sum_{k=1}^{P} a_k \cdot s(n-k) \right] \cdot [-s(n-j)] = \\ &= -2 \sum_{n=0}^{N-1} \left[ s(n) - \sum_{k=1}^{P} a_k \cdot s(n-k) \right] \cdot s(n-j) = 0 \end{aligned} \tag{F.5}$$

- Realizamos el despeje:

5. Realizamos la igualación $\phi_{i,k} = \sum_{n=0}^{N-1} s(n-i) \cdot s(n-k)$, quedando la expresión así:

$$\phi_{i,0} = \sum_{i=1}^{P} \phi_{i,k} \cdot a_k \tag{F.6}$$

Como podemos observar, esta ecuación provoca un sistema lineal de $P$ ecuaciones con $P$ incógnitas como el que sigue:

$$\begin{bmatrix} \phi_{1,0} \\ \phi_{2,0} \\ \phi_{3,0} \\ \cdots \\ \phi_{P,0} \end{bmatrix} = \begin{bmatrix} \phi_{1,1} & \phi_{1,2} & \phi_{1,3} & \cdots & \phi_{1,P} \\ \phi_{2,1} & \phi_{2,2} & \phi_{2,3} & \cdots & \phi_{2,P} \\ \phi_{3,1} & \phi_{3,2} & \phi_{3,3} & \cdots & \phi_{3,P} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \phi_{P,1} & \phi_{P,2} & \phi_{P,3} & \cdots & \phi_{P,P} \end{bmatrix} \cdot \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ \cdots \\ a_P \end{bmatrix} \tag{F.7}$$

Por tanto, lo único que debemos hacer es resolver este sistema para obtener los coeficientes $a_k$ del filtro. Las diferentes formas que se proponen para llevar a cabo esta tarea son:

- Métodos directos
  Este nombre hace referencia a los métodos clásicos de resolución de sistemas de ecuaciones como Gauss o Gauss-Jordan. Aunque dan la solución exacta al sistema, son muy costosos desde un punto de vista computacional (complejidad cúbica) por lo que, en una aplicación como esta que necesita ser bastante rápida, queda descartado este método.

- Métodos iterativos
  Con este apartado nos referimos a los métodos de resolución de ecuaciones de tipo Jacobi o Gauss-Seidel. Son bastante más rápidos que los anteriores pero tienen dos factores en su contra: no dan la solución exacta al problema y su éxito en la resolución del sistema depende de un valor inicial, pudiendo dar un resultado correcto, incorrecto o incluso no acabar nunca.

- Método de la autocorrelación
  Este método es el que normalmente se implementa para realizar este tipo de procesado. Se basa en que las expresiones de la solución final que no son las incógnitas (las valores que hemos representado con $\phi$) son, básicamente, expresiones de autocorrelación de la señal a procesar, quedando entonces la ecuación genérica así:

$$R_s(|j|) = \sum_{k=1}^{P} a_k \cdot R_s(|j-k|) \tag{F.8}$$

Por la propiedad de la función de autocorrelación de ser una función par, la matriz que genera la expresión es una matriz de tipo Toeplitz, la cual es un caso particular de matriz con gran cantidad de simetría en ella como podemos ver a continuación:

$$\begin{bmatrix} a & b & c & d \\ d & a & b & c \\ c & d & a & b \\ b & c & d & a \end{bmatrix}$$

Por el caso tan particular de matriz, el sistema que se genera se puede resolver de una forma muy eficiente mediante el llamado algoritmo de Levinson-Durbin, el cual se describe a continuación[1]:

**Algoritmo 13** Algoritmo de Levinson-Durbin

1: $E^{(0)} = r[0]$
2: **Bucle desde** i=1 **hasta** i=P
3: $k_i = \frac{1}{E^{(i-1)}} \left( r[i] - \sum_{j=1}^{i-1} a_j^{(i-1)} \cdot r[i-j] \right)$
4: $a_i^{(i)} = k_i$
5: $a_j^{(i)} = a_j^{(i-1)} - k_i \cdot a_{i-1}^{(i-1)}$
6: $E^{(i)} = (1 - k_i)^2 \cdot E^{(i-1)}$
7: **Fin Bucle**

Una vez realizado este proceso tenemos los coeficientes $a_k$ del filtro deseado, llamado filtro del tracto vocal, además de un valor de energía residual, $E$. Cabe destacar que, aunque ya tengamos esos coeficientes $a_k$ nosotros hemos de incluir uno nuevo que será el primero de todos y cuyo valor será 1 para que el filtro IIR sea estable.

