Escuela Politécnica Superior

Estimación automática de acordes para uso en transcripción musical a partir de audio

Grado en Ingeniería en Sonido e Imagen en Telecomunicación

Trabajo Fin de Grado

Autor:
Miguel Segura Sogorb

Tutor:
José Manuel Iñesta Quereda

Septiembre 2015

Universitat d'Alacant
Universidad de Alicante

Estimación automática de acordes para uso en transcripción musical a partir de audio

Miguel Segura Sogorb

Agradecimientos

Es el momento de pararse un momento, tomar aire, y dar gracias a todas las personas que, de una manera u otra, han hecho que llegue a realizar este proyecto y que mi paso por la universidad sea ilusionante, inolvidable o, simplemente, posible.

Al mirar atrás recuerdo muchos momentos duros en los que fácilmente hubiera tirado la toalla. A todos los que me habéis regalado vuestro cariño, vuestras palabras de aliento y, también a veces, vuestra paciencia: gracias. Sé que en ocasiones puedo ser un poco como el personaje estrella de Woody, neurótico, soñador y difícil de soportar.

Este proyecto nace de una inquietud muy íntima hacia la música, que me acompaña siempre como una noble y desinteresada compañera. Quiero dar las gracias a José Manuel Iñesta por el apoyo recibido en su realización, así como por despertar en mí un vivo interés hacia la síntesis y la tecnología del sonido y de la música con sus clases en la asignatura Síntesis Digital de Sonido.

También en relación con este proyecto, quiero agradecer especialmente a José Javier Valero su infinita paciencia y su apoyo en el desarrollo del mismo. Voy a echar de menos esas explicaciones para no iniciados. Ánim amb eixe doctorat, xe, que te'l fas amb honors. Muchas gracias también a José Viejo por su asesoramiento en el momento más difícil.

Del mismo modo quiero agradecer a los profesores Sergio Bleda, Miguel Romá, Juan Manuel Sáez, Carolina Villalobos, Augusto Beléndez, Josué Nescolarde, Ana Amilburu y otros tantos que han hecho fácil lo difícil con sus clases, que han hecho que comience a pensar como un ingeniero y que quiera seguir aprendiendo.

A mis padres no les puedo pedir nada, porque me lo han dado todo, y no les puedo estar más agradecido. La Mari y Miguel padre han hecho posible que estudie en la universidad y me han apoyado en todo lo que he decidido acometer. Sabéis que si elijo hacer algo, lo hago con pasión y asertividad, y en el futuro voy a esforzarme porque siga siendo así. A mi abuela Concha y a mi abuelo Vicente, que siempre me dieron ánimos y cariño; fuisteis vosotros quienes me ayustasteis a que tuviese mi primer ordenador; sé que el iaio Vicente se sentiría muy orgulloso de que su nieto entienda esas líneas de código y esas ecuaciones tan bizarras. Gracias también a mi hermana Cris, mucha suerte en lo que te propongas, niña; a los tetes licenciados, Vicente y Alberto; a Vicente padre y Maite; a Rafa y a Carmen; a los altos e incombustibles primos de Onil… ¡gracias a todos!

A la familia Edhellen, que me ha dado algunas de las mayores alegrías de mi vida y siempre me ha sacado una sonrisa en los momentos difíciles. Gracias a Andrés, Dennis, Fer, Moresi, Álvaro, Marina, Nacho, Moi, sois únicos cada uno de vosotros. Edu, juntos descubrimos cómo sonaba el cuarto grado mayor en una tonalidad menor, qué maravilla, ¿verdad?

Gracias a mis compañeros de carrera y amigos Hirahi, Dani, Ángel, Josélu, Jorge, Jaime, Leti, Ima, Lupo, Jesús, Iván, Álex, Juanfran, y todos los que me dejo. Me alegro de haber compartido con vosotros estos años y espero que volvamos a organizar alguna que otra teleco-jam, esos momentos son los que siempre van a valer la pena.

No me extiendo más, por Dios. Os quiero a todos *más que al cali'*, ahora podéis disfrutar de este texto que me ha costado tanto generar. Hacedlo, malditos. ¡Un abrazo a todos!

Índice de contenidos

Capítulo 1

Introducción al proyecto

Los acordes de una pieza musical aportan información concisa acerca del perfil armónico de la misma. Prueba de esto es que este conocimiento es a menudo suficiente para que un conjunto de instrumentistas toquen juntos, sin haber ensayado previamente y sin tener delante una partitura.

Conocer la progresión de acordes y transcribirla en texto posee un gran interés en el contexto de la creación e interpretación musical, pero más allá de esto aporta una muy valiosa información estructural y semántica, y por ello juega un papel importante en la organización de la información de las grandes colecciones de música digital actuales, especialmente cuando dicha progresión armónica puede inferirse de manera automática.

De aquí a unos años atrás se ha venido desarrollando la técnica aplicada a esta tarea, incrementándose la precisión y sofisticación de los algoritmos y pudiendo así incluir en el análisis acordes más complejos, como los cuatriada y sus inversiones. Con ello, las posibilidades de la estimación han ido aumentando cada vez más.

Los investigadores han hecho de la detección automática de acordes un importante elemento a la hora de realizar numerosas tareas de alto nivel como la estimación de tonalidad, la búsqueda e identificación de versiones de un mismo tema, la sincronización letra-música, la identificación automática de estilo o compositor, etc. Así mismo, puede hacerse uso de esta herramienta como soporte armónico en la tarea de transcripción automática de audio a partitura, que se encuentra en desarrollo en la actualidad.

En el presente proyecto va a diseñarse un sistema de estimación automática de acordes en base al análisis de archivos de audio, en el contexto de un proyecto mayor dedicado a la transcripción polifónica monotímbrica desarrollado por el Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Alicante. Se trabajará, a priori, extrayendo características de croma y siguiendo métodos de similitud de patrones o plantillas, proponiendo en primer lugar un prototipo software e incrementando el grado de sofisticación en la versión final, en la medida en que el tiempo nos lo permita. El sistema deberá presentar a su salida un listado de acordes referenciado a los instantes de tiempo en los que éstos suenan en el audio de entrada.

1.1. Origen del proyecto

Este proyecto nace a raíz de una propuesta del autor, refinada por José Manuel Iñesta Quereda con el fin de integrar el sistema de estimación automática de acordes diseñado en un sistema mayor dedicado a la transcripción musical automática de audio a partitura, desarrollado en el contexto de las investigaciones que se están llevando a cabo, al respecto de esta tarea, en el Departamento de Lenguajes y Sistemas Informáticos (DLSI) de la Universidad de Alicante.

Para ello, se abordará el diseño desde un punto de vista modular y flexible, pudiendo cada uno de los componentes que componen el código ser revisado y mejorado en cualquier momento sin excesivos ambages.

Asimismo, el enfoque adoptado es el de tratamiento de la señal de audio, aplicando sobre ésta técnicas de procesado en el dominio de la frecuencia. No se emplea, por tanto, un tratamiento simbólico de la información musical contenida en la señal, que requeriría expresarla en código MIDI (lo cual sería ya en sí mismo una transcripción).

1.2. Objetivos

Los objetivos principales del presente proyecto se detallan a continuación:

- Comprender las técnicas de procesado más comunes relativas a la extracción de información musical (MIR), y en particular a la estimación automática de acordes (ACE).
- Conocer los parámetros intervinientes en la ejecución de los algoritmos de ACE.
- Estudiar el efecto de dichos parámetros en la precisión de la estimación.
- Extraer información armónica de señales de audio con contenido tonal, dentro del marco de la música occidental.
- Crear un sistema que automatice el análisis mencionado y pueda extenderse a bases de datos completas de ficheros de audio.
- Llevar a la práctica conceptos de teoría de la señal para el desarrollo de software específico.

1.3. Metodología de trabajo

Para lograr los anteriores objetivos ha de seguirse una metodología de trabajo que permita cubrir todos los aspectos necesarios y producir gradualmente el sistema descrito:

- Fase de revisión bibliográfica (I): en esta fase se asientan los fundamentos teóricos y se recaba la bibliografía específica, ya sean tratados de teoría de la señal, tesis sobre MIR, trabajos de investigación publicados en revistas especializadas o tratados de música. Es especialmente

importante para contextualizar los esfuerzos posteriores y acotar el alcance final del proyecto, en base a las técnicas escogidas para el desarrollo del mismo. Puede aportar también múltiples ideas en los diferentes estadios del proyecto para solventar problemas concretos.

- Fase de exploración (II): una vez recabada toda la información y valorado las opciones de implementación, comienza una etapa exploratoria en la cual se realizará el estudio comparado de diferentes soluciones software al problema de la ACE, así como un primer diseño modular del sistema y su prototipado en lenguaje Matlab. También se diseña un modelo de evaluación del prototipo diseñado, que deberá dar cuenta de cuán precisa es la estimación.

- Fase de desarrollo (III): finaliza el prototipo inicial y su evaluación y comienza a implementarse una versión definitiva del sistema, refinando el prototipo e incluyendo nuevas características.

- Fase de evaluación (IV): finaliza el sistema final y comienza la etapa de evaluación completa. No se descarta un flujo de trabajo iterativo entre la fase III y la IV, añadiendo nuevas características al sistema final, en el caso de que la evaluación arroje resultados pobres.

- Fase de documentación (V): A medida que se desarrollan estas fases se irán reflejando los progresos en la memoria técnica.

Como herramientas se emplearán el software *Sonic Annotator*, desarrollado por la Universidad Queen Mary de Londres, y diversos plug-ins de análisis de audio (*Vamp plug-ins*) para realizar las transformaciones esenciales sobre la señal. Los resultados de estas transformaciones podrán ser estudiados gráficamente mediante *Sonic Visualiser*.

Por otra parte, la estimación de acordes en sí será desarrollada de manera modular empleando el lenguaje de programación MATLAB, que permitirá implementar las funcionalidades específicas que se deseen en cada etapa. Existen paquetes software (*toolbox*) para MATLAB, de diferentes desarrolladores, específicos del área de conocimiento que nos ocupa. Si lo estimamos necesario incorporaremos alguno de estos paquetes.

Contaremos con otras herramientas auxiliares en determinadas fases del proyecto, como el lenguaje *Csound* para sintetizar señales de prueba con los que evaluar los algoritmos de análisis de audio escogidos, el software *Audacity*, también de libre distribución, cuando sea necesario extraer fragmentos de audio de la grabación de una pieza o visualizar la forma de onda rápidamente, *Notepad++* y *Microsoft Excel* como apoyo para la elaboración y manejo de anotaciones de referencia y *Avid Sibelius* para presentar diferentes figuras que requieran utilizar notación musical.

1.4. Motivación de la extracción de información musical

La cantidad de música disponible en formato digital crece de forma continua, hecho motivado por un creciente interés por parte de los usuarios y por el desarrollo de nuevas tecnologías para el acceso y disfrute ubicuo de la música.

Existen diversas razones que pueden explicar esta tendencia; en primer lugar, las características del propio lenguaje musical. La música es una forma de arte que puede ser compartida por personas procedentes de diferentes culturas, puesto que traspasa las barreras de las lenguas vernáculas y los contextos culturales, empleando un código propio vinculado directamente con el nivel emocional.

Otra razón es que la tecnología de la grabación musical, digitalización y reproducción permite a los usuarios un grado de acceso que es tan solo comparable a la escucha de una interpretación en directo, al menos en lo que respecta a calidad de audio, ya que la relación señal a ruido es superior en los formatos digitales con respecto a muchos de los formatos analógicos, situación que no se reproduce en el caso de otras formas de arte, que necesitan ser apreciadas in situ.

O tal vez el creciente interés hacia la música digital está motivado por su portabilidad y la posibilidad de acceder a dicha música mientras se realizan otras actividades de carácter principalmente primario (desplazarse, hacer deporte o incluso trabajar y estudiar), pasando a entenderse como un complemento dinamizador en lugar de como una obra de arte.

Por último, pero no menos importante, la industria musical persigue la producción ininterrumpida de nuevas piezas musicales, especialmente en géneros como el pop y el rock. La continua disminución de la media de edad de las personas que habitualmente compran y consumen música se ha producido en paralelo con una creciente simplificación a nivel de estructura de los principales géneros de música comercial, lo cual facilita y motiva esta producción masiva. El número de ítems vendidos a diario en las plataformas de venta de música basadas en web, o descargadas de servicios como iTunes –sin mencionar la compartición ilegal de archivos musicales a través de redes punto a punto–, pone de manifiesto hasta qué punto la música es importante cultural y comercialmente.

La extracción de información musical (*Music Information Retrieval*, MIR) es un área de investigación emergente dedicada a satisfacer las necesidades de información musical que tienen los usuarios. Como se verá, a pesar del énfasis en la palabra "extracción", la MIR engloba un amplio abanico de enfoques y tareas orientadas a la gestión, accesibilidad y consumo de la música. La mayor parte del trabajo de investigación en MIR, de las técnicas propuestas y de los sistemas desarrollados se basan en el conocimiento del contenido musical (Orio, 2006).

La idea principal subyacente de los enfoques basados en contenido es que un documento puede ser descrito por un conjunto de características que son directamente inferidas por un ordenador a partir

del contenido del propio documento. Habitualmente, el acceso basado en contenido a datos multimedia requiere metodologías específicas que han de ser adaptadas a cada medio particular. Las principales técnicas de extracción de información (*Information Retrieval*, IR), que se basan en teoría probabilística y estadística, son empleadas generalmente en medios no textuales, porque poseen una complejidad distinta; los modelos subyacentes a estos medios es probable que describan características fundamentales compartidas por muchos de los elementos del medio, pertenecientes al lenguaje y dominio de aplicación dado. No obstante, los resultados alcanzados en la investigación en IR, en particular en el caso de los documentos de texto, son una referencia constante para los enfoques propuestos en MIR.

La hipótesis básica de este punto de vista es que los metadatos son inadecuados, poco fiables o no están disponibles. En el caso de MIR, las tres premisas son válidas. Los metadatos de la música pueden no ser adecuados debido a que sean muy genéricos para diferenciar distintas piezas musicales (existen cientos de baladas cantadas por una voz femenina con guitarras acústicas de fondo) o demasiado específicos, sin por ello resultar útiles. Los metadatos pueden no ser fiables en el caso de la música compartida con otros usuarios, porque no hay control de cómo la información ha sido añadida, y por supuesto puede que no haya en absoluto metadatos. Esto tiene su origen en la falta de previsión en el momento de la creación del estándar MPEG Layer-3 o MP3, que no dispuso una estructura rigurosa para almacenar esta información salvo campos opcionales que debe completar el autor o la persona que genera el archivo. A pesar de estos problemas, han sido desarrollados diversos sistemas que permiten a los usuarios acceder y extraer información de música basados en descriptores textuales, y que están disponibles en la red.

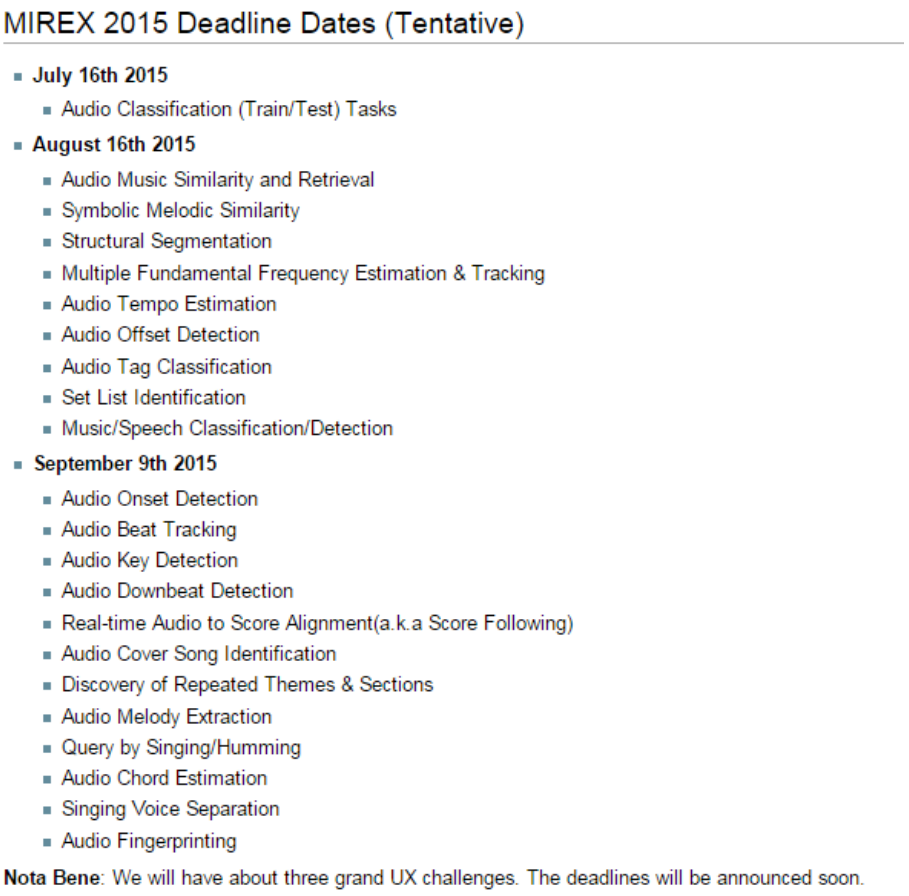
Se ha de tener presente que la música no transmite generalmente conceptos ni puede ser descrita en términos de ideas que presumiblemente transmite, por lo que no podrán emplearse palabras clave en este sentido para segmentar un elemento musical en una colección salvo que dichos conceptos figuren expresamente en el título de la pieza (*La tempestad* de Beethoven, por ejemplo). La música debe describirse, por tanto, casi exclusivamente en términos musicales que caractericen a la pieza, generalmente en su totalidad, aludiendo a su estructura interna.

El debate sobre las relaciones entre el enorme campo de investigación de las bibliotecas digitales y la MIR haría necesaria una disertación mucho más extensa dedicada en exclusiva a esto (como por ejemplo en Gómez, 2006a u Orio, 2006) y, si bien la motivación de la extracción musical tiene su origen en gran medida en esta problemática y nos ayuda a contextualizar la tarea que nos ocupa, no es el foco principal del presente trabajo.

Existe un foro internacional de puesta a prueba de las investigaciones sobre extracción musical, conocido como MIREX (*Music Information Retrieval Evaluation eXchange*), que se celebra

anualmente. Es organizada principalmente por la Universidad de Illinois, aunque se caracteriza por contar con una comprometida comunidad investigadora a nivel internacional; prueba de ello es la existencia de la ISMIR (*International Society of Music Information Retrieval*). Se hace uso de una plataforma wiki en la que se deja constancia de los avances de anteriores convocatorias y se organiza el trabajo para las siguientes. En la Figura 1.1 vemos un extracto de dicha web, en la que puede observarse, además, un listado de las múltiples tareas de las que se ocupa la disciplina.

Otras importantes asociaciones como el IEEE (*Institute of Electrical and Electronics Engineers*), la ASA (*Acoustical Society of America*) y la AES (*Audio Engineering Society*) organizan numerosos simposios en todo el mundo, realizan publicaciones especializadas regulares y elaboran los estándares del sector, entre otras actividades, en las que las tecnologías del sonido y la música están muy presentes.



MIREX 2015 Deadline Dates (Tentative)

- **July 16th 2015**
  - Audio Classification (Train/Test) Tasks
- **August 16th 2015**
  - Audio Music Similarity and Retrieval
  - Symbolic Melodic Similarity
  - Structural Segmentation
  - Multiple Fundamental Frequency Estimation & Tracking
  - Audio Tempo Estimation
  - Audio Offset Detection
  - Audio Tag Classification
  - Set List Identification
  - Music/Speech Classification/Detection
- **September 9th 2015**
  - Audio Onset Detection
  - Audio Beat Tracking
  - Audio Key Detection
  - Audio Downbeat Detection
  - Real-time Audio to Score Alignment(a.k.a Score Following)
  - Audio Cover Song Identification
  - Discovery of Repeated Themes & Sections
  - Audio Melody Extraction
  - Query by Singing/Humming
  - Audio Chord Estimation
  - Singing Voice Separation
  - Audio Fingerprinting

Nota Bene: We will have about three grand UX challenges. The deadlines will be announced soon.

Figura 1.1. Extracto de la *wiki* oficial del MIREX. Convocatoria de trabajos de investigación para la celebración del MIREX 2015[1] (www.music-ir.org/mirex).

[1] Todas las figuras que no hacen referencia explícita a una fuente son de elaboración propia.

1.5. Enfoques de la extracción de información hacia la transcripción musical

La música es una forma de arte que consiste en la disposición de sonidos y silencios de manera organizada a lo largo del tiempo. Por tanto, no es de extrañar que la MIR clasifique algunas de las características musicales más importantes en términos de tiempo, atendiendo a cuánto necesita un oyente para apreciar dichas características (Orio, 2006). Éstas son: timbre, orquestación/instrumentación, acústica, ritmo, melodía, armonía y estructura, y se presentan clasificadas en la tabla 1.1, junto a una breve descripción de las mismas.

Escala temporal	**Dimensión**	**Contenido**
Corto plazo	Timbre	Cualidad del sonido producido (fuente)
	Orquestación	Fuentes que producen el sonido
	Acústica	Cualidad del sonido producido (entorno)
Medio plazo	Ritmo	Patrones en el tiempo formados por los *onsets*
	Melodía	Secuencias de notas
	Armonía	Secuencias de acordes
Largo plazo	Estructura	Organización de la pieza musical

Tabla 1.1. Escala temporal y características de las diferentes dimensiones de la música (Orio, 2006).

Las características a medio plazo son las que más información sensible aportan acerca de una composición musical, si obviamos en primera instancia la tímbrica y entendemos el factor estructural como una consecuencia lógica de la organización del material a medio plazo (esto es, ritmo, melodía y armonía) en un periodo de tiempo largo. Estos materiales pueden detectarse en intervalos desde 50 a 100 ms (Everest, 2001) y desarrollarse hasta alcanzar decenas de segundos, conformando las líneas maestras de la partitura.

Una partitura musical es una colección estructurada de símbolos que se corresponden con eventos acústicos y que describe los medios y procedimientos necesarios para su producción. La simbología de la partitura musical no es la única manera de representar la música, pero es la que ha sido empleada tradicionalmente en la música occidental desde su introducción de la mano de Guido d'Arezzo a

comienzos del siglo XI. La misma notación ha sido aplicada, con escasas modificaciones, a multitud de géneros y a la música tradicional de muchos países, como el pop, el rock o el jazz. Por esta razón, la notación musical occidental es la referencia que se toma casi siempre en MIR (en el Capítulo 3 se verá con mayor detenimiento en qué consiste).

En un segundo nivel, la interpretación musical tiene diversa naturaleza en función del género y el tipo de instrumentista, y su análisis arroja diversas ideas acerca de la aplicación de la MIR. La música clásica se interpreta siguiendo estrictamente una partitura musical, que es producida a priori por un compositor. Puede no ocurrir así en el caso de la música pop, rock y jazz. En estos géneros puede o no existir partitura, y si existe es posible que haya sido producida a posteriori de una grabación de las piezas musicales. No debemos olvidar tampoco la música improvisada, algo común especialmente en el caso del jazz y las músicas folclóricas: en este caso, composición e interpretación se producen simultáneamente, en base a pocas o ninguna pauta acordada previamente entre los músicos. Cabe preguntarse si no resultaría interesante registrar esta interpretación improvisada y generar una partitura a posteriori, empleando para ello métodos computacionales.

Un sistema de transcripción musical automática debe ser capaz de analizar una señal de audio con el fin de detectar los diferentes elementos que la componen. Transcribir música por métodos computacionales es una tarea muy compleja que permanece todavía sin resolver. Existen principalmente dos enfoques: transcripción audio-MIDI y transcripción audio-partitura. La utilización del estándar MIDI permite abstraerse de muchos de los parámetros musicales que deberían ser capturados en caso de tener que representar la música en una partitura convencional, por lo que es un enfoque extendido (Kabamba, 2013). En modelos de transcripción semiautomática puede darse que se obtenga una secuencia MIDI a partir de audio de manera automática para que, posteriormente, un experto complete esa información manualmente para obtener la partitura.

Generalmente un sistema de transcripción se compone de numerosas tareas que se realizan secuencialmente. La más básica de ellas es la estimación polifónica de altura (*multipitch estimation*), que consiste en tratar de determinar los contornos de frecuencia fundamental que están sonando en cada instante de tiempo en la señal de audio. Típicamente se lleva a cabo empleando técnicas de procesado de señal y presenta una gran complejidad. Si bien no se ha logrado una precisión del 100% en señales monofónicas, sí se pueden garantizar buenos resultados. En el caso de las señales polifónicas, especialmente si son multitímbricas, es realmente complicado lograr resultados prácticos (véase el Capítulo 4 para una discusión más detallada acerca de estas tareas). Por otro lado, la detección de *onsets* (inicio de nota) y *offsets* (final de nota) puede ayudar a delimitar la melodía y los cambios armónicos, y contribuir a la obtención de información de tempo y métrica. Un diagrama de pianola típico del MIDI ya podría obtenerse en base a las tareas mencionadas.

El desarrollo de la transcripción se ha basado en las técnicas de procesado digital de señal (y, ocasionalmente, en el aprendizaje automático o *machine learning*), pero las aproximaciones actuales al problema no son capaces de capturar la gran diversidad que puede encerrar una señal de audio. Se habla de que se ha llegado a un límite práctico (*glass ceiling*) con las técnicas actuales (Benetos, et al., 2012) y algunos autores proponen el uso de transcripción semiautomática, como decíamos, con la intervención de un experto humano para obtener mejores resultados, así como la aplicación de enfoques menos convencionales (Benetos, et al., 2013), como la transcripción interactiva y la multimodal.

Hablar de interacción implica que el usuario es parte del sistema. Este es el caso de la transcripción semiautomática o asistida: los errores que se dan en el resultado de la transcripción son corregidos por el usuario. Una característica que algunos autores exploran es que el sistema aprenda del usuario, combinar la transcripción asistida con el aprendizaje automático para que las correcciones pasadas doten al sistema de cierta inteligencia de cara a las futuras y cada vez se cometan menos errores, reduciéndose así la carga de trabajo del usuario.

Por otro lado, la transcripción multimodal busca integrar varias fuentes de información para mejorar la precisión del sistema. Por ejemplo, puede combinarse la información de tonalidad con los resultados obtenidos de la estimación de alturas. Desde un punto de vista probabilístico, una estimación de alturas puede entenderse como la probabilidad de que una frecuencia dada esté presente en un instante de tiempo determinado. Si se dispone de la información de tonalidad, se sabe qué notas es más probable que aparezcan. Así, pueden combinarse dichas informaciones de varias maneras: como post-procesado, valorando que una nota pueda estar presente o no en función de la tonalidad en la que se encuentre el fragmento y modificar si procede los resultados, o como información a priori, condicionando el sistema para que la estimación de alturas dé mayor peso a las notas propias de la tonalidad.

Es importante distinguir entre descripción y transcripción. Muchos trabajos en la literatura han tratado de transcribir piezas musicales, es decir, extraer una representación en partitura a partir de la señal de audio. Gómez afirma que la transcripción no es un paso imprescindible para la descripción del contenido musical (Gómez, 2006a). No obstante, puede ser un objetivo en sí mismo, como es el caso de las investigaciones que se están llevando a cabo en el DLSI de la Universidad de Alicante.

1.6. Evolución de la estimación automática de acordes

En los últimos años, la estimación automática de acordes (*Automatic Chord Estimation*, ACE) se ha convertido en un área de investigación muy activa, atrayendo a un amplio espectro de

investigadores en ingeniería eléctrica, informática, procesado de señal e inteligencia artificial. Los sistemas ACE han sido incluidos en la lista de tareas a evaluar del MIREX, y se ha producido una lenta pero constante mejora de la precisión desde sus inicios en 2008, superando los algoritmos presentados en 2012 el 72% de precisión sobre los conjuntos de datos empleados, medido en términos de muestras temporales identificadas con éxito de entre una colección de canciones de las cuales se conoce la progresión de acordes veraz (*ground truth* o anotaciones de referencia). Como veremos a la hora de diseñar el método de evaluación del sistema, no se trata exclusivamente de valorar un acierto o un fallo. Se valoran positivamente situaciones en las que el error ha sido mínimo siguiendo algún criterio establecido previamente.

Muchas de las técnicas de modelado utilizadas en ACE son también de interés general en MIR siempre que hay involucrada una tarea de identificación de secuencias. Los acordes definen el esqueleto de la música occidental, y como tal es lógico que cualquier tarea de MIR que esté relacionada con la detección de clases de notas musicales (independientemente de su octava) se beneficie de la comprensión de los acordes.

En general, un algoritmo de reconocimiento de acordes se compone de dos etapas clave: extracción de características y análisis de patrones. Éstas son acompañadas de otras dos etapas opcionales de refinamiento (pre o post-procesado), como se ilustra en la Figura 1.2. En el primer paso importante, una señal de audio dada es troceada en muestras y cada una de ellas es transformada en un vector de características apropiado para interpretar la información musical que se está dando en ese instante de tiempo. En el segundo paso, se determinan los nombres de los acordes en cada instante, en base a la comparación de los vectores de características con un modelo.

Las características musicales que se obtienen del audio comenzaron siendo simplemente espectrogramas con octavas y alturas sumadas, conocidos como cromagramas (*chromagram*), pero han ido incorporando de manera continua diferentes mejoras como afinación, eliminación del espectro de fondo *y* sincronización con el pulso musical, entendidas todas ellas como pre-procesado según el esquema presentado.

En paralelo al diseño de características que extraer del audio, decodificar los cromagramas para obtener una secuencia de acordes estimada comenzó con un simple decodificador Viterbi siguiendo una arquitectura de modelo oculto de Markov (*Hidden Markov Model*, HMM), pero en los últimos años ha aumentado la complejidad de esta etapa, haciendo posible la predicción de acordes de séptima e inversiones a través del uso de arquitecturas HMM factoriales y redes bayesianas dinámicas (*Dynamic Bayesian Networks*, DBNs) (McVicar, et al., 2014).

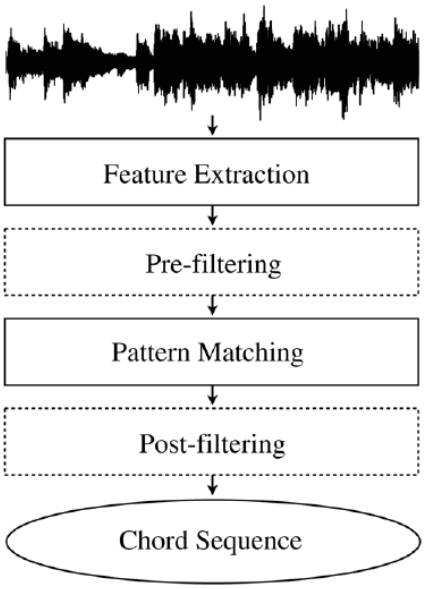


Figura 1.2. Arquitectura básica de un sistema de reconocimiento de acordes (Cho y Bello, 2014).

En general, pueden contrastarse dos enfoques: sistemas basados en datos frente a sistemas expertos. Existe una gran cantidad de datos disponibles, con anotaciones parciales o totales, que la comunidad científica emplea para entrenar sus modelos. Desde las primeras colecciones elaboradas artesanalmente con 180 canciones de los Beatles (*Isophonics*), gradualmente se ha ido incrementando el número de colecciones provistas de anotaciones completas, y recientemente hemos visto convertirse a la colección Billboard (de cerca de 1000 acordes y anotaciones) en la más significativa en los últimos años. Algunos autores han estado explorando el uso de colecciones parcialmente anotadas como fuente adicional de información.

La abundancia de información disponible en la actualidad en forma de archivos de audio de acordes (que pueden incluir hasta cuatro o cinco factores, estando algunos más allá del ámbito de octava, y considerarse las inversiones) ofrece la oportunidad a los investigadores de ampliar los horizontes en la tarea de detección automática, y al mismo tiempo los exhorta a ello, pues con el fin de incrementar la tasa de acierto en la evaluación se ha limitado la colección objeto de estudio durante mucho tiempo a las triadas mayor y menor en los trabajos presentados al MIREX (Barbancho, et al. 2013; McVicar, et al., 2014).

1.7. Sistemas comerciales de estimación automática de acordes

Por un lado existen diferentes paquetes de software disponibles referidos a la ACE que han ido saliendo en los últimos años. Desde el comienzo del nuevo siglo ha habido mejoras graduales pero constantes en las implementaciones de sistemas ACE.

Por ejemplo, el analizador musical Melisma[2], cuya primera versión se publicó en 2000, ofrece en su última versión código fuente en C que utiliza lógica probabilística para identificar información métrica, armónica y de flujo a partir de audio.

Más recientemente, el repositorio ACE de labROSA[3] elaboró una colección de algoritmos de MATLAB sobre estimación de acordes supervisada que fue presentado al MIREX 2008, 2009 y 2010, empleando en primer lugar un sencillo sistema de arquitectura HMM gaussiana y finalizando con el uso de avanzados HMM discriminantes. Éstos realizan la estimación de acordes en base a cromagramas sincronizados con el pulso musical.

Otro software muy útil es Chordino. Proporciona un sistema experto basado en el algoritmo NNLS Chroma[4], que también empleamos en el presente trabajo para el cálculo de cromagramas. Este software ha sido utilizado para aplicaciones web como Yanno[5], que permite a sus usuarios extraer los acordes de vídeos de YouTube. Otro sistema comercial similar es Chordify[6], que admite tanto enlaces de YouTube como de Soundcloud o Deezer, y dispone de funcionalidades gratuitas y otras tan solo disponibles bajo suscripción.

En el momento actual, el sistema ACE más avanzado es el HPA[7] (*Harmony Progression Analyzer*). Éste es un sistema de reconocimiento simultáneo de tonalidad, acordes y línea de bajo que se basa exclusivamente en técnicas de aprendizaje automático. Se incluye en el software un modelo previamente entrenado y código para volver a entrenarlo en base a nuevas anotaciones de referencia.

Otros programas de propósito musical general se han convertido en piezas muy relevantes dentro de la estimación de acordes. Vamp[8] es un sistema de soporte y creación de plug-ins de procesado de audio que extrae información descriptiva de éste. Basado en dicha tecnología, Sonic Annotator[9] ofrece una herramienta para la extracción de características y la confección de anotaciones de archivos de audio. Funciona como programa plataforma para ejecutar plug-ins Vamp en un amplio

[2] http://theory.esm.rochester.edu/temperley/melisma2/
[3] http://labrosa.ee.columbia.edu/projects/chords/
[4] http://isophonics.net/nnls-chroma
[5] http://yanno.eecs.qmul.ac.uk/
[6] http://chordify.net/
[7] https://patterns.enm.bris.ac.uk/hpa-software-package
[8] http://vamp-plugins.org/
[9] http://omras2.org/SonicAnnotator

abanico de formatos de audio, y puede escribir los resultados en una también amplia selección de formatos de salida. Finalmente, Sonic Visualiser[10] proporciona una aplicación para visualizar y analizar los contenidos de los archivos de audio musical. Sonic Visualiser y Chordino tienen la ventaja de permitir la visualización de secuencias de acordes predichas, permitiendo a los usuarios interpretar música intuitivamente observando los resultados del análisis (McVicar, et al. 2014).

[10] http://www.sonicvisualiser.org/

Capítulo 2

Conceptos físicos y matemáticos relevantes

2.1. Transformada discreta de Fourier

En este capítulo se presentan conceptos de la física del sonido y de los modelos matemáticos funcionales que lo describen y que tienen relevancia en el presente trabajo. Se consideran básicas las matemáticas involucradas en el análisis espectral de una señal analógica e, inversamente, en la reconstrucción de la onda a partir de las amplitudes y fases de su espectro.

A la hora de acometer el procesado de una onda sonora digitalmente registrada, surge rápidamente la pregunta, ¿cómo calcular su espectro en el dominio digital? Cuando registramos digitalmente un sonido ya no tenemos funciones, sino series de números, por lo que la teoría de Fourier en el ámbito analógico podrá ser válida sólo como concepto pero no es aplicable directamente. La clave de la aplicación en el dominio digital es la transformada *discreta* de Fourier (*discrete Fourier transform*, DFT).

Las señales que manipulan los ordenadores han sido previamente sometidas a un proceso de muestreo. En estas señales, el tiempo no transcurre de manera continua, sino con pequeños saltos equiespaciados determinados por la frecuencia a la que se han tomado las muestras de la señal continua (la denominada *frecuencia de muestreo*, f_s). La inversa de esa frecuencia es el periodo de muestreo $T_s = 1/f_s$, que indica el tiempo que transcurre entre cada dos muestras consecutivas. Por tanto, la magnitud continua *tiempo*, t, se representa en el ámbito digital mediante un índice $n \in \mathbb{N}$ de un vector en el que se van almacenando las muestras[11].

Supongamos que la señal muestreada es un vector de N valores, $s[n]$, con $n \in [0, N-1]$. Cada muestra nos dirá el valor de la señal en nT_s. La DFT calcula el espectro mediante

$$S[k] = \sum_{n=0}^{N-1} s[n]\, e^{-j\frac{2\pi kn}{N}} \quad , \qquad k = 0, \dots, N-1 \quad .$$

De la misma manera que en el caso continuo, $S[k]$ es un número complejo y con él podemos conocer tanto el espectro de amplitudes como el de fases. Si sólo queremos calcular el espectro de amplitudes,

[11] Para las señales discretas se emplea la notación de corchetes [ ] porque se trata de vectores de valores del mismo tipo almacenados en la memoria de un ordenador, frente a las señales continuas que se representan como funciones; éstas aparecen con paréntesis ( ).

basta con utilizar una de las componentes (seno o coseno). En particular emplearemos sólo números reales:

$$|S[k]| = \sum_{n=0}^{N-1} s[n] \sin\left(-\frac{2\pi kn}{N}\right), \qquad k = 0, \dots, N-1 \quad ,$$

siendo este un espectro también discreto que toma valores sólo para N puntos, enumerados mediante el índice k, del eje de frecuencias. Si $S[k]$ es nulo para un determinado k, quiere decir que la señal $s[n]$ no tiene energía a esa frecuencia, no está presente. $S[k]$ está compuesto, por tanto, por una serie de N frecuencias, f_k, uniformemente repartidas, múltiplos de una frecuencia Δf que llamaremos *resolución espectral* (ver Figura 2.1). En consecuencia, el valor de la frecuencia k-ésima del espectro discreto corresponderá a una frecuencia en Hz igual a $k\Delta f$. A continuación calcularemos el valor de esta resolución espectral.

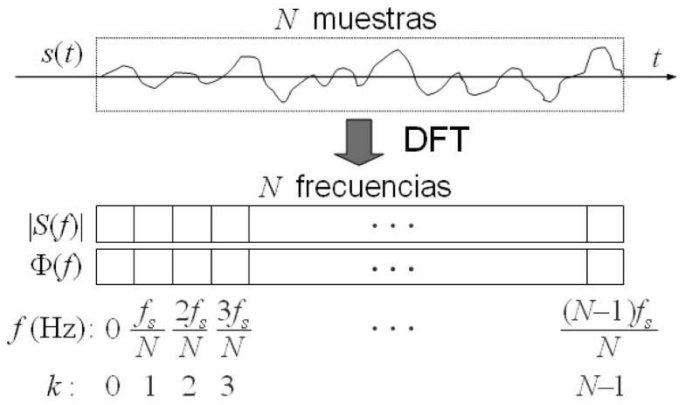


Figura 2.1. La DFT toma una señal en un vector de N muestras y produce un espectro (con amplitudes y fases) con N frecuencias (Iñesta, 2013).

Como se indica en las ecuaciones anteriores, para N muestras de la señal, la DFT produce N valores del espectro. El resultado es equivalente a un muestreo del espectro para N frecuencias entre 0 y la frecuencia de muestreo f_s. Si tenemos N valores uniformemente distribuidos en un rango de f_s Hz, es evidente que la frecuencia de la resolución espectral (la separación entre cada dos frecuencias contiguas) se calculará como $\Delta f = f_s/N$.

Caso práctico. Para una frecuencia de muestreo de 50.000 muestras por segundo, si $N = 10.000$ muestras, la resolución espectral será de $\Delta f = 5$ Hz y las frecuencias del espectro digital serán los múltiplos enteros $k\Delta f = 0, 5, 10, 15, \dots, 49.995$ Hz. Como se observa, la primera frecuencia calculada es 0 Hz. Este valor $S[0]$ representa el desplazamiento de continua de la señal equivalente al promedio de sus valores, $S[0] = \left(\frac{1}{N}\right) \sum_{n=0}^{N-1} s[n]$.

La teoría de Fourier dice que un espectro debe tener frecuencias desde $-\infty$ hasta $+\infty$, por lo que el espectro digital, limitado en principio a N valores, se replica de forma que sus valores vuelven a ser los mismos cada N frecuencias ($S[k+N] = S[k] = S[k-N]$ (ver Figura 2.2). Esto es lo mismo que decir que el espectro digital se repite cada f_s Hz: $S[f+f_s] = S[f] = S[f-f_s]$.