Realizando un último apunte, esa energía residual tiene una relación directa con la ganancia que necesita nuestro filtro del tracto vocal, que viene dada por la expresión $G = \sqrt{E}$.

Finalmente, el modelo del filtro del tracto vocal queda así:

$$H(z) = \frac{G}{1 + \sum_{k=1}^{P} a_k \cdot z^{-k}} \tag{F.9}$$

[1]**Nota**: Destacar que la notación que vamos a utilizar para las matrices es $r = R \cdot a$ que simbolizan $R_s(|j|) = \sum_{k=1}^{P} a_k \cdot R_s(|j-k|)$.

Por lo tanto, para poder sintetizar sonido con este filtro deberemos realizar el siguiente proceso:

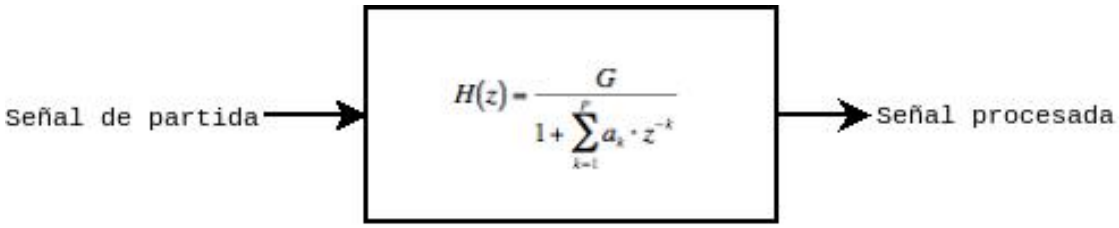


Figura F.2: Proceso de síntesis de sonido mediante LPC

# Apéndice G

# Códigos fuente

En este anexo se incluyen los códigos[1] creados para la realización de pruebas del Capítulo 5. Además de estos, el resto de códigos fuente implementados (tanto de los sistemas de transposición como de síntesis de voz), con la intención de no crear una memoria muy larga, son accesibles desde la dirección web comentada en el Prólgo.

1. Método de los formantes

```
instr 1; Sintesis de la m

idur = p3
iamp = ampdb(p5)
ifreq = cpspch(p4)

iharms=(sr*.4) / ifreq ; Cantidad de armonicos

asig    gbuzz   1, ifreq, iharms, 1, .8, 2 ; Senal a filtrar

;Envolvente
kenv    linseg  0, .1*idur, iamp, idur*0.8, iamp, .1*idur, 0

aout = kenv * asig ; Aplicamos la envolvente

;Filtrado de formantes
afilt_sa1  reson   aout, 172, 80, 1
afilt_sa2  resonz  aout, 215, 90, 1
```

[1]Es destacable el hecho de que los comentarios no cumplen con las reglas ortográficas del castellano por la imposibilidad de incluir acentos y consonantes $\tilde{n}$.

```
afilt_sa3  resonz  aout, 1249, 120, 1
afilt_sa4  resonz  aout, 3531, 130, 1

;Amplitud para cada formante
iamp1 = ampdb(70)
iamp2 = ampdb(70-6)
iamp3 = ampdb(70-20)
iamp4 = ampdb(70-32)

;Amplitud total
itotal = iamp1+iamp2+iamp3+iamp4

;Porcentaje
ip1 = iamp1 / itotal
ip2 = iamp2 / itotal
ip3 = iamp3 / itotal
ip4 = iamp4 / itotal

;Creacion de la salida
aout2 = (afilt_sa1 + ip2 * afilt_sa2 + ip3 * afilt_sa3 \
         + ip4 * afilt_sa4 )
out aout2

endin
```

Cabe destacar que en este código sólo se utiliza un tipo de tabla de onda, que es un ciclo de onda sinusoidal, y que podemos encontrar en la línea del *opcode* **gbuzz**. La tabla, tipificada como 2, tiene el siguiente código:

**f** 2 0 8192 **9** 1 1 0.25

2. LPC

```
instr 1 ; Sintesis de la consonante s

idur = p3
iamp = ampdb(p5)
ifreq = cpspch(p4)

;Envolvente de amplitud
kenv linseg 0,0.1*idur,iamp,0.6*idur,iamp,0.3*idur,0
```

```
;Senyal de para conseguir el nivel RMS correcto
ayuda oscil kenv, ifreq, 101

iharms=(sr*.4) / ifreq ; Numero de armonicos
asig gauss iamp ; Seal a filtrar

;Cargado del filtro LPC
krmsr, krmso, kerr, kcps lpread 1, "s_interp.lp"
asal lpreson asig ; Filtrado LPC

asal balance asal, ayuda ; Igualacion del nivel RMS
out asal
endin
```

Cabe destacar que, para la implementación de los diferentes casos, la señal a filtrar no siempre ha sido la misma: aunque en este caso tenemos ruido blanco de distribución gaussiana (*opcode* **gauss**) también se han utilizado señales periódicas generadas por el *opcode* **gbuzz**.

En cuanto a las tablas de ondas utilizadas encontramos una sinusoide cuyo código es:

**f** 101 0 1024 **10** 1

3. Envolventes de amplitud

```
instr 1 ; Sintesis de la consonante p
idur = p3
iamp = ampdb(p4)
ifreq = cpspch(p5)

ayuda oscil iamp,440,101 ; Senyal para igualar el nivel RMS

; Envolvente de amplitud
kenv linseg 1, 0.05*idur, 1, 0.1*idur, 0, 0.85*idur, 0

knh = sr*0.4/ifreq ; Cantidad de armonicos
asenal buzz iamp, ifreq, knh, 101 ; Senyal a filtrar

krmsr, krmso, kerr, kcps lpread 0, "p_lpc.lp" ; Carga del filtro
asenal1 lpreson asenal ; Filtrado LPC
```

```
aout2 balance asenal1, ayuda ; Igualacion del nivel RMS
aout2= aout2 * kenv ; Aplicacion de la envolvente

out aout2
endin
```

En este caso ocurre algo como en el anterior: además de la señal original que encontramos aquí, para otras pruebas también se ha utilizado señales de tipo ruido para el modelado de consonantes. Con esto, lo que se quiere es dejar claro que hay códigos que, en lugar de utilizar el *opcode* **gbuzz** han utilizado otros como **rand** o **gauss**.

En lo relativo a tablas de onda utilizadas, encontramos una sinusoide con el siguente código:

**f** 101 0 1024 **10** 1

4. Síntesis aditiva

```
instr 1 ; Sintesis de la consonante l
idur = p3
iamp = ampdb(p5)
ifreq = cpspch(p4)

ayuda oscil iamp,ifreq,101 ; Senyal para el ajuste RMS
kenv linen 1, 0.5*idur,idur,0.1*idur ; Envolvente de amplitud

;Creamos las sinusoidales
asin1 oscil ampdb(64),1723,101
asin2 oscil ampdb(38),2756,101
asin3 oscil ampdb(50),3747,101
asin4 oscil ampdb(20),3400,101

aout = asin1+asin2+asin3+asin4 ; Senyal con los formantes

aout = aout * kenv ; Aplicacion de la envolvente
aout balance aout,ayuda ; Ajuste de nivel RMS
out aout
endin
```

Para este código únicamente ha sido necesario el utilizar una tabla de onda correspondiente a una sinusoide, cuyo código es el que sigue:

**f** 101 0 1024 **10** 1

5. Tablas de onda

```
instr 1 ; Sintesis de la consonante s

idur = p3
iamp = ampdb(p5)
ifreq = cpspch(p4)

;Senyal de ayuda:
ayuda oscil iamp/10, ifreq, 1

asig2   oscil iamp, ifreq, 130 ; Senyal a filtrar

kenv    linseg  0, .1*idur, iamp, idur*0.8, iamp, .1*idur, 0
aout = kenv * asig2

;Filtrado por formantes
afilt_sa1  reson   aout, 400, 80, 1
afilt_sa2  resonz  aout, 1600, 90, 1
afilt_sa3  resonz  aout, 2600, 120, 1
afilt_sa4  resonz  aout, 3400, 130, 1

;Amplitud de cada formante
iamp1 = ampdb(70)
iamp2 = ampdb(70-6)
iamp3 = ampdb(70-32)
iamp4 = ampdb(70-20)

itotal = iamp1+iamp2+iamp3+iamp4

;Porcentaje
ip1 = iamp1 / itotal
ip2 = iamp2 / itotal
ip3 = iamp3 / itotal
ip4 = iamp4 / itotal

;Salida
aout2 = (afilt_sa1 + ip2 * afilt_sa2 + ip3 * afilt_sa3 \
         + ip4 * afilt_sa4 )
aout2 butterlp aout2, 4000
;Balance de potencia
```

```
aout2 balance aout2, ayuda

sout aout2

endin
```

Este código es exactamente igual al del filtrado por formantes excepto por el hecho que la forma de obtención de la señal de ruido, que ha sido por medio de tablas de onda. Cabe destacar que, aunque aquí aparezca el filtrado por formantes, en realidad se podra haber utilizado también el método LPC o el de las envolvente pues, como se ha dicho antes, lo novedoso del método es la utilización de tablas de onda para la obtención de la señal de excitación.

La tabla aquí utilizada, numerada como 130, tiene el siguiente código asociado:

**f** 130 0 1024 **21** 6

6. Phase–Vocoder

```
instr 1 ; Sintesis de la consonante k

idur = p3
iamp = ampdb(p5)
ifreq = cpspch(p4)

; Parametros previos:
ktimpnt=0.03
kfmod= 1
ifn=102
ibins=20

;Utilizacion de la sintesis del Vocoder en Fase
;asig pvoc 0.001, 1, "k_pv.pvx"
asig pvadd ktimpnt, kfmod, "k_pv.pvx", ifn, ibins

;Envolvente de amplitud
kenv linseg 1, 0.05*idur, 1, 0.1*idur, 0, 0.85*idur, 0

ayuda oscil iamp,ifreq,101 ; Senyal para igualar el nivel RMS
asig = asig * kenv ; Aplicacion de la envolvente de amplitud
```

```
asig balance asig, ayuda ; Ajuste de nivel RMS

out asig
endin
```

En este código se pueden ver las dos variantes llevadas a cabo en la experimentación:

- *Opcode* **pvoc**: Para la síntesis por medio de unos osciladores previamente definidos por CSound.
- *Opcode* **pvadd**: Síntesis en la que nosotros definimos la forma de onda de los osciladores (en nuestro caso, sinusoides y ruido aleatorio).

En lo relativo a las tablas de onda utilizadas en este código, básicamente se han utilizado las que siguen:

- Onda sinusoidal: **f** 101 0 1024 **10** 1
- Ruido aleatorio: **f** 102 0 8192 **21** 6

7. FOF

```
instr 1 ; Sintesis de la consonante m
iamp = ampdb(p4)
idur = p3
kenvol linen 1, 0.1*idur, idur, 0.5*idur ; Envolvente de amplitud

;Parametros comunes a todos los formantes:
kfund init 261.659
koct init 0
kris init 0.003
kdur init 0.02
kdec init 0.007
iolaps = 14850
ifna = 101
ifnb = 102
itotdur = p3

;Formante 1
iamp1=ampdb(70)
iform1=172
iband1=80
```

```
;Formante 2
iamp2=ampdb(64)
iform2=215
iband2=90

;Formante 3
iamp3=ampdb(50)
iform3=1249
iband3=120

;Formante 4
iamp4=ampdb(38)
iform4=3531
iband4=130

;Formante 5
iamp5=ampdb(20)
iform5=3400
iband5=140

; Creacion de las senyales con los formantes:
ar1 fof iamp1,kfund,iform1,koct,iband1,kris,kdur,kdec,iolaps,\
        ifna,ifnb,itotdur
ar2 fof iamp2,kfund,iform2,koct,iband2,kris,kdur,kdec,iolaps,\
        ifna,ifnb,itotdur
ar3 fof iamp3,kfund,iform3,koct,iband3,kris,kdur,kdec,iolaps,\
        ifna,ifnb,itotdur
ar4 fof iamp4,kfund,iform4,koct,iband4,kris,kdur,kdec,iolaps,\
        ifna,ifnb,itotdur
ar5 fof iamp5,kfund,iform5,koct,iband5,kris,kdur,kdec,iolaps,\
        ifna,ifnb,itotdur

ayuda oscil iamp, 440, 101 ; Senyal de ajuste RMS
asal = (ar1+ar2+ar3+ar4+ar5)/5 ; Obtencion de la senyal
asal balance asal, ayuda ; Ajuste RMS
asal = asal * kenvol ; Aplicacion de la envolvente

out asal
endin
```

En este caso, todos los programas creados para este método tienen esta misma estructura en la que primero se declaran las características de cada formante, después se crea cada señal con el formante correspon-

diente mediante el *opcode* **fof** y, finalmente, se obtiene la señal final como suma de todas.