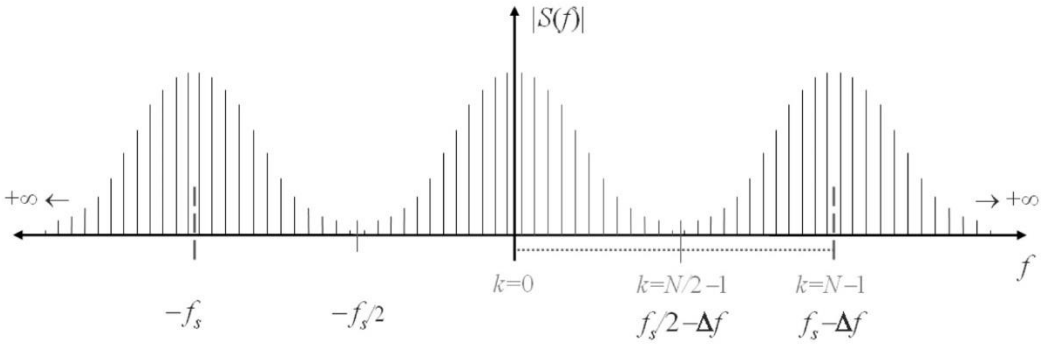


Figura 2.2. Espectro de una señal muestreada. Está discretizado en frecuencia y sus valores se repetirán cada N frecuencias tanto positivas como negativas hasta el infinito (Iñesta, 2013).

Por otra parte, la propiedad de simetría del espectro de amplitudes obliga a que $|S[k]| = |S[-k]|$. Por tanto, el espectro digital debe ser no sólo simétrico respecto al eje $f = 0$ sino respecto a $f = f_s$ y a todos sus múltiplos, tanto positivos como negativos (ver Figura 2.14).

Cuando se combinan ambas propiedades se llega a la conclusión de que la única posibilidad para que esto se cumpla es que la segunda mitad de valores de frecuencia calculados, $N/2 \leq k < N-1$, sean repeticiones de las frecuencias negativas $-f_s/2 \leq f_k < 0$ y entonces se llega a la conclusión de que las únicas frecuencias positivas útiles (por no ser réplicas de otras previamente calculadas) son las $0 \leq f_k < f_s/2$, que corresponden a la primera mitad de los valores del espectro: $0 \leq k \leq (N/2) - 1$.

De la misma forma que el caso continuo, esta operación es reversible. A partir del espectro discreto se puede generar la señal temporal discreta correspondiente. La transformada discreta de Fourier inversa (DFT^{-1}) permite el cálculo de muestras a partir de su espectro. Análogamente al caso continuo, su fórmula es

$$s[n] = \frac{1}{N}\sum_{k=0}^{N-1} S[k] e^{j\frac{2\pi kn}{N}} \quad , \quad n = 0, \dots, N-1 \quad ,$$

o bien

$$s[n] = \frac{1}{N}\sum_{k=0}^{N-1} S[k] \sin\left(\frac{2\pi kn}{N}\right) \quad , \quad n = 0, \dots, N-1 \quad .$$

A partir de un espectro de N frecuencias, la DFT^{-1} produce una secuencia de N muestras (Iñesta, 2013).

2.2. Armónicos y formantes

Cada una de las frecuencias que contiene un espectro se denomina *frecuencia parcial* o simplemente *parcial*. La amplitud para cada frecuencia indica la "cantidad de energía" que está presente en la onda analizada para esa frecuencia. De entre todas las frecuencias parciales hay una que tiene especial importancia y que recibe el nombre de *frecuencia fundamental*, f_0.

No siempre existe la fundamental, sólo en los espectros de las señales periódicas, siendo su valor $f_0 = 1/T_0$, el inverso de su periodo más largo, el *periodo fundamental*, T_0 (ver Figura 2.3). En la evolución temporal de una onda se puede identificar este T_0 midiendo el tiempo en el que se repite el ciclo de la onda.

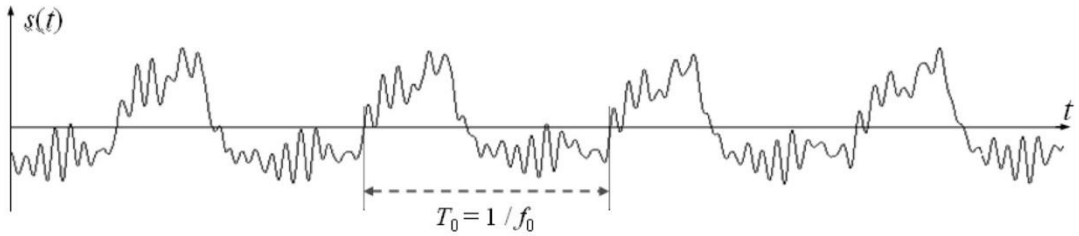


Figura 2.3. Evolución temporal de una señal periódica, con indicación de su periodo fundamental, T_0 (Iñesta, 2013).

El valor de f_0 también se puede calcular a partir de las frecuencias de los parciales del espectro como el máximo común divisor de todas ellas. No obstante, este cálculo no siempre es tan sencillo. La frecuencia fundamental es un parámetro con implicaciones en la percepción de una propiedad muy importante del sonido musical como es la altura. Es por esto que habrá que tener en cuenta una serie de consideraciones. Deberán tenerse en cuenta sólo las frecuencias parciales que podemos oír, que son aquellas que están por encima de 20 Hz y por debajo de 20 kHz. En realidad, normalmente es posible encontrar un valor de frecuencia que sea el máximo común divisor de los parciales, pero puede que sea tan pequeño que sea inaudible.

En otras ocasiones ocurre que la frecuencia fundamental se percibe sin estar presente, lo cual se conoce como fenómeno de fundamental ausente. Evidentemente, el cálculo mediante el máximo común divisor de los parciales no garantiza que la f_0 sea una de las frecuencias de los parciales. Por ejemplo, si las frecuencias parciales son 200, 300, 400 y 500 Hz, evidentemente el máximo común

divisor es 100 Hz, que no es una de esas cuatro frecuencias, pero que sí será la frecuencia fundamental y como tal se percibirá. La explicación a este fenómeno es que el oído es capaz de reconstruir la frecuencia fundamental percibiendo las diferencias en frecuencia entre los armónicos, de forma que si un tono está compuesto de armónicos separados 100 Hz, aunque esa frecuencia no esté presente, nuestro oído la percibirá por la interferencia que se produce entre esos parciales vecinos. La otra explicación se puede buscar en el dominio del tiempo. En términos de frecuencia, dada una onda compuesta de frecuencias f y $2/3f$, tendremos $f_0 = 1/3f$ y los parciales estarán relacionados con la fundamental con $2f_0$ y $3f_0$ (Figura 2.4).

El espectro de una onda ideal, con principio y fin, presenta amplitudes en todas las frecuencias de su ancho de banda (espectro continuo) debido a que la zona en la que se analiza la onda es finita en el tiempo. Para una señal ideal infinita aparecerían sólo amplitudes distintas de cero en determinadas frecuencias (ver Figura 2.5).

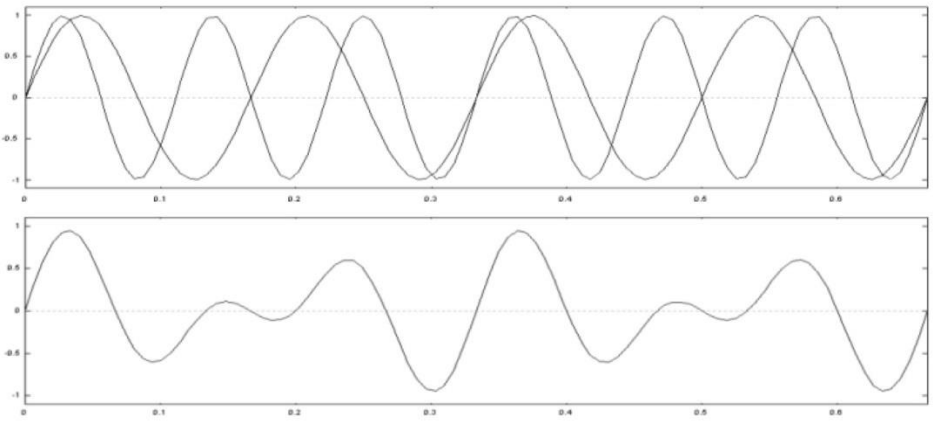


Figura 2.4. Ondas con periodos T y 1.5 T (arriba). La suma de esas ondas da como resultado otra onda periódica con periodo fundamental 3 T (abajo) (Iñesta, 2013).

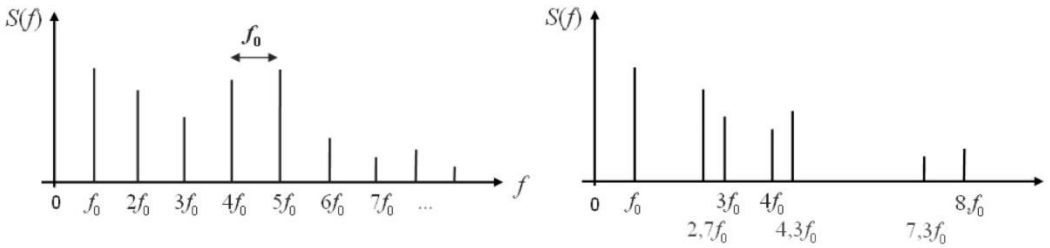


Figura 2.5. Espectros armónicos ideales de señales infinitas: espectro armónico (izq.); espectro inarmónico (dch.) (Iñesta, 2013).

Una categoría muy importante de espectros son los espectros *armónicos*, en los que todos los parciales son múltiplos enteros del valor de la frecuencia fundamental (como el de la Figura 2.5-

izq.): f_0, $2f_0$, $3f_0$, $4f_0$, etc. Como se observa, la f_0 se corresponde también con la diferencia entre las frecuencias de parciales vecinos. Estos espectros corresponden a ondas periódicas cuyo período es, por definición, $1/f_0 = T_0$.

En caso de que haya frecuencias en el espectro que no sean múltiplos enteros de f_0 se dice que es *inarmónico* (Figura 2.5-dch.). Los instrumentos que emiten sonidos inarmónicos son los de percusión, ya sean afinados (campanas tubulares, vibráfonos, etc.) o no (bombos, platillos, etc.).

La principal diferencia que existe entre los sonidos armónicos y los inarmónicos desde el punto de vista perceptual es que, mientras que en los segundos percibimos los parciales de forma individual e independiente, en los primeros oímos sólo la fundamental, mientras que los armónicos pasan a formar parte del "colorido" del sonido. Este fenómeno se denomina *fusión espectral* y es crucial para la percepción de la música.

La mayor parte de los instrumentos musicales emiten sonidos con espectros armónicos. Son aquellos que están afinados, en los que se percibe una altura definida. En la figura 2.6 se observa un ejemplo del espectro del sonido de una flauta emitiendo una nota SOL a 392 Hz. Como es un sonido real, no infinito, presenta amplitudes para todas las frecuencias analizadas. En estos casos, se considera que los parciales son los máximos locales del espectro (los picos suficientemente grandes). Se observa también como los parciales ocurren a frecuencias que son múltiplos enteros de la f_0 y, en efecto, el sonido que produce este instrumento es armónico.

La definición de armonicidad que se ha dado es matemática, pero en la práctica los sonidos no son estrictamente armónicos o inarmónicos, sino que lo son en mayor o menor grado. Podemos expresar este grado de armonicidad mediante la fracción del número de parciales que son armónicos (una vez determinada la f_0 del espectro) partido por el total de parciales. Para un sonido armónico puro esta relación valdrá 1 y para un inarmónico puro valdrá 0.

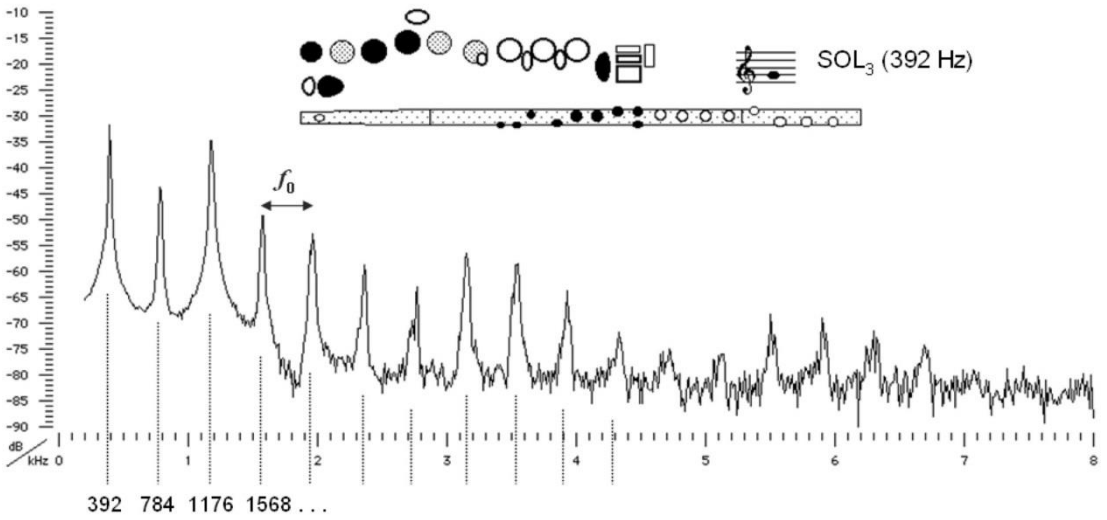


Figura 2.6. Espectro del sonido emitido por una flauta para una nota SOL a 392 Hz. Se indican las posiciones de los parciales y las frecuencias de los cuatro primeros (Iñesta, 2013).

De todas maneras, en los sonidos reales no existen ni la armonicidad pura (siempre se producen pequeñas desviaciones en las frecuencias) ni la inarmonicidad pura (sólo posible matemáticamente si hay relaciones irracionales entre los parciales) (Iñesta, 2013). Es más, un sonido musical real, en mayor o menor grado, tiene una componente ruidosa característica del instrumento que produce el sonido, que en general es inarmónica y no aporta, por tanto, información a la hora de determinar la altura.

Los primeros armónicos de un tono dotado de armonicidad forman ciertas relaciones interválicas con la frecuencia fundamental, tal y como se muestra en la Tabla 2.1 y la Figura 2.7 para la nota LA (ver Capítulo 3 para aclarar los conceptos musicales). La colección de frecuencias armónicas de un tono complejo se denomina *serie armónica* (Gómez, 2006a).

Armónico	**Frecuencia**	**Intervalo aproximado con f_0**	**Nota del conjunto cromático**
1	f_0	unísono	LA
2	$2 \cdot f_0$	octava	LA
3	$3 \cdot f_0$	octava + 5ª	MI
4	$4 \cdot f_0$	2 octavas	LA
5	$5 \cdot f_0$	2 octavas + 3ª mayor	DO♯

6	$6 \cdot f_0$	2 octavas + 5ª	MI
7	$7 \cdot f_0$	2 octavas + 7ª	SOL
8	$8 \cdot f_0$	3 octavas	LA

Tabla 2.1. Intervalos entre los primeros 8 armónicos de un tono complejo y su frecuencia fundamental f_0. Ejemplo para los armónicos de LA.

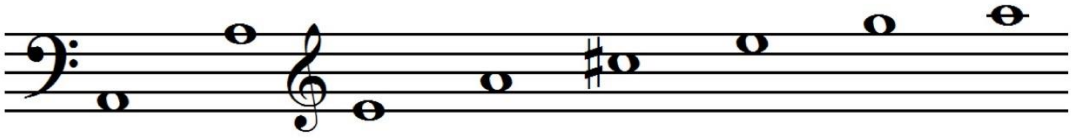


Figura 2.7. Serie armónica de LA_2 (Gómez, 2006a).

De especial importancia son los *formantes*, que son las zonas del espectro en las que las frecuencias tienen mayor amplitud (Figura 2.8). Son muy importantes para la caracterización del timbre. También juegan un papel clave en la síntesis y reconocimiento del habla, principalmente para las vocales.

Un formante es una zona del espectro, por lo que se caracteriza mediante dos parámetros: su frecuencia central y su ancho de banda. El formante con la frecuencia más baja se designa F_1, el segundo F_2, el tercero F_3, etc.

Las posiciones de los formantes son independientes de la frecuencia fundamental. Por tanto, son las mismas para voces masculinas y femeninas (gracias a eso los hablantes de ambos sexos pronuncian las mismas vocales). Si la f_0 es mayor que la frecuencia de algún formante, éste se pierde. Por esta razón, el canto de una soprano suele ser más difícil de entender.

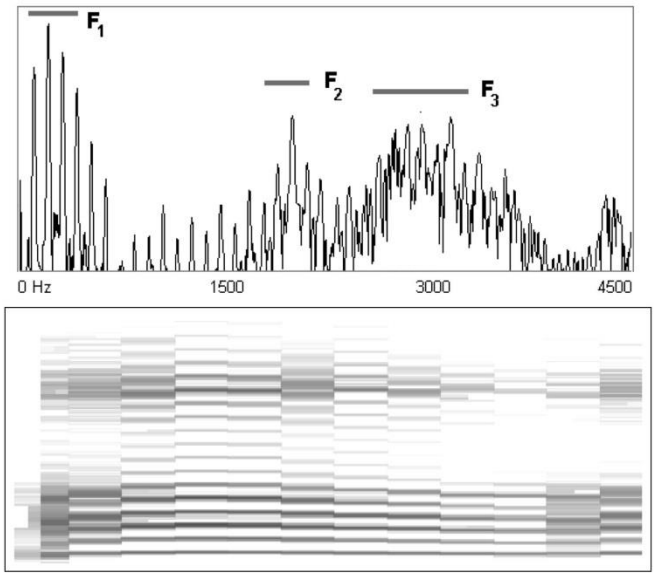


Figura 2.8. Arriba: formantes del espectro de una señal real, correspondiente al sonido de la vocal 'i'. Se señalan 3 formantes, F_1, F_2, F_3, en las bandas de frecuencias que se observan. Abajo: espectrograma de un sonido de la vocal 'a' en el que se observan energías de dos bandas del espectro (ver sección 2.3) (Iñesta, 2013).

2.3. Evolución temporal del espectro

Basta con fijarse en los límites de la integral de la FT para comprender que Fourier considera un intervalo de tiempo inenarrable y poco práctico. Las señales musicales tienen una duración mucho más limitada. Es decir, la FT calcula el espectro de una señal teórica infinita y estacionaria, pero en la práctica los sonidos son finitos y cambiantes. Normalmente necesitamos analizarlas en determinados momentos. La manera de conseguirlo sin modificar la teoría es usar una función "ventana" $w(t)$ que será nula en todo momento excepto en las inmediaciones del instante τ que se quiere estudiar. La multiplicación de esta función por la señal original resultará en el fragmento de señal a transformar, mediante una nueva versión de la FT llamada *transformada de Fourier a corto plazo* (*short-time* FT, STFT), en este caso dependiente del tiempo (τ) (ver Figura 2.9). Se emplea la letra τ para diferenciar la evolución temporal del espectro, que constará como variable independiente y eje de abcisas en la correspondiente representación, del tiempo t utilizado para transformar la señal al dominio frecuencial.

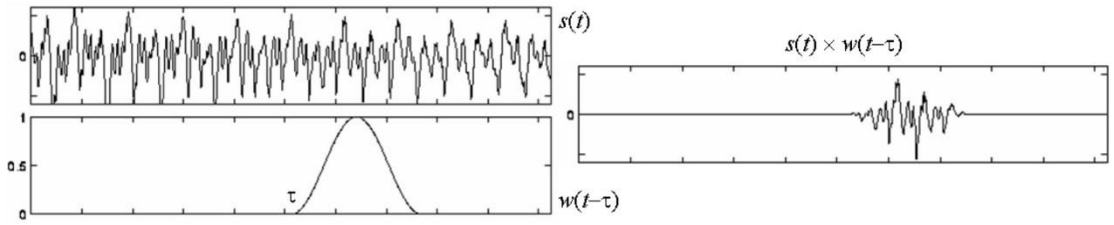


Figura 2.9. Ejemplo de cómo se aplica cada vez la ventana $w(t)$ a la señal $s(t)$ en el instante τ para realizar la transformada de Fourier a corto plazo (STFT) (Iñesta, 2013).

La expresión de la STFT es, por tanto,

$$S(f,\tau) = \int_{-\infty}^{\infty} s(t)\, w(t-\tau) e^{-j2\pi f t} dt \ .$$

Puede considerarse como el espectro de la señal $s(t)w(t-\tau)$, que se calculará para cada τ considerado. Todo esto queda ilustrado en la Figura 2.10, donde se observa el aspecto de una ventana y cómo se multiplica por la señal para generar la nueva señal de la que se calcula el espectro.