En este caso, las tablas utilizadas han sido:

- Onda sinusoidal: **f** 101 0 4096 **10** 1
- Onda para el conformado del formante: **f** 102 0 1024 **19** 0.5 0.5 270 0.5
- Ruido aleatorio: **f** 103 0 8192 **21** 6

8. Samples

```
instr 1 ; Sintesis de la consonante z
ayuda oscil ampdb(p4),440,101 ; Senyal para el ajuste RMS
asal soundin "Z1.wav" ; Lectura de la muestra
asal butterlp asal, 4000 ; Filtrado paso-bajo
asal balance asal, ayuda ; Ajuste de RMS
out asal
endin
```

Cabe destacar que, para la síntesis con este método, en algunos casos como la s o la z hemos utilizado el filtro paso-bajo, el cual se puede ver en estas líneas de código, para eliminar componentes de alta frecuencia, mientras que en otros casos como la p no ha sido utilizado.

En lo relativo a las tablas de onda utilizadas, simplemente se ha utilizado una sinusoidal para crear una señal auxiliar para su ajuste RMS. El código de la misma es el que sigue:

**f** 101 0 1024 **10** 1

9. Opcodes

- **voice**

```
instr 1
kamp = ampdb(80) ; Amplitud senyal
kfreq = 440 ; Altura senyal
kphoneme = 0 ; Fonema a sintetizar
kform = 0.56 ; Ganancia del fonema
kvibf = 0.04 ; Frecuencia del vibrato
```

```
kvamp = 1 ; Amplitud vibrato
ifn = 101 ; Tabla para la sintesis del fonema
ivfn = 102 ; Tabla de vibrato

;Sintesis con el opcode
asal voice kamp, kfreq, kphoneme, kform, kvibf, kvamp, ifn, ivfn

out asal

endin
```

Las tablas de ondas utilizadas en este caso son:

- Seno: **f** 102 0 1024 **10** 1
- Tabla de fichero: **f** 101 0 256 **1** "eee.aif" 0 0 0

Respecto a esto, simplemente destacar que la tabla 102 es un seno porque es la que se utiliza para realizar el vibrato mientras que la 101 es el fichero externo por ser la base de la síntesis del fonema que se vaya a crear.

- **vosim**

```
instr 1 ; Sintesis de la consonante m
idur = p3
iamp = ampdb(p5)
ifreq = cpspch(p4)

ayuda oscil iamp, 440, 101

kamp = 2*iamp ; Amplitud de salida
kFund = ifreq ; Altura de la nota
kDecay = 0.1 ; Factor de bajada de cada pulso
kPulseCount = 10 ; Numero de pulsos en cada rafaga
;Factor de incremento/decremento de la anchura de cada pulso
kPulseFactor=0.9
ifn = 17 ; Tabla de onda

kenv linen 1, 0.01*idur, idur, 0.01*idur ; Envolvente de amplitud

;Primer formante
aout1 vosim kamp, kFund, 172, kDecay, kPulseCount, kPulseFactor, ifn
;Segundo formante
aout2 vosim kamp, kFund, 215, kDecay, kPulseCount, kPulseFactor, ifn
;Tercer formante
aout3 vosim kamp, kFund, 1249, kDecay, kPulseCount, kPulseFactor, ifn
;Cuarto formante
aout4 vosim kamp, kFund, 3531, kDecay, kPulseCount, kPulseFactor, ifn
```

```
;Creacion de la senyal final con los formantes
aout2 = aout1 + aout2 + aout3 + aout4

;Aplicacion de la envolvente
aout2 = aout2 * kenv

out aout2

endin
```

Cada formante se sintetiza con una serie de pulsos producidos por el *opcode* **vosim**, por lo cual necesitamos tantos como formantes haya en la señal.

Las tabla utilizada en este caso ha sido medio ciclo de una sinusoide para así realizar la síntesis de los pulsos glotales.