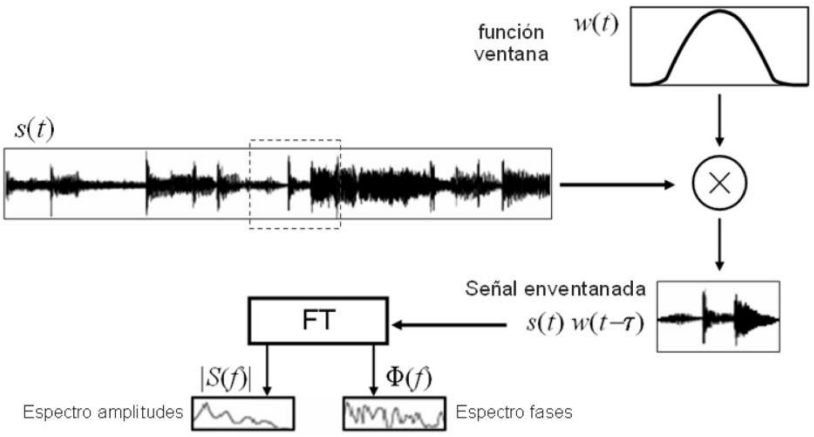


Figura 2.10. Diagrama de bloques de la STFT. De la señal $s(t)$ se selecciona una ventana y se calcula la FT de esa parte de ella. Del avance de la ventana, τ, van saliendo los diferentes espectros que conforman el espectrograma (Iñesta, 2013).

El avance de la ventana a lo largo de la señal, con saltos de magnitud τ segundos, nos permite obtener la evolución del espectro de la señal como una sucesión de "fotografías" espectrales hechas a una porción de la señal a incrementos fijos de tiempo: $\tau, 2\tau, 3\tau, \ldots$ Siguiendo esta analogía, podríamos considerar que la FT es como una fotografía en la que el obturador está abierto durante mucho tiempo (es decir, el tiempo de exposición es muy grande y cualquier cambio en los objetos de la escena producirá borrones en la información que se quería capturar), mientras que la STFT es como una película construida como la sucesión de fotografías individuales tomadas cada poco tiempo.

Finalmente podemos aplicar las mismas consideraciones que en el caso continuo para una evolución temporal del espectro de una señal discreta. La DFT es una representación de $s[n]$ para todos los valores de la muestra, pero no nos da una visión *local* de la evolución del sonido. Para ello se introduce, igual que en el caso continuo, la *transformada de Fourier discreta a corto plazo* (DSTFT) que es equivalente a aplicar la DFT sobre sucesivas ventanas $w[n]$ de ancho N muestras, superpuestas sobre la señal. Para considerar las diferentes posiciones posibles de la ventana, definimos el tiempo propio de la ventana como $w[rI]$, donde r es el número entero de ventana e I el tamaño del avance (medido ahora en muestras) de una ventana a la siguiente, por lo que rI es la posición en el tiempo de la ventana.

La expresión de la versión discreta de la STFT es análoga a la continua:

$$S[k,r] = \sum_{n=0}^{N-1} s[n]w[n-rI]e^{-j\frac{2\pi kn}{N}} \quad , \qquad k = 0, \dots, N-1 \quad .$$

Para cada posición r_0 de la ventana, $S[k,r_0]$ puede considerarse como el espectro de la señal enventanada $s[n]w[n-r_0I]$. En la figura 2.11 se observa cómo la ventana en su desplazamiento va "mostrando" las distintas zonas de la señal original sobre las que se calculará el espectro para ver su evolución.

La inversa de esta transformada recupera la señal en la zona cubierta por la ventana, $s[n,r_0I]$. Sólo se distingue de la directa en el signo menos de la exponencial, el factor de normalización $1/N$ y el subíndice k que debe haber en el sumatorio en este caso, porque lo que se va a hacer es sumar para todas las frecuencias.

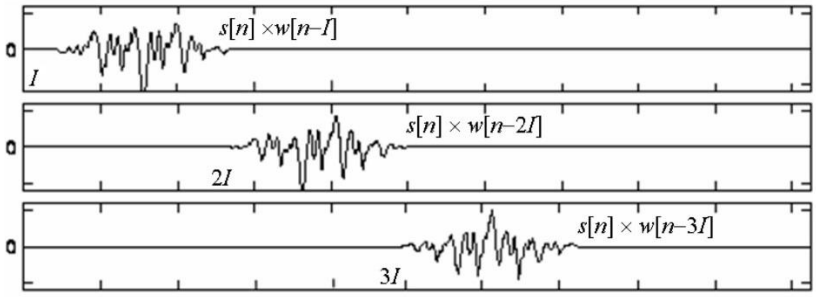
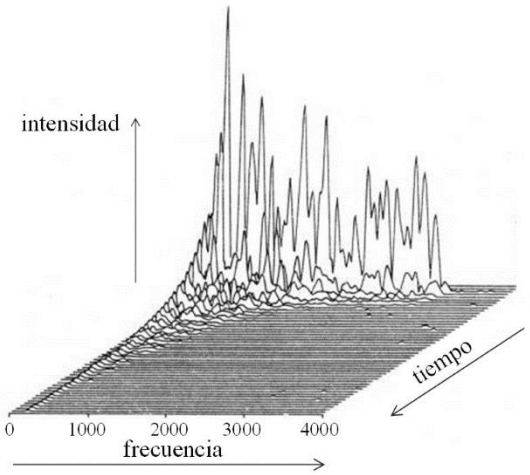


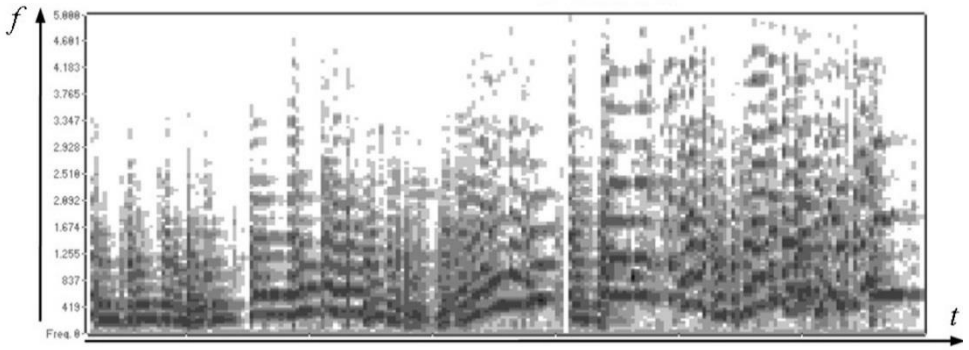
Figura 2.11. La ventana $w[n-rI]$ va avanzando sobre la señal $s[n]$ para distintos valores de r. De la señal de arriba se obtiene $S[k,1]$, de la de en medio se obtiene $S[k,2]$ y de la de abajo se obtiene $S[k,3]$ (Iñesta, 2013).

La sucesión en el tiempo de espectros locales calculados mediante la STFT forma diagramas de la evolución temporal del espectro a lo largo de la duración de la señal. Estos diagramas reciben el nombre de *espectrogramas*. Estos pueden ser tridimensionales, como el mostrado en la figura 2.12.a,

o planos (ver Figura 2.12.b), lo cual es más habitual, siendo el eje X el tiempo, el eje Y la frecuencia y representando las amplitudes por niveles de gris. De esta forma, una franja oscura muestra la presencia de energía a determinada frecuencia. Cuanto más oscura sea la franja mayor es la amplitud. Cuanto más arriba esté esa franja mayor es la frecuencia a la que corresponde.



(a) Espectrograma 3D



(b) Espectrograma 2D

Figura 2.12. Tipo de espectrogramas de señales musicales: (a) Representación 3D en la que las intensidades se representan en el eje vertical. En este caso se observa el espectrograma de un sonido inarmónico con muchos parciales al inicio del sonido pero que se extinguen rápidamente hasta sólo quedar uno a la izquierda que persiste hasta el final del análisis. (b) Representación plana. En este caso la intensidad se representa mediante el nivel de gris, más oscuro cuanto mayor es la amplitud. En este caso el sonido analizado es armónico, pues se aprecia un patrón de bandas en el eje vertical separadas aproximadamente el mismo valor de frecuencias, lo cual es característico de los espectros armónicos, por definición (Iñesta, 2013).

Como ilustración de la información musical que se puede encontrar en el espectrograma, en la figura 2.13 se muestra el análisis del sonido generado al sintetizar con sonidos sinusoidales (un solo parcial en cada momento) una partitura. Como se puede observar, cada nota deja una traza en el espectrograma, las duraciones se pueden seguir en el eje horizontal y las alturas quedan reflejadas en el eje vertical.

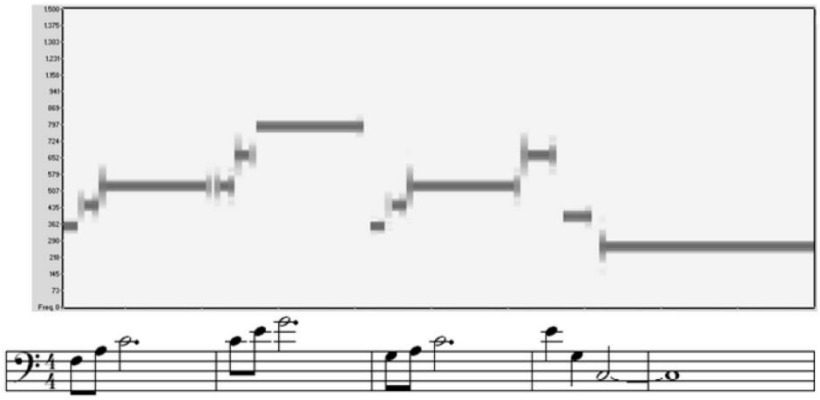


Figura 2.13. El espectrograma de una melodía sintetizada mediante ondas sinusoidales revela información musical sobre las notas que la componen (Iñesta, 2013).

2.4. Ventanas

Como se ha dicho anteriormente, una ventana es simplemente una función que sólo es distinta de cero en un rango limitado de tiempo, normalmente de decenas de milisegundos. Hay muchos tipos de ventanas y no hay una que sea la mejor, pues cada una tiene ventajas e inconvenientes. Se escoge una concreta en función de la aplicación que se persigue. Puede considerarse que hay dos criterios independientes a la hora de escoger una ventana: su longitud y su forma.

2.4.1. Longitud de la ventana

La longitud de la ventana de análisis, cuando trabajamos con señales discretas, se da en número de muestras y condiciona la resolución que se puede conseguir en el cálculo de las frecuencias. Normalmente esta longitud es una potencia de 2 por razones de eficiencia en el cálculo de la DFT.

La duración del fragmento de señal que selecciona una determinada ventana de N muestras se puede calcular fácilmente, conocida la frecuencia de muestreo f_s a la que ha sido digitalizada la señal. Su inversa es el periodo de muestreo $1/f_s = T_s$, que informa del tiempo que transcurre entre dos muestras consecutivas. Por lo tanto, una ventana de N muestras tendrá una duración de NT_s segundos.

Hemos visto en la sección 2.1 que las frecuencias de un espectro discreto son múltiplos enteros de $\Delta f = f_s/N$. Por tanto, el tamaño de la ventana condiciona la resolución frecuencial del espectro, de

manera que, una vez establecida la frecuencia de muestreo, a mayor longitud de la ventana, calculamos más frecuencias y por tanto tendremos mayor precisión en el espectro ($\Delta f = f_s/N$ será menor) y viceversa, una ventana pequeña permite calcular menos frecuencias y tendremos menos precisión (Δf será mayor).

Caso práctico. Si $f_s = 20$ kHz y la longitud de la ventana es de $N = 512$ muestras, la resolución espectral será de $\Delta f = 20000/512 = 39$ Hz. Con estos datos también podemos calcular la duración de la ventana: NT_s segundos, que nos da $512 \times (1/20000) = 512 \times 0{,}00005 = 25{,}6$ ms.

La ventana se va desplazando por la señal mediante saltos de I muestras en el tiempo que podemos interpretar como la *resolución temporal*; es decir, cada cuántos segundos analizamos el sonido. Este valor es $\Delta t = NT_s = N/f_s$, que es la inversa de la resolución en frecuencia del espectro $\Delta t = 1/\Delta f$. Este hecho lleva asociado una importante consecuencia: no podemos conocer de manera arbitrariamente precisa lo que sucede en el tiempo y en la frecuencia; más aún, si aumentamos la resolución temporal disminuimos la resolución frecuencial y viceversa. Debemos elegir entre conocer bien qué frecuencias están presentes en el sonido y saber cuándo se han producido dichas frecuencias. En la práctica se suele usar una solución de compromiso que permita conocer suficientemente ambas cosas.

Caso práctico. Si $f_s = 44100$ Hz y tomamos una ventana de $N = 128$ muestras, obtendremos una resolución espectral de $\Delta f = 44100/128 = 344{,}5$ Hz y una resolución temporal de $\Delta t = 1/f_s = 2{,}9$ ms. Esta Δt es muy precisa porque $2{,}9$ ms es muy poco tiempo para la música, pero en cambio la Δf es inaceptablemente pobre; saber qué ocurre sólo cada 344 Hz no sirve para nada en música. Por el contrario, si nuestra ventana es de $N = 4096$ muestras, su duración será ahora de $\Delta t = 92$ ms, que es ahora bastante tiempo (casi una décima de segundo) pero la resolución frecuencial será $\Delta f = 10{,}8$ Hz, que es muy fina. El resultado de ambos casos se muestra en la figura 2.14.

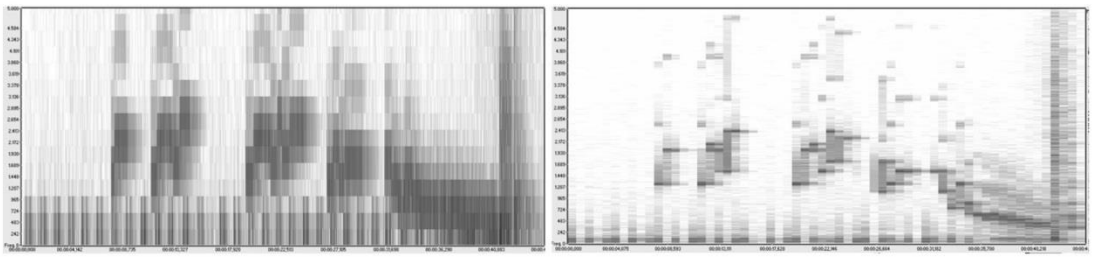


Figura 2.14. Dos espectrogramas de un mismo sonido digitalizado a una frecuencia de 44100 Hz pero con tamaños de ventana diferentes: $N = 128$ muestras (izq.), $N = 4096$ muestras (dch.) (Iñesta, 2013).

Para paliar en parte este problema se puede usar un desplazamiento menor que nos permita analizar la señal con una resolución más fina al hacer que los saltos en las posiciones de la ventana sean menores, sin empeorar Δf. Hasta ahora siempre hemos considerado que el desplazamiento de la ventana es igual a su tamaño ($I = N$). Ahora planteamos la posibilidad de que $I < N$. En vez de utilizar un desplazamiento definido en términos absolutos, se suele hacer relativo al tamaño de la ventana, especificándolo como un porcentaje de *solapamiento*, S, definido como el porcentaje de N que se avanza (ver Figura 2.15).

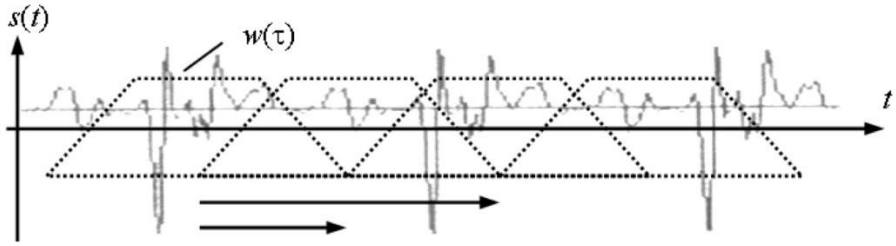


Figura 2.15. Avance de una ventana con un solapamiento del 50%. Se muestra el avance que tendría la ventana si no hubiera solapamiento (flecha larga) y el que hay con este solapamiento (flecha corta) (Iñesta, 2013).

Caso práctico. Si tenemos una ventana de $N = 1024$ muestras y consideramos un desplazamiento con solapamiento $S = 75\%$, el desplazamiento absoluto será de $I = 0{,}25 \times N = 256$ muestras, pues si las ventanas en las posiciones r_0 y $r_0 + 1$ se superponen un 75%, el desplazamiento de la ventana será de un 25% de su tamaño.

En general, para calcular la resolución temporal en esta nueva situación (menores saltos cuanto mayor sea el solapamiento), basta con multiplicar el cálculo de la resolución sin solapar por $1 - S/100$:

$$\Delta t = \frac{N(1 - S/100)}{f_s} \quad .$$

Otro motivo para considerar un desplazamiento con solapamiento es que, para algunas funciones de ventana que tienen valores muy bajos o nulos cerca de los extremos (ver la sección siguiente), no conviene que el salto sea del mismo tamaño que la ventana, pues perderemos o alteraremos la información que haya en sus extremos.

Existe otro problema, y es que si en la ventana no cabe un número entero de periodos de la onda aparecerán amplitudes extrañas (véanse los principios y finales de las notas en el espectrograma de la figura 2.13). Esto sólo podría arreglarse en caso de tener ondas estáticas, pero para eso no haría falta la STFT, bastaría con la FT.

2.4.2. Forma de la ventana

La forma de la ventana afecta al cálculo del espectro. La forma de ventana más tosca es simplemente seleccionar una parte de la señal $s[n]$ de N muestras, dejando el resto a cero. Esto es una ventana rectangular:

$$w[m] = \begin{cases} 1 & si \quad 0 \leq m \leq N-1 \\ 0 & para\ el\ resto \end{cases}$$

Así la señal se multiplica por 1 en ese intervalo (permanece igual) y por cero fuera de él (se anula).

Los extremos abruptos de esta ventana introducen saltos bruscos a cero en la señal (véase Figura 2.16(izq.-arriba)) que generan valores espectrales que no estaban presentes en la señal original, frecuencias ficticias denominadas *artefactos*. Obsérvese este hecho en la figura 2.16(dch.) en la que se muestra la transformada de Fourier de esta ventana (su espectro, en definitiva). El mismo efecto que se observa es el que produce esta ventana cuando se usa para obtener la STFT de una onda sinusoidal de frecuencia f, sólo que el lóbulo central aparecerá centrado en esa frecuencia, en vez de en 0 como en la figura. En el análisis de ondas complejas, aparecerá este patrón alrededor de cada parcial que componga la onda.

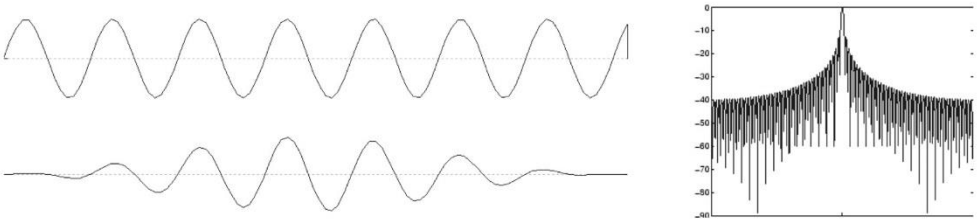


Figura 2.16. Onda sinusoidal enventanada con una ventana rectangular (izq.-arriba); se observa el salto brusco en la señal al final de la ventana por la derecha. La misma onda enventanada con una función que decae suavemente en los extremos (izq.-abajo). Espectro de la ventana cuadrada (dch.) Iñesta (2013).

Este problema no tiene solución, puesto que la naturaleza estática de la función seno quedará en cualquier caso alterada al imponer una ventana de duración finita. Podemos paliar este problema aplicando una función de ventana que escale de forma suave la amplitud de la señal hasta que se haga

0 en los extremos de la ventana. Esto no evita la distorsión de la señal al ser enventanada, pero elimina la discontinuidad[12] (véase Figura 2.16(izq.-abajo)).

En la figura 2.17 se muestran comparativamente las formas de algunas de las ventanas más utilizadas. Ninguna elimina completamente los artefactos del espectro, pero sí que los pueden reducir sustancialmente, dependiendo del tipo de ventana que utilicemos. En cualquier caso, todas las ventanas que se utilizan son simétricas respecto a su punto medio y sus valores se calculan mediante fórmulas conocidas en el intervalo $[0, N-1]$.

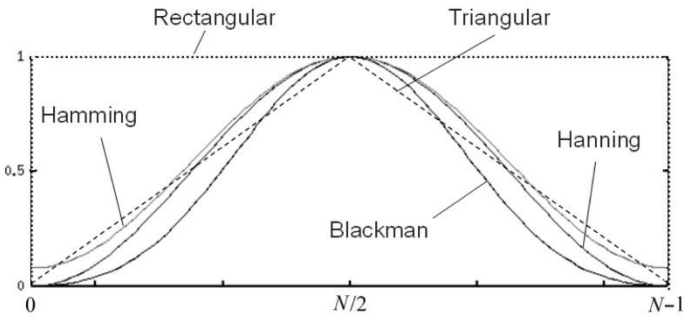


Figura 2.17. Diferentes formas comparadas de funciones ventana (Iñesta, 2013).

Cada una de ellas tiene un distinto comportamiento en cuanto a la anchura del lóbulo central y a la diferencia de amplitud de los lóbulos laterales. Se observa que sus espectros presentan un lóbulo central grande y unos lóbulos laterales más pequeños. Las propiedades más importantes de una ventana son el ancho del lóbulo central (medido en unidades de frecuencia y definido como la distancia entre los primeros cruces con cero laterales de la FT de la ventana) y el nivel del mayor lóbulo lateral (medido en dB con relación al nivel del lóbulo central). Estas magnitudes dependen de la forma y de la anchura de la ventana. En el caso de la ventana rectangular, el ancho del lóbulo central se calcula como $4\pi/N$. El primer lóbulo lateral está 13 dB por debajo del central y el resto de lóbulos van decayendo unos -6 dB por cada octava.

No existe una ventana que sea mejor que las demás en términos absolutos. La elección de una u otra depende de la aplicación o del tipo de señales que tengamos. Si la señal de entrada está compuesta de tonos más o menos simples, conviene tener una ventana con un lóbulo central estrecho para que localice bien las frecuencias presentes, como la rectangular. En cambio, para señales ruidosas en las

[12] Un salto brusco en la forma de onda tiene implicaciones importantes en el espectro. Una variación suave en el dominio del tiempo se traduce en baja frecuencia, mientras que una variación rápida implica alta frecuencia. Una discontinuidad hacia cero es el caso extremo, la frecuencia de este trozo de señal tenderá a infinito. En el dominio digital se procura evitar este tipo de situaciones porque provocan distorsiones muy graves, como se ha comentado, pues pueden violar el rango $[0, f_s/2]$ y por tanto introducir solapamiento espectral (*aliasing*).

que la energía no está bien localizada a frecuencias concretas, una ventana con el lóbulo central más ancho, como la Hamming, funcionará mejor (Iñesta, 2013).

2.5. Cromagrama

Existe un refinamiento del espectrograma, que ya hemos visto que es capaz de representar la evolución temporal del espectro, para ajustar todavía más su aplicación al análisis de señales musicales (McVicar, et al., 2014).

El hecho de que una misma altura y su octava sean percibidas como la misma nota lleva a pensar en dos dimensiones: altura absoluta y *croma* (Shepard, 1964) (ver sección 3.6).

Fujishima (1999, 2000) obtuvo un vector de características de croma partiendo del cálculo de la DFT de un fragmento de audio de entrada y de la evolución de la energía en un conjunto de bandas de frecuencia, cuyas frecuencias centrales estaban muy relacionadas con las clases de alturas[13] (*pitch classes*) (C, C♯, …, B). Finalmente realizaba un plegado de todas las bandas de frecuencia que correspondían a la misma clase de altura (independencia de la octava o altura absoluta), obteniendo así un vector multidimensional para cada instante de tiempo: un cromagrama (*chromagram*), que describe cómo la prominencia o energía asociada a estas clases de alturas varía con el tiempo, y es la más extendida de las representaciones usadas por los sistemas modernos de ACE (Wakefield, 1999).

Puede ser representado por una matriz $\boldsymbol{X}$ de elementos reales, que contiene una fila para cada *pitch class* considerada, y una columna para cada muestra de tiempo considerada (Figura 2.24). Un vector que contiene la prominencia de cada clase de altura en un instante dado, que se corresponde con una columna $\boldsymbol{x}$ de $\boldsymbol{X}$, se denomina vector de croma (*chroma vector* o *chroma feature*).

[13] La inmensa mayoría de la literatura sobre extracción de información musical (altura, tonalidad, acordes…) se encuentra en inglés y en el presente trabajo se ha decidido traducir algunos de los términos para la mayor comprensión del texto en castellano. En general se procura incluir el término en inglés entre paréntesis.

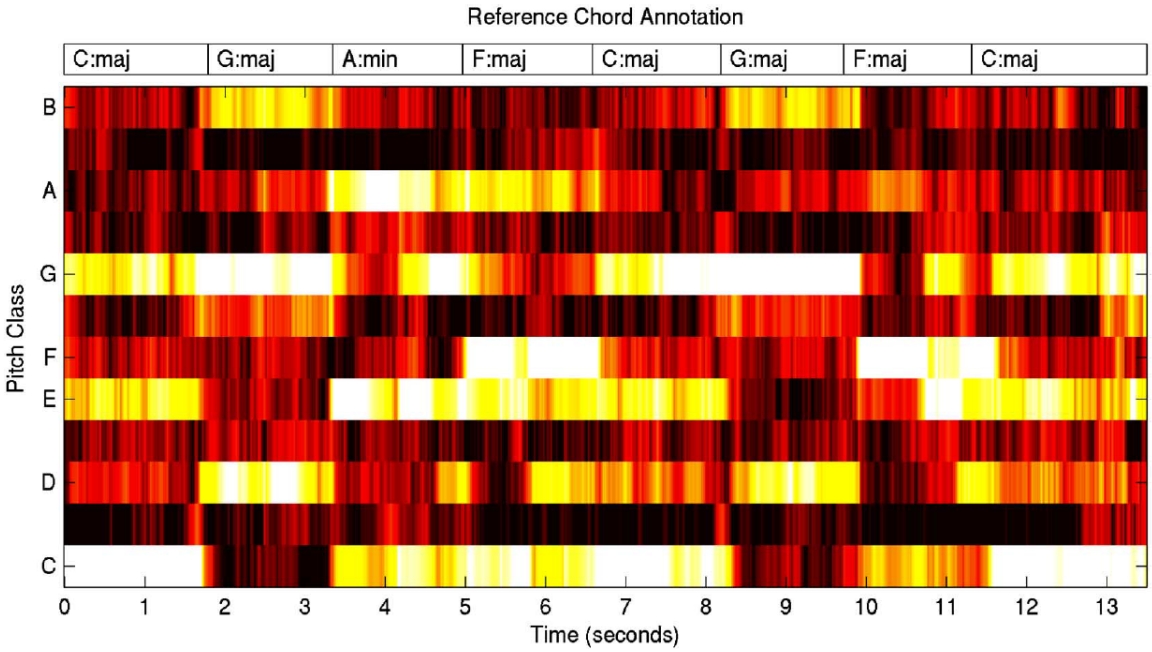


Figura 2.24. Una matriz de croma o cromagrama común, mostrando aquí la introducción de *Let it be* (Lennon/McCartney) por medio de la intensidad (prominencia) de la clase de altura p en el instante t (McVicar, et al., 2014).

2.6. Transformada Q

Una de las limitaciones de la STFT es que emplea ventanas de longitud fija. Fijar este parámetro implica alcanzar un compromiso entre la resolución frecuencial y la temporal, como comentábamos: si las ventanas son pequeñas, las frecuencias con longitud de onda grande no pueden ser apreciadas, mientras que con una ventana grande se obtiene una resolución temporal pobre. Para los objetivos de la estimación automática de acordes la resolución frecuencial viene prefijada de partida, puesto que han de distinguirse frecuencias hasta la mitad de un semitono (ver sección 3.1), lo cual supone un límite inherente en la resolución temporal. Esta resolución será particularmente pobre si se desea capturar información de baja frecuencia con la resolución frecuencial impuesta, es decir, que la elección del rango de frecuencia en el que calcular la transformada es una importante decisión de diseño (aunque los sistemas que utilizan ponderación A son menos sensibles a este problema, ya que las frecuencias fuera del rango de sensibilidad óptima del ser humano verán mermada su influencia; McVicar, et al., 2014).

Una alternativa a la STFT que resuelve parcialmente este problema, haciendo uso de ventanas de longitud dependiente con la frecuencia, es la transformada de Q constante o *transformada Q*, usada por primera vez en un contexto musical por Brown (1991). Esta transformada está estrechamente relacionada con la FT. Al igual que la FT, la transformada Q es un banco de filtros, pero a diferencia de la anterior ésta tiene frecuencia centrales geométricamente espaciadas $f_k = f_0 \cdot 2^{\frac{k}{b}}$ $(k = 0, \dots)$,

donde b establece el número de filtros por octava. Para encadenar todos los filtros ha de escogerse el ancho de banda del filtro k-ésimo tal que $Q = \frac{f_k}{\Delta_k^{cq}} = (2^{\frac{1}{b}} - 1)$. Lo que hace tan útil a la transformada de Q constante es que, escogiendo una apropiada f_0 (frecuencia central mínima) y b, las frecuencias centrales se corresponden directamente con las notas musicales. Por ejemplo, al escoger $b = 12$ y f_0 tal que equivalga a la nota MIDI 0 se consigue hacer corresponder el coeficiente k-ésimo de índice cq con la nota MIDI número k (Blankertz, s.f.).

Otra característica interesante de la transformada de Q constante es que su resolución temporal incrementa con la frecuencia. Una situación similar se da en nuestro sistema auditivo; al igual que el oído humano, un sistema digital necesita más tiempo para percibir la frecuencia de un tono grave. Esto se suele relacionar con que, generalmente, la tasa de cambio de nota en los registros graves es menor.

Nos remitimos al Capítulo 4 de este trabajo para una explicación más en profundidad acerca del cálculo de la transformada Q de una señal sonora (Gómez, 2006a). En Blankertz (s.f.) se propone una implementación en lenguaje MATLAB empleando ventanas Hamming al igual que Brown (Brown, 1991; Brown y Puckette, 1992).

Capítulo 3

Fundamentos de música y psicoacústica

Los materiales de la música popular actual deben más a la música culta occidental que a cualquier otra. Ésta ha establecido de qué elementos está hecha la música y cómo están unidos entre sí. La música trata con cantidades precisas, entidades discretas: las notas musicales tienen frecuencias fundamentales que las definen y en una composición musical se especifican con precisión sus características de duración, intensidad, etc.

En el marco de la música occidental puede hablarse de dos dimensiones que se nutren la una de la otra: al aspecto vertical y el horizontal. El aspecto horizontal se suele asociar a cambios relativamente rápidos en la altura de los sonidos que dan lugar a la melodía, y el aspecto vertical a un grupo de sonidos mantenidos en el tiempo y coincidentes que cambian con menos frecuencia, conocidos como acordes y que dan lugar a la armonía.

No está entre los objetivos del presente trabajo tratar el aspecto funcional de las progresiones de acordes, o dicho de otra forma, qué función tiene cada acorde en la progresión, por lo que no se ahondará en estas cuestiones. Si presentaremos nociones de métrica y tempo, así como de caracterización del timbre. Nos dedicaremos especialmente a los aspectos relativos a la definición de los acordes en un contexto tonal, la percepción humana de la altura y la caracterización espectral y temporal del timbre.

3.1. Escalas e intervalos

La unidad básica de la armonía es el intervalo, que se define como la distancia entre dos sonidos. Cuando dos sonidos suenan a la vez, la distancia entre ellos es un intervalo armónico. Si los dos sonidos se oyen uno tras otro, la distancia es un intervalo melódico. Existen 12 sonidos (clases de alturas) utilizados para crear tanto la melodía como la armonía, separados cada uno de ellos por un *semitono*[14]. Comúnmente se emplea un subconjunto de estos sonidos conocido como la *escala diatónica*, que consta de 7 grados o notas diferentes y tiene dos formas básicas con intervalos definidos: la escala diatónica menor y la escala diatónica mayor (Piston, 1998).

[14] Se dice que es el intervalo más pequeño de la música occidental. No obstante, la subdivisión del semitono se ha empleado musicalmente en diversos momentos de la historia, con mayor o menor éxito, y da lugar al *microtonalismo*. En este contexto y para otros fines prácticos (afinación de instrumentos, procesamiento digital de señal) se utiliza la *centésima* para subdividir el semitono, siendo 1 st = 100 cents en el sistema de afinación bien temperado.

Estas notas o grados de la escala diatónica se identifican por un número ordinal, como puede verse en la Figura 3.1. El primer grado o *tónica* da nombre a la escala. La denominación de grado en este contexto diatónico se extiende también a los acordes que pueden formarse sobre cada una de las notas, así como a cada una de sus notas constituyentes. Pero para poder hablar de acordes con rigor en primer lugar hemos de definir unos pocos conceptos.

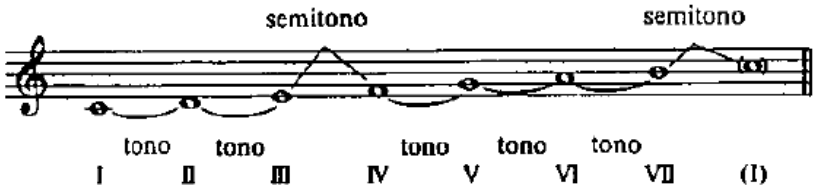


Figura 3.1. Escala diatónica mayor con tónica en Do. La posición relativa de cada nota (grado) de la escala se indica con números romanos. Las distancias (intervalos) entre las notas están predeterminadas y se dan en tonos y semitonos (Piston, 1998).

Las notas se nombran, en notación anglosajona, con las letras C, D, E, F, G, A, B (Do, Re, Mi, Fa, Sol, La, Si/Ut en la tradición latina). A estos sonidos se les conoce como *naturales*. Los sonidos restantes, o *alterados*, se obtienen modificando los naturales con los símbolos sostenido (♯) (incremento de un semitono) y bemol (♭) (decremento de un semitono), dando lugar a la *escala cromática*, que contiene los 12 sonidos: C, C♯, D, D♯, E, F, F♯, G, G♯, A, A♯, B[15] (Figura 3.2). Esta manera de organizar las clases de alturas tiene sus cimientos en la música de base diatónica: siete son los nombres de las notas naturales y siete los nombres de los grados funcionales, por el contexto diatónico precisamente (Lester, 2005). Otro símbolo sintáctico importante es el becuadro (♮), usado para anular el efecto de una alteración previa. Los nombres de las notas se repiten cíclicamente conforme aumenta la frecuencia y dan lugar a las diferentes octavas. Todos los intervalos posibles se describen en términos de tonos y semitonos. Toda nota puede expresarse con una letra y un número que indica la octava (ej. A3, C6, F♯0).

En el presente trabajo usaremos indistintamente las notaciones anglosajona y latina. Sí priorizaremos la latina a la hora de realizar la mayoría de explicaciones, pero conforme nos situemos en el contexto del sistema en MATLAB emplearemos la notación anglosajona con mayor asiduidad.

[15] Empleando bemoles se genera una lista diferente de nombres para describir los mismos sonidos o *pitch classes*: C, D♭, D, E♭, E, F, G♭, G, A♭, A, B♭, B. Hay, pues, equivalencias del tipo D♭ ≡ C♯, lo cual se conoce como *enharmonía*.

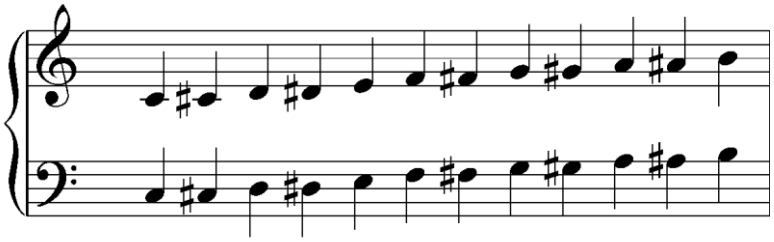


Figura 3.2. Escala cromática, [C, C♯, D, D♯, E, F, F♯, G, G♯, A, A♯, B], en clave de Sol (arriba, de C4 a B4) y en clave de Fa (abajo, de C3 a B3).

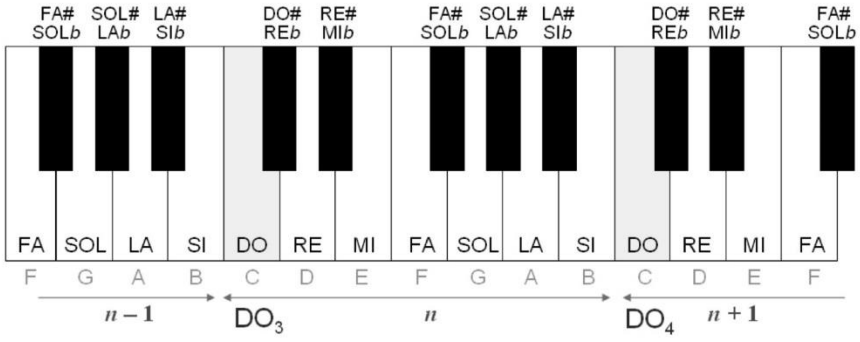


Figura 3.3. Teclado de piano en el que se observan las notas naturales (asociadas generalmente a las teclas blancas) y las notas alteradas (asociadas a las teclas negras), con sus nombres. En la parte inferior se representa el cambio del número de octava en cada Do (Iñesta, 2013).

Los intervalos, ya sean melódicos o armónicos, se expresan con dos términos: un ordinal y una cualidad. Los ordinales de los intervalos simples, que se limitan al ámbito de una octava, son: segunda, tercera, cuarta, quinta, sexta, séptima y octava, además del unísono, nombre que se le da al intervalo entre dos notas idénticas. La cualidad de éstos varía en función del intervalo. Los intervalos entre una nota dada y los grados llamados tonales (cuarta, quinta y octava) sólo pueden ser justos, aumentados o disminuidos. Los de los grados restantes, llamados modales, pueden ser mayores, menores, aumentados o disminuidos (Tabla 3.1). Ahora bien, unas denominaciones son más comunes que otras, dado que muchas son equivalentes en número de semitonos. Los intervalos más comunes se muestran en la Figura 3.4.

	DISMINUIDA	MENOR	JUSTA	MAYOR	AUMENTADA
2ª	≡ Unísono	**1 st**		**2 st**	3 st
3ª	2 st	**3 st**		**4 st**	5 st
4ª	4 st		**5 st**		**6 st**
5ª	**6 st**		**7 st**		**8 st**
6ª	7 st	**8 st**		**9 st**	**10 st**
7ª	**9 st**	**10 st**		**11 st**	12 st
8ª	11 st		**12 st**		13 st

Tabla 3.1. Tabla de intervalos en el ámbito de una octava, medidos en semitonos (st) respecto a una nota dada. Las denominaciones más comunes aparecen en negrita.

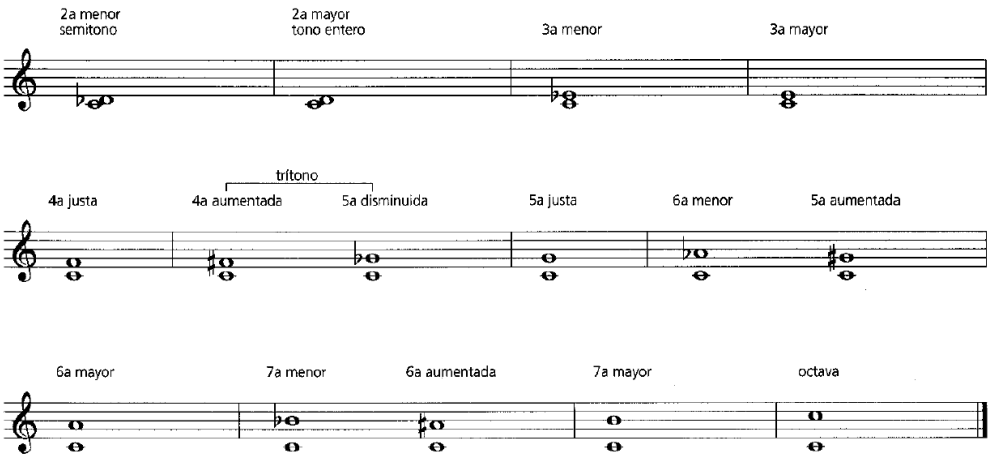


Figura 3.4. Representación musical en pentagrama de los intervalos más comunes (Levine, 2003).

El pentagrama es el medio de representación tradicional de la música escrita y consiste en cinco líneas equidistantes y sus cuatro espacios intermedios que representan las alturas de las notas. Se dan unos intervalos determinados entre las líneas y espacios del pentagrama. Independientemente de la clave que se utilice, que establece en qué posición se indexan las notas *naturales* [C, D, E, F, G, A,

B] en el pentagrama[16], hay una distancia de 1 tono entre todas las líneas-espacios salvo en el caso de E-F y B-C, donde hay tan solo 1 semitono (Figura 3.1). Esto se ve reflejado también en las teclas del piano, la alternancia tecla negra-tecla blanca se rompe en los intervalos E-F y B-C, donde se dan dos teclas blancas consecutivas (Figura 3.3). Los modos gregorianos formaban escalas empleando únicamente las teclas blancas o notas naturales, con alguna alteración casual[17]. A través de un proceso de sensibilización o alteración de alguno de sus grados se generaron algunas de las escalas mayores y menores, que más tarde se convertirían en el estándar para producir música[18] (Valero-Castells, 2012). No entraremos en mayores discusiones sobre el origen de estos materiales.