**f** 17 0 8192 **9** 0.5 1 0

- **fmvoice**

```
instr 1
idur = p3
iamp = ampdb(p4)
ifreq = ampdb(p5)
ivocal = 40 ; Vocal seleccionada
ktilt = 40 ; Ajuste espectral del sonido
itabla = 101 ; Tabla para la sintesis

kenv linen 1, 0.1*idur, idur, 0.1*idur ; Envolvente de amplitud

;Sintesis FM
asal fmvoice iamp, ifreq, ivocal, ktilt, 2, 2, itabla, \
             itabla, itabla, itabla, itabla
asal = asal * kenv ; Aplicacion de la envolvente de amplitud

out asal
endin
```

En este caso, la única tabla utilizada ha sido una sinusoide, cuya declaración es la que sigue:

**f** 101 0 1024 **10** 1

# Bibliografía


[1] Jesús Bernal Bermúdez, Jesús Bobadilla Sancho, Pedro Gómez Vilda, "Reconocimiento de voz y fonética acústica", Ed. Ra-Ma, Madrid

[2] Udo Zölzer, "Digital Audio Effects", Ed. John Wiley & Sons, Chichester

[3] Alan V. Oppenheim, Ronald W. Schafer, John R. Buck, "Tratamiento de señales en tiempo discreto", Prentice Hall

[4] Juan Manuel Sáez Martínez, Miguel Ángel Cazorla Quevedo, Ramón Rizo Aldeguer, "Métodos matemáticos en la computación", Publicaciones Universidad de Alicante

[5] Adolfo Núñez, "Informática y electrónica musical", Ed. Paraninfo, Madrid

[6] Martin Russ, "Sound Synthesis and Sampling", Ed. Elsevier

[7] Sergi Jordà Puig, "Audio digital y MIDI", Guías Monográficas Anaya Multimedia, Madrid, 1997

[8] Richard Boulanger, "The CSound Book", MIT Press

[9] Anastasia Georgaki, "Virtual voices on hands: Prominent applications on the synthesis and control of the singing voice"

[10] Perry R. Cook, "Singing Voice Synthesis: History, Current Work and Future Directions", Computer Music Journal (Massachussets Institute of Technology), 1996

[11] Jussi Pekonen, "Source-Filter Synthesis of the Singing Voice", TKK Helsinki University of Technology

[12] Michael W. Macon, Leslie Jensen-Link, James Oliveiro, Mark A. Clements, E. Bryan George, "A singing voice synthesis system based on sinusoidal modeling"

[13] Alex Loscos, "Spectral Processing of the Singing Voice", Universidad Pompeu Fabra, 2007

[14] "Apuntes Tratamiento Digital de Audio", Universidad de Alicante

[15] "Apuntes Técnicas de Reconocimiento y Síntesis del Habla", Universidad de Alicante

[16] "Apuntes Síntesis Digital del Sonido", Universidad de Alicante

# Glosario

- **A**
  - ADSR: Hace referencia al tipo de envolvente que, además de incluir los tres tramos de la envolvente ASR, añade uno más llamado Decaimiento, situado entre el Ataque y el Sostenimiento, que define la caída desde el nivel máximo que se alcanza con el Ataque hasta el nivel de señal constante del tramo de Sostenimiento.
  - Algoritmo: Lista bien definida y ordenada de instrucciones que permiten dar solución a un problema.
  - Aliasing: Se dice que una señal tiene aliasing cuando, por efecto de haber incumplido el Criterio de Nyquist, ésta queda distorsionada por efecto del solapamiento espectral entre diferentes bandas del espectro de la señal, dando lugar a la imposibilidad de la reconstrucción de la señal original.
  - ASR: Hace referencia al tipo de envolvente que se define en tres tramos llamado Ataque (el sonido va aumentado de volumen), Sostenimiento (parte estable del sonido en la que el volumen es constante) y Relajación (parte en la que el sonido se va apagando).
- **B**
  - Back-end: En software, back-end hace referencia a todo lo que se dedica a procesar la salida de información, contrariamente a un front-end, cuya finalidad es la comunicación con el usuario.
  - Bajo (música): Se dice de la voz cuya tesitura comprende, aproximadamente, desde un $MI_2$ (164.42 Hz) hasta un $DO_4$ (523.26 Hz).
  - Banda ancha (Señal): Se dice que una señal es de banda ancha cuando su espectro tiene bastantes componentes frecuenciales.
  - Banda-eliminada (filtro): Tipo de filtro que se encarga de suprimir componentes frecuenciales de una señal.