Los modos mayor y menor son el eje de la inmensa mayoría de la música que se escucha en la actualidad, ya sea procedente de la tradición culta o de los artistas de pop, rock, reggae, folk, etc. Los acordes suelen formarse por superposición de notas pertenecientes a estos modos[19], como ya se ha ido apuntando. La diferencia esencial entre el modo mayor y el modo menor natural es que en el modo menor los grados III, VI y VII están rebajados un semitono. En el caso de la escala menor armónica, tan sólo el III y el VI se encuentran rebajados, pues se busca crear una *sensible* o grado VII a distancia de semitono de la tónica, análogo al que ya se da en el modo mayor por definición (Tabla 3.2). Los músicos de estilo clásico y algunos sistemas de detección de tonalidad rastrean la partitura en busca de estas sensibles, logradas mediante la aplicación de alteraciones accidentales[20] (Figura 3.5), con el fin de determinar si un pasaje está en modo menor, puesto que la escala menor armónica está mucho más presente que la menor natural en la práctica común clásica como rasgo definitorio de la tonalidad menor.

[16] La clave de Sol (G) sitúa la nota G4 en la segunda línea y la clave de Fa (F) indica que la cuarta línea es F3, como se mostraba en la Figura.

[17] La nota Si se alteraba en ocasiones con el fin de evitar el intervalo de quinta disminuida o tritono. Tendía a prescindirse de esta nota por su sonoridad agresiva hasta que Guido d'Arezzo (991-1050) le dio el estatus de nota diatónica, si bien alterándola cuando procedía para evitar el mencionado intervalo, que el compositor austríaco Johann Joseph Fux (1660-1741) describió como *Diabolus in Musica*. En términos físicos, su armonicidad es muy baja, puesto que los espectros de ambas notas son muy distintos y sus parciales producen batidos.

[18] También se asocian los modos mayor y menor con los modos griegos jónico y eólico, respectivamente, que se forman construyendo una escala de 7 sonidos con las notas naturales partiendo de C y de A.

[19] La palabra modo en la actualidad trae consigo cierta ambigüedad, dado que el contexto es muy amplio: podemos estar hablando de canto gregoriano, de compositores innovadores como Messiaen o Persichetti, de modos de jazz o de modos griegos. En general hace referencia, como se ha visto, a un subconjunto de la escala cromática con el cual se pueden formar melodías y acordes.

[20] No indicadas al inicio de la pieza sino en el transcurso la misma, de manera puntual.

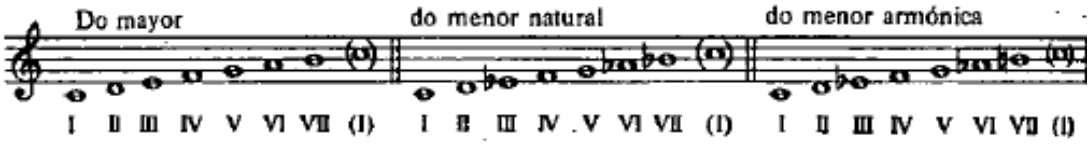


Figura 3.5. Escalas diatónicas mayor, menor natural y menor armónica de Do (Piston, 1998).

DO		+2		+2		+1		+2		+2		+2		+1	
mayor	**DO**		RE		MI	FA		SOL		LA		SI		**DO**	
Grado	I		II		III	IV		V		VI		VII		VIII	
DO		+2		+1		+2		+2		+1		+2		+2	
menor	**DO**		RE		MI♭		FA		SOL		LA♭		SI♭		**DO**
Grado	I		II		III		IV		V		VI		VII		VIII

Tabla 3.2. Notas e intervalos de las escalas diatónicas para el caso concreto de las tonalidades de DO mayor y DO menor (escala menor natural). La octava se indica para mostrar que se llega a la tónica (VIII ≡ I), no porque pertenezca dos veces a la escala (Iñesta, 2013).

3.2. Tonalidad

La música tonal, que es objeto de este estudio, se organiza en cada pieza en torno a un sonido que actúa como centro de gravedad y que recibe el nombre de tónica. Con esta nota como comienzo, se construye una escala diatónica como la que ya se ha descrito.

Este orden establecido de jerarquía sonora se conoce como tono o tonalidad. Se caracteriza por la armadura que, en esencia, es una lista de los sonidos que están alterados, y se indica al comienzo de cada sistema de pentagrama o en mitad de una pieza si se da una modulación (cambio de tonalidad). En cada una de las tonalidades se despliega, desde una tónica dada, el patrón de tonos y semitonos del modelo diatónico y se dan las mismas relaciones de alturas que en otras tonalidades (Figura 3.6).

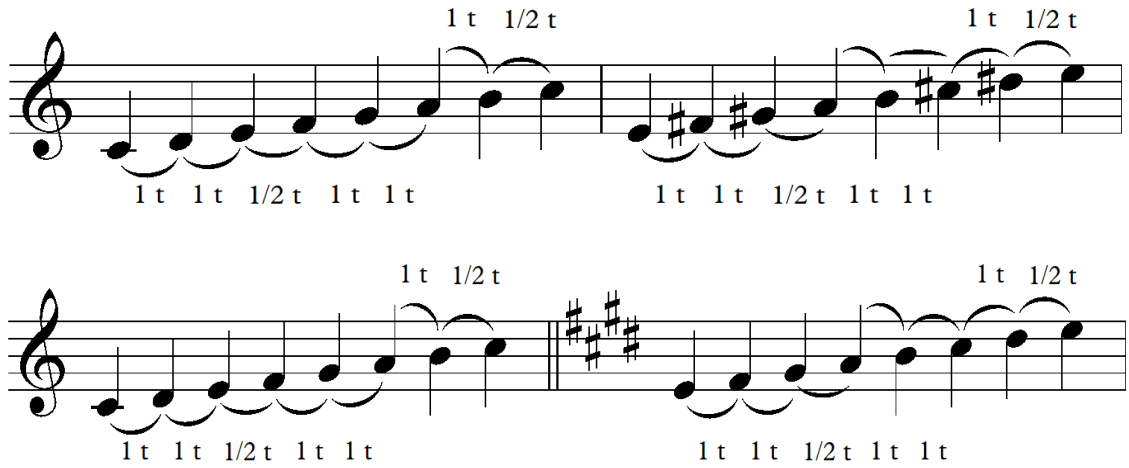
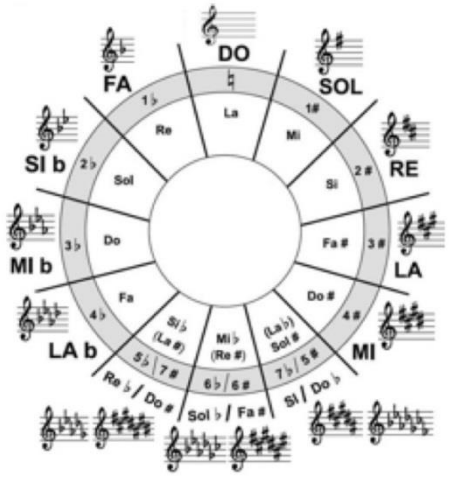


Figura 3.6. (a) Escala diatónica mayor ascendente en las tonalidades de Do Mayor (izquierda) y Mi Mayor (derecha). Nótese que la disposición de tonos y semitonos es idéntica en ambos casos. (b) Ejemplo idéntico pero empleando una armadura para indicar la tonalidad de Mi Mayor, en lugar de utilizar alteraciones accidentales para reorganizar los intervalos del pentagrama.

La manera más gráfica de entender el concepto de tonalidad la encontramos en el llamado círculo de quintas (Figura 3.7), empleado de múltiples formas prácticas en el arte musical. Una tonalidad en modo menor se entiende como un tono en sí mismo, por lo que observamos 12 tonos mayores y 12 menores. Además, éstas se encuentran relacionadas estrechamente en el círculo puesto que para cada tonalidad mayor (en el exterior) existe una tonalidad menor (en el interior) con idéntica armadura, diferenciándose tan solo en la tónica o centro tonal.

Se habla de tonos relativos, *relativo mayor* y *relativo menor*, denominación que se puede extender a los propios acordes. Por ejemplo, un acorde de Mi menor (triada menor con tónica en Mi) es el relativo menor del acorde de Sol Mayor (triada mayor con tónica en Sol), porque las tonalidades a las que dan nombre respectivamente comparten armadura. Debido a la doble nomenclatura sostenido/bemol se da pie de nuevo a la *enharmonía*, es decir, a que existan tonalidades y acordes con sonido equivalente.



	con bemoles		con sostenidos	
	MAYOR	menor	MAYOR	menor
0	DO — la			
1	FA	re	SOL	mi
2	SI♭	sol	RE	si
3	MI♭	do	LA	fa♯
4	LA♭	fa	MI	do♯
5	RE♭	si♭	SI	sol♯
6	SOL♭	mi♭	FA♯	re♯
7	DO♭	la♭	DO♯	la♯

Figura 3.7. Círculo de quintas y tabla con las tonalidades mayor y menor que se conforman según el número de alteraciones bemol o sostenido (Iñesta, 2013).

3.3. Acordes

De manera formal, se define un acorde como dos o más notas sonando a la vez o en progresión cercana. Éstas son generalmente una nota fundamental o raíz (nota de comienzo escogida del conjunto de sonidos ya presentado) y una cualidad.

Se construyen comúnmente por intervalos de tercera. A las diferentes líneas de alturas de un fragmento polifónico se les llama comúnmente *voces*, por tanto podemos hablar de las diferentes voces de un acorde (Levine, 2003); también se habla de *factores* del acorde. Una tríada común está compuesta por tres factores, una fundamental, su tercera y su quinta, que forma a su vez un intervalo de tercera con la tercera del acorde.

La cualidad de estos intervalos define también la cualidad del acorde en sí. Así, de entre las posibles combinaciones de intervalos de tercera mayor/menor entre los factores, surgen los cuatro tipos básicos de acorde tríada: mayor, menor, disminuida, aumentada (Tabla 3.3, Figura 3.8).

Las triadas diatónicas son aquellas que se forman por superposición de tres notas pertenecientes a las escalas diatónicas. Todas las posibilidades de acordes diatónicos se dan en la Figura 3.9. El acorde aumentado sólo es diatónico en el contexto de la escala menor armónica (III de Do menor) o la escala melódica ascendente[21].

[21] Menor natural con los grados VI y VII elevados cromáticamente (+1 st).

CUALIDAD	***RELACIÓN FUNDAMENTAL-3ª***	***RELACIÓN 3ª-5ª***
Mayor	3ª mayor	3ª menor
Menor	3ª menor	3ª mayor
Disminuido	3ª menor	3ª menor
Aumentado	3ª mayor	3ª mayor

Tabla 3.3. Intervalos definitorios de los acordes tríada básicos (fundamental, 3ª y 5ª).

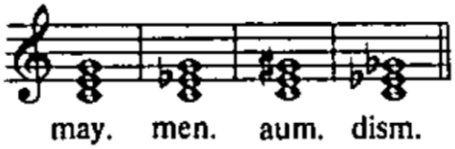


Figura 3.8. Tipos de acordes triada básicos (Piston, 1998).

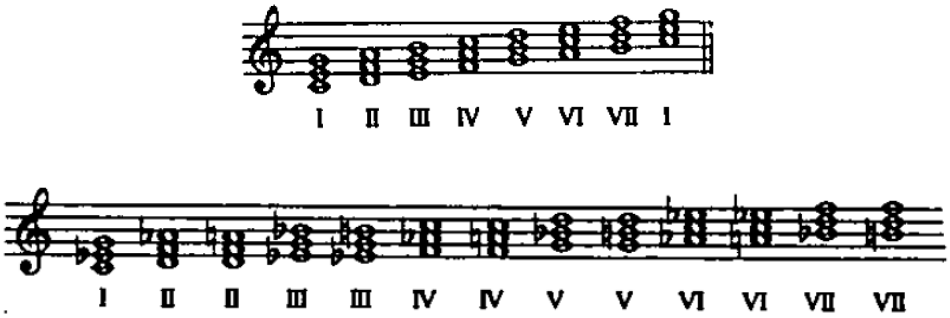


Figura 3.9. Acordes triada diatónicos, debidos a la escala mayor (arriba) y las escalas menores (abajo) (Piston, 1998).

El orden en que están dispuestas las alturas es otro aspecto a considerar: se define la *inversión* del acorde en función de qué factor (fundamental, 3ª, 5ª y más si hubiera) se encuentre más abajo. Un acorde de Do mayor se encuentra en estado fundamental cuando Do es la nota más grave. En primera inversión tendrá el factor 3ª (que en este caso es la nota Mi) como nota más grave, pudiendo las demás notas estar situadas en cualquier disposición y octava. La segunda inversión correspondería

al factor 5ª (nota Sol) en la voz del bajo, y así sucesivamente si hubiera más factores como séptima, novena u onceava[22]. En la Figura 3.10 se ilustran la primera y segunda inversión.

Figura 3.10. Primera y segunda inversión de la triada de Do mayor, con indicación expresa de cuál de los factores se incrementa en una octava para pasar el relevo de la nota del bajo al siguiente factor (Nettles, 1987).

Existen otros acordes tríada que se consiguen mediante otra disposición diferente de los tres factores, y no necesariamente construidos por intervalos de tercera. Uno de estos acordes, que es diatónico y muy común, es el *acorde de cuarta suspendida*, que consiste en sustituir la tercera de la triada mayor o menor por una cuarta respecto a la fundamental (Figura 3.11).

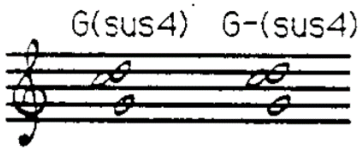


Figura 3.11. Acorde de cuarta suspendida (Nettles, 1987).

Como extensión de las triadas existen las cuatriadas, que se forman generalmente añadiendo un intervalo de séptima respecto a la fundamental o, desde otro punto de vista, una tercera respecto de la 5ª. La diversidad de acordes posibles aumenta en gran medida con cuatro factores o sonidos. A las opciones de acorde anteriores hay que añadir ahora que la séptima puede ser mayor, menor o disminuida. Las cuatriadas diatónicas se muestran en la Figura 3.12 y a continuación haremos una breve revisión de éstas, teniendo presente que también pueden formarse cuatriadas que contengan alguna nota no diatónica. De hecho, cualquier superposición de cuatro notas de distinto nombre forma una cuatriada, y por tanto su número es muy elevado.

[22] Novena, onceava y treceava son intervalos compuestos. Equivalen a una segunda, cuarta y sexta, respectivamente, pertenecientes a la octava superior. Se les llama *tensiones* del acorde, modifican su sonoridad pero generalmente no su función básica.

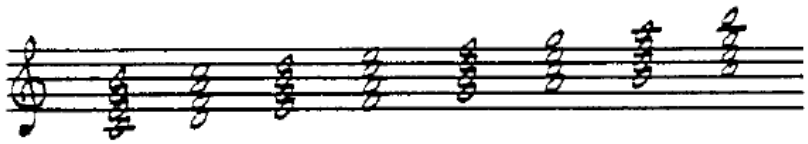


Figura 3.12. Acordes cuatriada diatónicos (Nettles, 1987).

Acordes con tercera mayor, quinta justa y séptima mayor respecto a la fundamental reciben el nombre de *acordes de séptima mayor.* Una notación extendida se preocupa de indicar la cualidad del intervalo de séptima, esto es, *Xmaj7* (Figura 3.13).

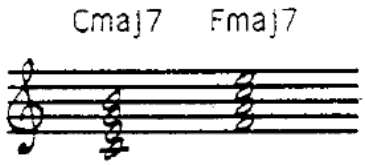


Figura 3.13. Acordes de séptima mayor (Nettles, 1987).

Acordes con tercera menor, quinta justa y séptima menor respecto a la fundamental reciben el nombre de *acordes de séptima menor*. Una posible notación, no libre de ambigüedad, es *X−7* (Figura 3.14).

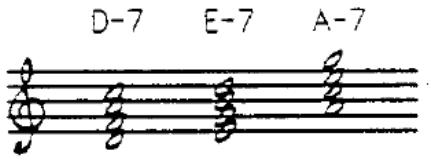


Figura 3.14. Acordes de séptima menor (Nettles, 1987).

Acordes con tercera mayor, quinta justa y séptima menor respecto a la fundamental reciben el nombre de *acordes de séptima de dominante*. Se indican simplemente como *X7* (Figura 3.15). El acorde de séptima de dominante es muy importante para la música tonal pues se dice que es definitorio de la tonalidad: contiene un intervalo de quinta disminuida, también conocido como tritono, que es exclusivo de la tonalidad a la que pertenece (las notas Fa-Si en el acorde G7). Suele indicar el final de una frase musical, pues resulta agradable al oído que este acorde desemboque en la tónica en forma de *cadencia perfecta* (la progresión G7-C, en la tonalidad de Do mayor). Se suele hablar de tensión y distensión asociándolas principalmente a los grados V y I, dominante y tónica. No obstante, puede haber acordes que tengan la estructura de séptima de dominante y no desempeñen la función de dominante en el discurso musical, pues todo depende del uso que se le dé dentro de la pieza. Es una herramienta útil para conocer la tonalidad pero no ofrece garantías absolutas, han de estudiarse otros elementos.

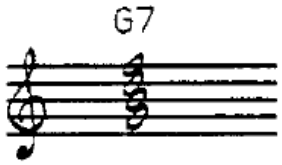


Figura 3.15. Acorde de séptima de dominante (Nettles, 1987).

Acordes con tercera menor, quinta disminuida y séptima menor respecto a la fundamental reciben el nombre de *acordes de séptima semidisminuida* o acordes de séptima menor con la quinta disminuida (en inglés, *half-diminished chords* o *minor 7(b5) chords*, Figura 3.16). Son diatónicos, pues corresponden al VII grado de las tonalidades mayores.

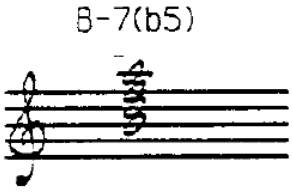


Figura 3.16. Acordes de séptima semidisminuida (Nettles, 1987).

Por último, a los acordes con intervalos de tercera menor, quinta disminuida y séptima también disminuida respecto a la fundamental se les llama *acordes de séptima disminuida* (*Xdim7*, Figura 3.17). Son diatónicos a la tonalidad menor siempre que se sensibiliza la VII, aunque sea momentáneamente; el II grado de las tonalidades menores es de este tipo.

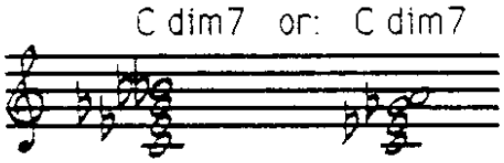


Figura 3.17. Acordes de séptima disminuida. En ocasiones se escriben enarmónicamente: el doble bemol (reducción de un tono entero) se sustituye por la nota diatónica equivalente (Nettles, 1987).

También mencionamos que, en ocasiones y en función de la época y el estilo, pueden aparecer modulaciones o acordes prestados de otras tonalidades u otras escalas (se habla de intercambio o mixtura modal), acordes alterados cromáticamente, etc. Por ejemplo, si un acorde de séptima disminuida aparece en una tonalidad mayor se suele entender como que ésta lo toma prestado de la tonalidad menor. Es estos casos es aún más importante confiar en el reconocimiento de acordes *frame* a *frame*, pudiendo relegar la interpretación de estas eventualidades armónicas a una etapa posterior si procede.

La detección de la tonalidad es una tarea muy relacionada con el reconocimiento de acordes dentro de la MIR, y puede valerse de ello para mejorar su rendimiento, puesto que una progresión de acordes diatónicos está basada en un modelo oculto, el tono. La progresión revela las notas que están empleándose de manera sistemática en la construcción del discurso musical y puede inferirse la tonalidad a partir de ella.

3.4. Consonancia

La consonancia es la base de la armonía, que trata cómo se combinan las alturas mediante intervalos melódicos (una nota con sus adyacentes anterior y posterior) y/o armónicos (una nota con las que suenan simultáneamente con ella). Esta es una de las más complejas disciplinas en música.

Podemos considerar la consonancia como una medida del "parecido" entre dos notas que provoca que suenen bien juntas. Se dice que un intervalo es más o menos *consonante* en función del número de armónicos que comparten los espectros de las notas que lo componen. Dicho de otro modo, en función de que sus frecuencias tengan múltiplos comunes. La falta de consonancia en un intervalo lo convierte en *disonante*.

De especial importancia es el intervalo de una octava que corresponde a una relación 2:1 en frecuencia. Por ejemplo, entre 100 y 200 Hz hay un intervalo de octava, pero también entre 1000 y 2000 Hz, y el incremento en frecuencia es claramente distinto. Ambos intervalos son de octava y se percibirán como iguales (no así las alturas absolutas de los sonidos involucrados).

No existen dos notas más parecidas que una dada y su octava. Tan parecidas son que reciben el mismo nombre. Para distinguir de qué frecuencia se trata se usa el número de la octava. Así, LA_3 son 220 Hz y LA_4, 440 Hz. Evidentemente, la consonancia no es una propiedad que se cumpla o no, sino que se cumple en mayor o menor grado, como vemos en el siguiente caso práctico.

Caso práctico. Sea una nota cuya $f_0 = 100\ Hz$. Consideraremos que esta es la nota de referencia, por lo tanto, la fundamental. La frecuencia de su quinta será $f_0 \times R_V = 3/2\ \cdot f_0 = 150Hz$. ¿Qué armónicos tendrán en común ambas notas? Las frecuencias en Hz comparadas son:

Fundamental:	$100, 200, \mathbf{300}, 400, 500, \mathbf{600}, 700, 800, \mathbf{900}, 1000, \ldots$
Quinta:	$150, \mathbf{300}, 450, \mathbf{600}, 750, \mathbf{900}, 1050, \mathbf{1200}, \ldots$

Observamos cómo 1 de cada 3 armónicos de la fundamental lo son también de la quinta, mientras que 1 de cada 2 armónicos de la quinta lo son también de la fundamental. Esto sucede por la relación 3/2 que hay entre ambas. Son notas muy consonantes, pues tienen muchos armónicos en común.

Podemos inferir que si la relación de frecuencias se puede expresar como una fracción de números muy pequeños, las notas comparadas serán muy consonantes. Conforme aumentan los números enteros que forman la fracción, la consonancia se va debilitando.

Si estudiamos la cuestión de la disonancia desde otro punto de vista, podemos referirnos al estudio al respecto de Helmholtz (1954), que se basaba en la Ley de Ohm Acústica (Rossing y otros (2002)) para explicar el fenómeno. El oído realiza un análisis de Fourier del sonido, separando los sonidos complejos en sus diferentes parciales. La disonancia se percibe cuando los parciales de dos tonos producen de 30 a 40 batidos por segundo. Cuanto más cerca se encuentren los parciales de dos tonos distintos, menor será la probabilidad de que los batidos en ese rango dado produzcan disonancia (Barbancho, et al., 2013).

La Tabla 3.4 muestra los principales intervalos entre notas usados en la música occidental, con sus relaciones de frecuencias correspondientes. Se puede observar que, salvo algunos casos concretos (que tienen una explicación técnica), todas las relaciones usadas son las fracciones más sencillas que pueden darse entre 1 y 2.

La razón para la selección de notas de entre las 12 posibles para construir escalas es histórica y cultural pero tiene la base física de la consonancia. Veámoslo para la escala diatónica mayor. Cuando colocamos en el segmento [1, 2] de la recta real los valores de las relaciones de frecuencias de los intervalos mayores y justos del cuadro anterior, que son los que forman la escala diatónica, comprobamos que la distribución no es homogénea (Figura 3.18).

Se puede ver que las relaciones de casi todas ellas con la tónica son las más consonantes posibles. Son las fracciones cuyos valores están entre 1 y 2, construidas con números y denominadores lo más pequeños posibles (excepto algunas relaciones como la II y la VII). Por el contrario, las notas que comparten pocos armónicos reciben el nombre de *disonantes*. Estas notas crean tensión en el oyente, que percibe una sensación de relajación cuando se le ofrece de nuevo un intervalo más consonante.

Existe un número mucho mayor de acordes disonantes que de acordes consonantes. La consonancia o disonancia en los acordes puede detectarse, como ya se ha adelantado, comparando el espectro del acorde con los espectros de las notas individuales. Durante esta comparación se observa la coincidencia o no de los armónicos, si los armónicos se encuentran muy distantes los unos de los otros, el resultado será consonante, y en caso contrario disonante, por el fenómeno de interferencia en forma de batidos (Barbancho, et al., 2013).

Grado del intervalo	**Distancia en semitonos**	**Relación de frecuencias**	**Ejemplo con notas**
Unísono	0	1 : 1	DO – DO
Segunda menor	1	16 : 15	DO – DO♯
Segunda mayor	2	9 : 8	DO – RE
Tercera menor	3	6 : 5	DO - MI♭
Tercera mayor	4	5 : 4	DO – MI
Cuarta justa	5	4 : 3	DO – FA
Cuarta aumentada	6	45 : 32	DO – FA♯
Quinta disminuida	6	64 : 45	DO – SOL♭
Quinta justa	7	3 : 2	DO – SOL
Sexta menor	8	8 : 5	DO – LA♭
Sexta mayor	9	5 : 3	DO – LA
Séptima menor	10	16 : 9	DO – SI♭
Séptima mayor	11	15 : 8	DO – SI
Octava	12	2 : 1	DO_N– DO_{N+1}

Tabla 3.4. Intervalos expresados mediante semitonos y relaciones de frecuencias según la afinación llamada *justa* o *física* (Iñesta, 2013).

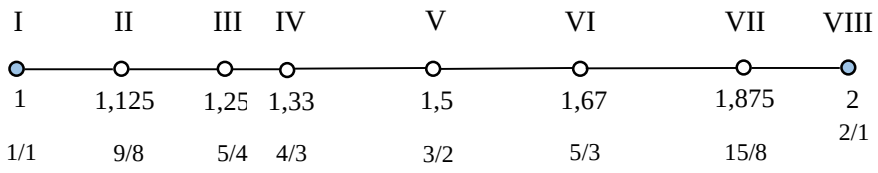


Figura 3.18. Relación entre las frecuencias de la escala diatónica mayor, situadas sobre la recta real (Iñesta, 2013).

3.5. Sistemas de afinación

La afinación en que se interpreta la música es otro factor a tener en cuenta, en absoluto trivial, para poder reconocer el material musical pese a haber una desviación en las frecuencias de las notas. Han existido varios sistemas de afinación desde la época de la antigua Grecia, pero hasta el siglo XVIII no se estableció un estándar comúnmente aceptado para que los instrumentos sonasen bien en cualquier tonalidad.

Los sistemas de afinación justo o físico y pitagórico se basan en las relaciones de consonancia comentadas en el apartado anterior. Con referencia a una nota dada, se calculan las frecuencias fundamentales de las notas restantes. No obstante, si se emplea una referencia distinta para calcular la misma frecuencia o se utilizan relaciones de paso, se obtendrán frecuencias distintas para la misma nota, lo cual es catastrófico para la consonancia.

Los intervalos entre notas consecutivas no son siempre iguales con el sistema de afinación justo (Figura 3.18). Logra que las frecuencias de las notas de la escala sean lo más consonantes posibles, obteniendo un sonido claro y agradable, pero el inconveniente es que sólo suenan bien cuando el instrumento se afina para determinada tonalidad (porque siempre se usa la misma referencia en frecuencia fija, la de la nota que establece la tonalidad), pero hay que volver a afinar el instrumento para ejecutar una pieza en otra tonalidad.

Esta es la razón por las que en las piezas de música clásica se especifica la tonalidad ("Concierto en Re menor para…"), dando de esta manera instrucciones al director sobre los instrumentos a elegir o cómo debían afinarlos. Esto fue necesario hasta que Johann Sebastian Bach demostró a principios del siglo XVIII que partir la octava en 12 partes proporcionalmente iguales era una solución de compromiso entre calidad de sonido y facilidad de afinación.

Los sistemas más conocidos para asignar las frecuencias a las notas son los siguientes:

Pitagórica. La escala pitagórica se basa en el círculo de quintas. Pitágoras propuso que partiendo de la relación privilegiada (por su consonancia y sencillez) entre una nota y su quinta $(3:2)$ junto con la octava $(2:1)$ se pueden construir todas las notas de la octava (ver Figura 3.19), puesto que después de 12 quintas se cierra el círculo y volvemos a estar en la misma nota. Una característica positiva de esta afinación es que todos los intervalos de un tono tienen la relación $9:8$ y todos los de un semitono una de $256:243$ $(2^8:3^5)$.

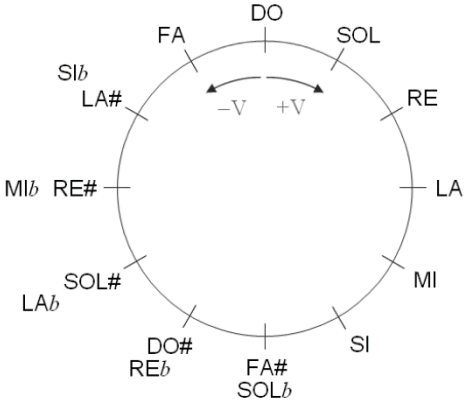


Figura 3.19. Círculo de quintas en el que se basa la afinación pitagórica (Iñesta, 2013).

Para calcular las relaciones de frecuencias de los grados de la escala diatónica mayor se trata de usar siempre la relación de quinta (3: 2) e ir ascendiendo de quinta en quinta multiplicando por este factor. Cuando haciendo la cuenta hasta la siguiente quinta se supera el ámbito de la octava, se toma la nota de la octava inferior dividiendo[23] la frecuencia obtenida entre 2.

Como inconveniente de esta afinación podemos decir que, aunque las relaciones para el semitono y el tono son siempre las mismas, las relaciones respecto de la tónica no son muy sencillas, lo que redundará en falta de consonancia. Se pueden calcular mediante las relaciones mostradas en Tabla 3.5.

Intervalo	II	III	IV	V	VI	VII	VIII
Relación	9: 8	81: 64	4: 3	3: 2	27: 16	243: 128	2: 1
Centésimas	204	408	498	702	906	1110	1200

Tabla 3.5. Relaciones de la afinación pitagórica para obtener las frecuencias fundamentales de la escala diatónica mayor (Iñesta, 2013).

Sin embargo, el gran problema que Pitágoras encontró con esta afinación es que existe un pequeño pero importante desajuste. Cuando se cierra el círculo después de 12 quintas llegamos a la misma nota pero de 7 octavas más arriba (si se ha recorrido el círculo en sentido horario). La relación de frecuencias de 12 quintas es $(3/2)^{12} = 129{,}75$ y una relación de 7 octavas es $2^7 = 128$. Vemos

[23] Nótese que para subir o bajar en términos de altura se suman y restan semitonos o se multiplican y dividen frecuencias.

que la nota final es la misma pero las frecuencias distintas según se hayan calculado. Esto significa que algo no funciona correctamente[24].

Física o justa. En vez de basarse sólo en las quintas, usa también las terceras para formar los acordes de tríada mayores (primero, tercero y quinto grados) en las que se basan las principales sonoridades de la música occidental.

Para calcular las relaciones de frecuencias de los siete grados de la escala diatónica mayor, se construye a partir de DO una triada mayor: I+III+V = DO-MI-SOL utilizando las relaciones $5:4$ (para III) y $3:2$ (para V). Tenemos así ya asignadas las relaciones para MI ($5/4\, f_{DO}$) y SOL ($3/2\, f_{DO}$). Después se parte del punto de llegada anterior (SOL) para construir otra triada mayor a partir de ahí: SOL-SI-RE haciendo los cálculos desde los $3/2$ del SOL. Finalmente, para obtener los grados que faltan hay que construir la triada FA-LA-DO y hacer las cuentas con respecto al DO (quinta descendente) y luego subir el FA y el LA (tercera ascendente) una octava.

La ventaja de esta afinación es que ofrece relaciones muy sencillas entre las notas de la escala diatónica (por tanto, mayor consonancia), como mostramos en la Tabla 3.6, pero presenta distintas relaciones entre notas consecutivas. Las relaciones para un semitono son siempre $16:15$ pero las de un tono son $9:8$ entre I y II y entre VI y VII, mientras que es de $10:9$ entre II y III y entre V y VI.

Intervalo	II	III	IV	V	VI	VII	VIII
Relación	$9:8$	$5:4$	$4:3$	$3:2$	$5:3$	$15:8$	$2:1$
Centésimas	193	386	503	697	884	1088	1200

Tabla 3.6. Relaciones de la afinación física o justa para obtener las frecuencias fundamentales de la escala diatónica mayor (Iñesta, 2013).

Bien temperada. Es la que usan todos los instrumentos modernos. Las anteriores son todas dependientes de la nota de referencia: si se cambia de tonalidad es necesario recalcular las frecuencias. La afinación bien temperada resuelve este problema haciendo lo contrario de lo que parece indicar su nombre: desafinando levemente todas las notas, pero de manera que suenen aceptablemente bien en todas las tonalidades (Figura 3.20).

[24] La 'solución' que adoptaron los músicos fue la llamada *quinta del lobo* (o *coma pitagórica*). Todas las quintas del círculo de quintas eran justas (relación $3:2$) salvo una de ellas empleada para compensar la inexactitud del método, de valor igual a la diferencia entre 7 octavas y 12 quintas ($24\,¢ = \times 1.014$, lo que para un MI$_{\flat 4}$ supone 4.343 Hz). Se situaba entre el SOL$\sharp$ (8 quintas desde el DO en sentido horario) y el MI$\flat$ (3 quintas desde el DO en sentido antihorario) para alejar este intervalo disonante de los habitualmente empleados.

Para demostrar que esto era posible, Bach compuso un estudio denominado *El clave bien temperado* que contaba con piezas en las 24 tonalidades posibles para ser interpretadas por un mismo instrumento sin necesidad de reafinación entre pieza y pieza.

Se consigue mediante la división de la octava en sus 12 notas constituyentes, repartiéndose las frecuencias de forma proporcionalmente igual mediante el factor $\sqrt[12]{2} = 2^{1/12} = 1{,}059463$. De esta manera, cuando se aplica este factor 12 veces desde la tónica para encontrar la octava, la relación que se obtiene es $\left(2^{1/12}\right)^{12} = 2$. Cada semitono se divide siempre en 100 centésimas con esta afinación (Tabla 3.7).

Intervalo	II	III	IV	V	VI	VII	VIII
Relación	$2^{2/12}$	$2^{4/12}$	$2^{5/12}$	$2^{7/12}$	$2^{9/12}$	$2^{11/12}$	2
Centésimas	200	400	500	700	900	1100	1200

Tabla 3.7. Relaciones de la afinación bien temperada para obtener las frecuencias fundamentales de la escala diatónica mayor (Iñesta, 2013).

Para calcular las relaciones de los siete grados de la escala diatónica se trata sencillamente de utilizar el valor de $\sqrt[12]{2}$. Multiplicando sucesivamente por este factor se van encontrando las relaciones $\sqrt[n/12]{2}$ entre las notas, separadas n semitonos de la tónica. Los intervalos bien temperados de un tono entero corresponden a multiplicar por $2^{1/6} = 1{,}122462$.

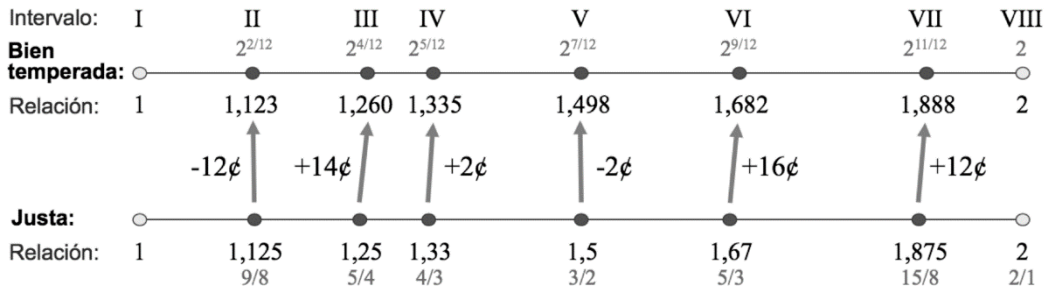


Figura 3.20. Desviaciones en frecuencia (expresadas en centésimas) que supone la afinación bien temperada respecto a la física (Iñesta, 2013).

De esta forma, una misma afinación sirve para cualquier tonalidad, y basta con afinar una vez o, dicho de otra manera, asignar unos valores fijos para todas las notas de una vez para siempre, pudiendo asignar unívocamente una frecuencia a cada nota.

Para algunos músicos con oídos sensibles y entrenados, el temperamento (es decir, la desafinación de compromiso) les puede resultar molesto, pero es la única solución para poder tocar melodías en cualquier tonalidad sin tener que cambiar de instrumento o volver a afinarlo entre tema y tema o cada vez que la obra cambia de tonalidad. Esta última ventaja es importante porque convierte a la evolución de la tonalidad en un mismo tema en un recurso compositivo, lo cual no era posible hasta Bach. Todos los instrumentos modernos usan este sistema.

En la Figura 3.21 se muestran las frecuencias asignadas según la afinación bien temperada a todas las teclas del piano si establecemos la referencia única en el $LA_4 = 440$ Hz. En la parte superior se muestra la localización de los pentagramas en clave de FA y SOL.

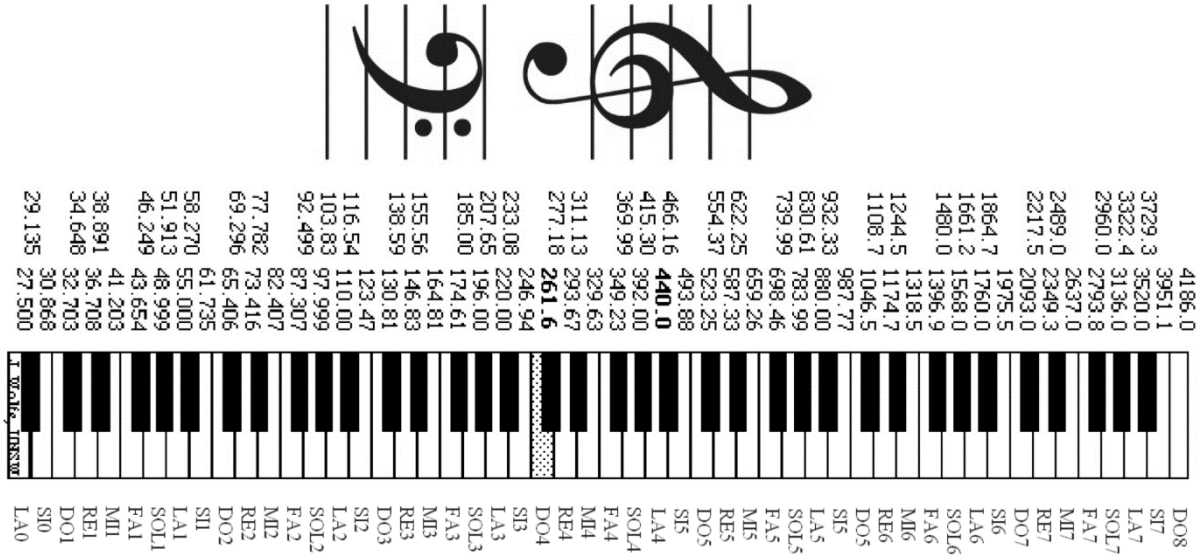


Figura 3.21. Frecuencia de todas las alturas del piano según la afinación bien temperada con la referencia en el $LA_4 = 440$ Hz (Iñesta, 2013).

3.6. Percepción de la altura

Cuando se trata de describir la percepción de la altura como propiedad musical entre en juego el concepto de la circularidad de los sonidos del conjunto cromático. La hélice de alturas es una representación de las relaciones de alturas que sitúa los tonos en la superficie de un cilindro (Shepard, 2001) (Figura 3.22).

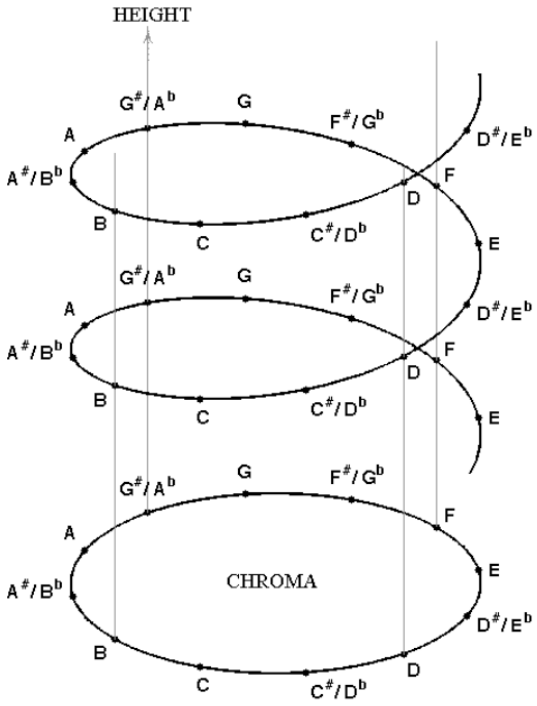


Figura 3.22. Hélice de alturas como modelo de la percepción de esta propiedad (Shepard, 2001).

Este modelo ilustra la relación especial que existe entre los intervalos de octava. Consta de dos dimensiones: la altura (*height*), que dispone los sonidos al uso del eje frecuencial, de más grave a más agudo, y hace referencia a la frecuencia fundamental de la nota; y el croma (*chroma*) que representa la circularidad inherente a la organización de los sonidos, hace referencia a qué clase de altura está sonando de entre las disponibles en la escala cromática bien temperada. Las notas con el mismo nombre se sitúan en la misma posición relativa de los distintos niveles de la hélice, pues son percibidas como la misma clase de altura.