- Butterworth (filtro): Filtro diseñado especialmente para tener la respuesta más plana posible en la banda de paso del filtro.

- **C**

  - Cero: En Teoría de Filtros, este es el nombre que se le da al valor (o valores) que anulan el numerador de la ecuación característica de un filtro. En el dominio frecuencial se traduce en que se elimina la frecuencia que determine este valor.
  - Compás: Período de tiempo regular en que se divide una composición musical. También se dice del ritmo de la misma.
  - Contralto: Se dice, en canto, de la voz femenina más grave, cuya tesitura abarca, aproximadamente, desde un $FA_3$ (349.23 Hz) hasta un $FA_5$ (1,369.94 Hz).

- **D**

  - Decibelio: Décima parte de un belio. Representa la diferencia, en escala logarítmica, entre dos magnitudes, de las cuales una es, normalmente, una unidad de referencia.
  - Diagrama de pianola: Formato de representación musical (como puede ser la partitura clásica o la tablatura de guitarra), asociado históricamente a la pianola, en el cual la altura de las notas se ve como teclas de piano en sentido vertical y la duración se codifica en sentido horizontal por medio de tiempos y compases.
  - Directo (Método): Método de resolución de sistemas de ecuaciones lineales que permite obtener, si existe, la solución exacta al sistema.

- **E**

  - Entero (tipo de dato): Hace referencia al tipo de dato que únicamente puede representar número enteros (sin decimales).
  - Espectrograma: Representación de la evolución del espectro de una señal en el tiempo. Su representación, típicamente bidimensional, comprende unidades de tiempo en el eje X y unidades de frecuencia en el eje Y, representando la mayor o menor cantidad de señal en escala de colores para cada par (x,y).
  - Estocástico: En estadística se conoce así a cualquier proceso que actúa de forma aleatoria, por azar, sin ninguna regla capaz de representarlo.

- **F**
  - FIR: Tipo de filtro en cuya ecuación característica no existe ningún polo (en realidad, están todos en el origen).
  - Flotante (tipo de dato): También conocido como real, es un tipo de datos que puede almacenar números con decimales.
  - Fonema: Unidad fonológica mínima que diferencia significados.
  - Formante: Hace referencia a cada uno de los picos de mayor amplitud que aparecen en la representación en frecuencia de una señal, típicamente voz o sonido.
- **G**
  - Gauss (Método): Método directo de resolución de sistemas de ecuaciones de tipo lineal. Posee complejidad computacional cúbica.
  - Gauss-Jordan (Método): Método directo de resolución de sistemas de ecuaciones de tipo lineal. Posee complejidad computacional de tipo cúbica.
  - Gauss-Seidel (Método): Método iterativo de resolución de sistemas de ecuaciones. Suele ser más preciso, o al menos en un menor número de iteraciones, que el método de Jacobi, del cual procede.
  - Glotal (pulso): Hace referencia a la forma de onda que obtenemos al hacer vibrar nuestras cuerdas vocales pero sin llegar a ser filtrada por las cavidades bucales que proporcionan los formantes.
- **I**
  - IIR: Tipo de filtro en cuya ecuación característica sí que hay algún polo (en realidad, es que tiene algún polo fuera del origen).
  - IRCAM (Centro de Investigación): Instituto de Investigación y Coordinación sobre Acústica y Música (Institut de Reserche et Coordination Acoustique/Musique).
  - Iterativo (Método): Método de resolución de ecuaciones con el que no obtenemos nunca la solución exacta, sino una aproximación.
- **J**
  - Jacobi (Método): Método iterativo de resolución de ecuaciones.
  - Jitter: Se conoce como *jitter* a la variación que sufre el valor de la frecuencia de muestreo del reloj de un sistema conversor Analógico-Digital. Suele ser un valor pequeño.

- **L**
  - LFO: Oscilador de baja frecuencia.
  - LPC: Siglas de *Linear Predictive Coding* (Codificación Lineal Predictiva). Consultar el anexo correspondiente.
- **N**
  - Nyquist (Criterio): Criterio que, en Teoría de Señal, establece que la frecuencia de muestreo al digitalizar una señal ha de ser, como mínimo, igual al doble de la frecuencia máxima de la señal. En la práctica, la frecuencia de muestreo ha de ser siempre superior a ese valor para evitar aliasing.
- **O**
  - Octava: Se dice que entre dos frecuencias hay una relación de octava cuando una es el doble de la otra.
  - Offset: Se llama nivel de offset a la cantidad de señal de frecuencia cero (señal continua) que hay presente en la señal.
  - Opcode: Son las rutinas programadas del lenguaje CSound.
- **P**
  - Paso-alto (filtro): Tipo de filtro que únicamente deja intactas las componentes de frecuencia superiores a una que nosotros establezcamos.
  - Paso-bajo (filtro): Tipo de filtro que únicamente deja intactas las componentes de frecuencia inferiores a una que nosotros establezcamos.
  - Paso-banda (filtro): Tipo de filtro que únicamente deja intactas ciertas componentes frecuenciales que nosotros establezcamos.
  - Plugin: Se dice de una aplicación que es capaz de interactuar con otra, dotándola así de alguna función nueva.
  - Polo: En Teoría de Filtros, este es el nombre que se le da al valor (o valores) que anulan el denominador de la ecuación característica de un filtro. En el dominio frecuencial se traduce en que se aumenta el nivel de la frecuencia que determine este valor.
  - Puntero (programación): Comúnmente se le conoce como variable que apunta. Es un tipo de variable en la cual, en lugar de guardar valores (por ejemplo un entero o un flotante), guardamos una posición de memoria en la cual hay algo.

- **R**
  - Reverberación: Efecto de la persistencia del sonido en un recinto por efecto de la reflexión de los rayos sonoros en su interior.
  - RMS: Siglas de *root mean square* (Valor eficaz de una señal). Partiendo de una señal variable en el tiempo, se define como el valor (de tensión o corriente) que debería tener una señal constante para dar la misma cantidad de energía que la señal variable.
- **S**
  - Samples: Literalmente muestra, es un sonido grabado que se utiliza como base en la síntesis por muestreo o sampler.
  - Secuenciador: Aparato electrónico (normalmente previo a la aparición de los ordenadores personales) o programa informático (más extendido en la actualidad) que permite crear, controlar y reproducir eventos musicales. En el presente proyecto nos referimos, normalmente, a secuenciadores software MIDI.
  - Sonoro (sonido): Se dice que una señal es de este tipo cuando tiene una cierta componente periódica.
  - Soprano: Se dice de la voz cuya tesitura comprende, aproximadamente, desde un $DO_3$ (261.63 Hz) hasta un $DO_7$ (1,046.52 Hz).
  - Sordo (sonido): Representa señales que no tienen ninguna componente periódica.
- **T**
  - Tenor: Se dice de la voz en música cuya tesitura se extiende desde un $SI_2$ (246.95 Hz) hasta un $SOL_4$ (392 Hz).
  - Tesitura: Rango de frecuencias, o notas en un ámbito más musical, que es capaz de producir un instrumento (entendiendo esto desde cualquier instrumento artificial hasta, por ejemplo, la voz humana).
  - Tiempo (música): Cada una de las partes de igual duración en las que se divide un compás.
  - Timbre (música): Cualidad del sonido que hace individual a cada uno y que permite distinguirlo de otros aunque tengan el mismo tono e intensidad.

- $\underline{\text{Tr}_{20}}$: Es una forma de medida del tiempo de reverberación, concretamente lo que tarda un sonido en caer de su nivel más alto hasta un valor 100 veces inferior (lo que tarda en caer 20 decibelios). Con este parámetro se puede obtener el $tr_{60}$ multiplicando por 3.
- $\underline{\text{Tr}_{60}}$: También denominado *tiempo de reverberación*, es el tiempo que tarda un sonido en caer desde su valor más alto hasta la millonésima parte del mismo (en decibelios, es el tiempo que tarda en caer 60 decibelios desde el punto más alto).

- **V**

  - Vocoder: Acrónimo de *VOICE CODER*. Se conoce así a los aparatos y/o algoritmos capaces de realizar algún proceso sobre la señal de voz, siendo típicamente la compresión de la misma.
  - VST: Siglas de *Virtual Studio Technology*. Se trata de una interfaz, desarrollada por la empresa Steinberg, cuya finalidad es la integración de plugins de audio y demás utilidades de este tipo con los editores de audio. Actualmente es la tecnología más difundida.