El croma describe el ángulo de rotación de las alturas a medida que se recorre la hélice. Dos clases de alturas idénticas, separadas un intervalo de octava, compartirán el mismo ángulo en el círculo de croma: una relación que no puede expresarse como una escala de alturas lineal. Para analizar la música tonal occidental se discretiza este círculo en 12 posiciones o clases de alturas.

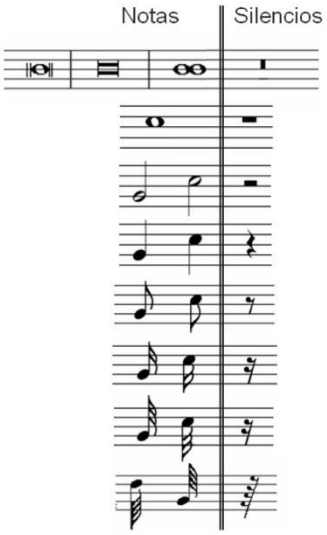
3.7. Duración y ritmo

Como ilustrábamos en la sección 1.5, en la duración de los sonidos musicales hay presentes varias escalas de tiempo diferenciadas. En el discurso musical más comúnmente entendido, podemos reducirlas a dos, una corta y una larga. La escala corta se relaciona con la percepción de cada sonido individual, y la larga da lugar al desarrollo del ritmo, la melodía, frases, pasajes, etc.

La **duración** de las notas musicales se expresa en relación a una nota de referencia o nota patrón definida por el compositor (se habla de tiempos o *beats*). Normalmente se considera que la nota que

dura 1 tiempo se llama *negra* y las demás duraciones se establecen como múltiplos o divisores de esa duración. En la Figura 3.23 se observa la notación gráfica usada para representar las distintas duraciones relativas de las notas y de los silencios.

8 tiempos	Cuadrada
4 tiempos	Redonda
2 tiempos	Blanca
1 tiempo	Negra
1/2 tiempo	Corchea
1/4 tiempo	Semicorchea
1/8 tiempo	Fusa
1/16 tiempo	Semifusa



]

Figura 3.23. Notación gráfica para las duraciones relativas de las notas y silencios. Se ha considerado la negra como patrón de tiempos (1 tiempo) (Iñesta, 2013).

En la escala larga de tiempos, la que relaciona las notas entre sí en *horizontal*, aparecen dos conceptos fundamentales relativos al ritmo:

- **Tempo**: que indica la *velocidad* con la que transcurre la música y del que depende, por tanto, la duración absoluta de las notas, particularmente en el caso de la música tonal (Lester (2005)). En la actualidad se expresa en *beats* por minuto (*bpm*), aunque tradicionalmente se ha descrito el carácter del tempo de manera aproximada con adjetivos italianos como *Allegro, Andante* o *Presto*.
- **Métrica**: que tiene que ver con cómo se organiza la música formando patrones de pulsos fuertes y débiles, fenómeno que da lugar a los distintos *compases* y convenciones para utilizarlos. El más común es el compás de cuatro por cuatro (4/4), que se caracteriza por constar de cuatro tiempos o *beats* que suelen ser: fuerte-débil-semifuerte-débil.

La combinación de estos dos aspectos, duraciones de las notas individuales y repetición de un patrón acentos que se repiten de manera predecible, nos proporciona la sensación que llamamos ritmo. Por tanto, no es sólo la repetición periódica de un pulso de energía sino más bien series de sucesos que se repiten en el tiempo con una cierta estructura.

El tempo es un fenómeno perceptual, no existe físicamente, pero el ser humano es capaz de, a través del patrón de acentos y generalmente sin gran esfuerzo o preparación musical alguna, encontrar el

tactum, el pulso que se suele marcar con el pie, con una mano o con un leve movimiento de cabeza y que, aun en un nivel estrictamente perceptual, proporciona el marco organizativo para toda la pieza (Grosche y Müller, 2011a). A un nivel más fino se habla de *tatum* o átomo temporal, que hace referencia a la tasa más alta de repetición de patrones con significado musical en la pieza, y a un nivel mayor en cuanto a duración encontramos la unidad de compás, que como comentábamos sirve para agrupar los pulsos en bloques regulares según su patrón de acentos (que no obstante puede ser alterado localmente en favor de la expresividad, pero mantiene casi siempre un carácter periódico).

El ritmo tiene una gran importancia en la transcripción automática de música, muchos de los investigadores citados en el Capítulo 4 han desarrollado mejoras basadas en la detección del *beat*, entre otras características, pero no nos detendremos a ahondar en estos conceptos en mayor profundidad porque tan solo los emplearemos tangencialmente en el presente trabajo. Trabajaremos en una escala de tiempos absoluta, expresada en segundos; esto es, tiempo físico (*chronos*) frente a tiempo percibido (*tempus*) (Roads (2001)). Sí que emplearemos en la versión final del sistema ACE propuesto un algoritmo para obtener información relativa al tempo y dotar así de cierta estabilidad a la estimación (Müller y Kurth (2010); Grosche y Müller (2011a); Grosche y Müller (2011b)). Nos remitimos a autores como Herrera (2015) o Lester (2005) para una mayor comprensión de los aspectos rítmicos de la música.

3.8. Timbre

El timbre es una cualidad asignada al sonido desde un nivel perceptual. Subjetivamente, se puede definir como aquella propiedad que nos permite distinguir un instrumento de otro, aunque esté tocando la misma nota, o las voces de distintas personas en una conversación.

Tiene que ver principalmente con el espectro o la forma de onda, que a fin de cuentas es lo mismo (Figura 3.24), y muy especialmente con los formantes, tanto la cantidad de ellos como las frecuencias en las que se centran. No obstante, no es el único factor que determina el timbre. Existen otros factores como la envolvente temporal o espectral y las componentes ruidosas inherentes a la fuente sonora.

Es un fenómeno complejo, puesto que en muy diferentes situaciones en las que el espectro se distorsiona o cambia (sonido de trompeta a través de un teléfono móvil; una voz humana susurrando o gritando) aún somos capaces de percibir el timbre.

Figura 3.24. Aspecto de 3 ciclos de las ondas sonoras de una misma nota emitida por (a) un clarinete, (b) un oboe y (c) un violín. En cada caso se ha marcado la duración de un periodo fundamental, que coincide para los tres casos, lo que indica que se está emitiendo la misma altura (Iñesta, 2013).

3.8.1. Descripción espectral del timbre

Existen numerosas características del espectro que tienen influencia en el timbre percibido y que se pueden expresar matemáticamente. Hay diferentes formas de clasificar estas características. Por ejemplo, pueden describir aspectos de bajo nivel como la energía o determinadas frecuencias relevantes, o de más alto nivel, como el brillo o la armonicidad, más relacionadas con la percepción humana. Hay características que se evalúan sobre la señal en el tiempo y otras que se evalúan sobre el espectro. También hay características *estáticas*, que se calculan a partir del espectro de una ventana, y *dinámicas*, que se calculan a partir de una serie de ventanas del espectrograma.

3.8.1.1. Características estáticas

Por ejemplo, una de las características estáticas más utilizadas es el *centroide espectral*, que está relacionada directamente con el "brillo" que percibimos en un sonido. Cuando decimos que un sonido es brillante es porque tiene componentes apreciables de altas frecuencias y esto queda reflejado en el centroide espectral, que viene a ser el cálculo de una media ponderada de las frecuencias f del espectro:

$$c_f = \frac{\sum_f f \cdot S(f)}{\sum_f S(f)} (\text{Hz}) \ .$$

Como se observa, la ponderación de las diferentes frecuencias del espectro se realiza mediante las amplitudes de dichas frecuencias, de manera que a mayor amplitud $S(f)$, mayor peso de f en el cálculo del centroide. En ocasiones podemos encontrar una versión adimensional de este descriptor, dividido por el valor de la frecuencia fundamental del sonido:

$$c = \frac{\sum_f f \cdot S(f)}{f_0 \sum_f S(f)} \ .$$

Nótese que en este caso lo que estamos haciendo es considerar el espectro como una distribución estadística de frecuencias. Siguiendo con esta analogía podemos ir más allá y calcular también el momento de orden 2 de esta distribución, que es la *varianza* del espectro:

$$d^2 = \frac{\sum_f \left(c_f - f\right)^2 \cdot S(f)}{\sum_f S(f)} \ ,$$

que es una medida de lo cerca o lejos que están las frecuencias alrededor del centroide. Dicho de otra manera, nos da una medida de la *dispersión* del espectro, que, para poder expresarla en Hz, se calcula como

$$d = \sqrt{d^2}\ (\text{Hz})\ .$$

Otra característica usada a menudo es la *pendiente espectral*, que nos indica cuánto decae la energía del espectro con la frecuencia. En todos los sonidos naturales, la energía tiende a decaer con la frecuencia (véase la Figura 3.25), por lo que si hacemos un ajuste por mínimos cuadrados de los valores del espectro obtendremos una recta de pendiente negativa. El valor de la pendiente de esa recta es la citada característica, que será mayor en valor absoluto cuanto más rápidamente decaiga la frecuencia y viceversa.

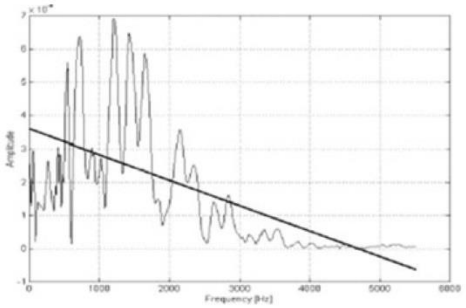


Figura. 3.25. La pendiente espectral se calcula como el valor de la pendiente de una recta que se ajusta a los valores del espectro (Iñesta, 2013).

Relacionada con la pendiente espectral también se usa la llamada frecuencia *roll-off* o de corte. Se define como el valor de frecuencia que hace que cierto porcentaje de la energía del espectro esté por debajo de ella. Este porcentaje suele fijarse entre el 80 y el 95%.

$$f_R = \max, f \left\{ \sum_{k=1}^{f} S(k) < 0.95 \cdot \sum_{k=1}^{N-1} S(k) \right\} \ ,$$

expresada en este caso para el 95% de la energía.

Finalmente el *flujo espectral* es una medida del grado de variación local del espectro. Se calcula entre ventanas adyacentes ($t-1$ y t) para el valor $\Phi(t)$. Para calcularlo se consideran los espectros de esas dos ventanas como si fueran dos vectores y se calcula la norma del vector diferencia:

$$\Phi = \sum_{t>0} \Phi(t) = \sum_{t>0} \sum_{k=0}^{N-1} \left(\tilde{S}_t(k) - \tilde{S}_{t+1}(k)\right)^2 \quad ,$$

donde las $\tilde{S}$ son las versiones normalizadas de los espectros

$$\tilde{S}_t(k) = \frac{|S_t[k]|}{\sum_{l=0}^{N-1}|S_t[l]|} \quad ,$$

que se utilizan para que el resultado sea independiente del número de bandas utilizadas en el espectro.

3.8.1.2. Características dinámicas

La evolución del espectro en el tiempo también es de gran importancia en la caracterización del timbre. Este concepto recibe el nombre de *envolvente* espectral. En el caso de la síntesis de sonido imitativa, que trata de emular un instrumento real, si no se modifican esos parámetros de acuerdo con un patrón característico, el sonido resultante resulta artificial y poco creíble, pues todos los sonidos naturales poseen esta característica.

Un parámetro fundamental en este caso que debe ser escogido a priori es el tamaño de la ventana y de su desplazamiento. Tamaños típicos están entre 512 y 4096 muestras (entre 11.6 y 92.9 ms a $f_s =$ 44100 Hz) y el desplazamiento suele tomarse del 50%.

Una característica muy sencilla es el promedio temporal del centroide, calculado como

$$\hat{c}_f = \frac{1}{N}\sum_i c_{f_i} \ ,$$

donde *N* es el número de ventanas utilizadas en el cálculo. Nos proporciona una idea de qué valores va tomando el centroide a lo largo de la duración del sonido. Si queremos más información sobre la evolución temporal del centroide también podemos calcular la desviación típica de los valores que toma. Otro parámetro relevante de su evolución temporal es la llamada *velocidad* del centroide, que nos informa de lo rápido o lento que evoluciona su valor en el tiempo. Para ello se suman las primeras derivadas de sus valores, mediante

$$\dot{c} = \frac{\partial c}{\partial t} = \sum_{m=n}^{M} c[n-m] \quad ,$$

donde M es el alcance de la diferenciación. A veces este descriptor se expresa mediante el operador gradiente como ∇c.

Nótese que estas mismas operaciones se pueden aplicar a la evolución de cualquier otra característica estática para conocer su evolución en el tiempo.

Envolvente temporal

En el dominio exclusivo del tiempo, la envolvente temporal de una señal es una línea imaginaria que une los puntos de amplitud máxima de una onda a lo largo del tiempo. Cada sonido tiene su propio tipo de envolvente (véase Figura 3.26) que forma parte de su timbre característico.

La descripción de la envolvente se puede parametrizar utilizando las siguientes magnitudes:

- Tiempo de ataque (t_A): el transcurrido desde que comienza el sonido hasta que se alcanza el máximo de amplitud.
- Amplitud de sostenimiento (A_S): la amplitud de la fase estable, caracterizada por una situación estacionaria en la que la nota mantiene aproximadamente su nivel sonoro mientras se suministra energía al instrumento.
- Tiempo de relajación (t_R): el transcurrido desde que el instrumento deja de tocarse hasta que la nota se extingue.

Existen varios procedimientos para calcular la envolvente a partir de los valores de una señal. Una posibilidad es filtrar la onda rectificada (el valor absoluto de la onda[25], $|s(t)|$) mediante un filtro paso bajo de frecuencia de corte muy baja, subsónica, como por ejemplo 5 Hz. De esta manera, el filtro responderá de manera muy lenta a las variaciones (ahora siempre positivas) de la señal, obteniendo una curva muy parecida a la envolvente buscada.

Otro procedimiento habitualmente utilizado es calcular el valor de amplitud RMS (*root mean square*) instantáneo en ventanas consecutivas de 100 ms, suponiendo que N sea el número de muestras que contiene una ventana de esa duración:

$$A_{RMS}\{s(t)|_{100\,\mathrm{ms}}\} = \sqrt{\frac{1}{N}[s^2(t) + s^2(t+1) + \ldots + s^2(t+N+1)]}$$

[25] Una alternativa a la rectificación de la señal es elevarla al cuadrado. De esta manera, se obtiene la ventaja adicional de trabajar sobre potencias en vez de amplitudes, lo cual se parece más al volumen percibido por nuestros oídos.

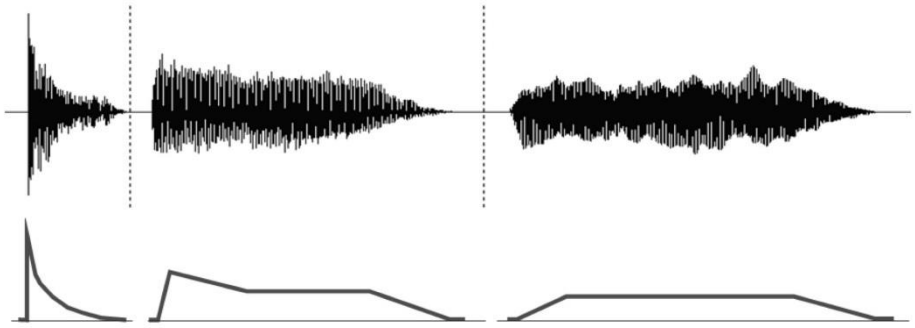


Figura 3.26. Tres ondas correspondientes a una nota emitida por: (a) un tambor, (b) un piano y (c) un violín. En la parte inferior se observa el modelo simplificado de las respectivas envolventes temporales (Iñesta, 2013).

O la amplitud de pico, que es simplemente la muestra de mayor valor de cada ventana

$$A_{pico}\{s(t)|_{100\text{ ms}}\} = max\ |s(t)|$$

La unión con una línea de las amplitudes calculadas para las diferentes ventanas consecutivas nos dará la envolvente (ver Figura 3.27). Ninguna de estas medidas puede ser negativa, pero pueden ser exactamente cero si la señal tiene valores nulos para toda la extensión de la ventana.

La amplitud RMS puede ser igual a la amplitud de pico, pero nunca excederla. Puede ser tan pequeña como $1/\sqrt{N} \times A_{pico}$, pero nunca ser más pequeña que eso.

Uno de los aspectos más interesantes de los mecanismos de percepción del timbre es que en el espectro de cualquier onda compleja como las de los instrumentos reales hay muchas frecuencias pero se integran en una sola altura con una cualidad que es el timbre concreto del instrumento. A este fenómeno se le llama *fusión* y es únicamente perceptual: no existe físicamente. Es el resultado cognitivo de combinar información espectral y temporal. La fusión se da de manera natural para espectros armónicos, mientras que en sonidos inarmónicos solemos tender a percibir los parciales como superpuestos pero sin darse esa fusión en un solo sonido de timbre determinado.

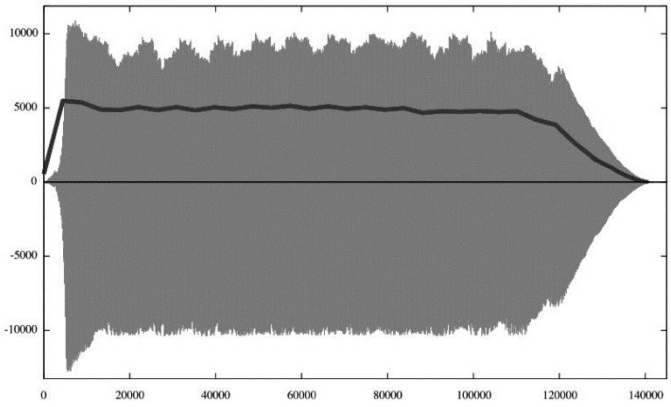


Figura 3.27. Onda del sonido de una nota de saxofón ($f_s = 44100$ Hz y $N = 4410$ muestras (100 ms)) y su envolvente temporal calculada usando RMS, representada sobre ella (Iñesta, 2013).

Componentes ruidosas

Otra característica importante es la presencia de ruido y su carácter. Junto a las componentes periódicas, en todo sonido musical también existen componentes no periódicas y ruidos que intervienen en la caracterización tímbrica. El ruido, por su naturaleza inarmónica, no tiene una altura definida y está compuesto por muchos parciales distribuidos irregularmente. Un golpe de ruido inicial de alta frecuencia es esencial para el timbre de algunos instrumentos como la flauta o las cuerdas.

Capítulo 4

Extracción de características musicales de la señal de audio

Con el fin de describir los aspectos tonales de una pieza musical en formato de audio digital, es necesario extraer automáticamente la información relativa a las notas musicales presentes en cada instante de tiempo a través del análisis de la señal de audio.

En este apartado evaluamos los diferentes enfoques de la extracción de características del audio, incluyendo los métodos orientados a la transcripción mediante la estimación de la altura y los descriptores de distribución de croma (Gómez, 2006a).

4.1. Enfoques basados en transcripción automática

Dado que la gran mayoría de la literatura en relación con la inducción de la tonalidad está orientada al análisis de partituras, una solución sencilla para aplicar estos métodos a audio sería extraer automáticamente información propia de la partitura de las señales de audio. Este es el objetivo de los sistemas de transcripción automática. No obstante, veremos que es posible emplear la información obtenida sobre las clases de alturas sin necesidad de disponer de una transcripción exacta. Por esta razón, en este examen nos centraremos en el análisis de los métodos de extracción de características de distribución de croma a partir de audio (ver sección 4.3). Tan sólo consideramos aquí los enfoques principales para la transcripción automática de audio polifónico. Nos centramos en la estimación de frecuencias fundamentales, aunque hay otras tareas relacionadas (por ejemplo, segmentación o estimación de las notas de la línea melódica de las diferentes voces) que tienen una gran relevancia en los sistemas de transcripción automática.

El objetivo de esta sección es hacer una revisión de las principales técnicas para estimar la frecuencia fundamental de señales monofónicas y polifónicas, para así hacerse una idea de cuál es el actual estado de la tecnología y la precisión que podemos alcanzar a día de hoy. Nos referimos a Gómez (2001); Gómez y otros (2003), Klapuri (2004) y de Cheveigné (2006) para una discusión detallada.

4.1.1. Estimación de la frecuencia fundamental de señales monofónicas

La frecuencia fundamental es el principal rasgo de bajo nivel a considerar cuando se describe la tonalidad. Dada la importancia de la detección de altura para el análisis del habla y la música se ha investigado profusamente en este campo. También podemos encontrar numerosos estudios y

evaluaciones de algoritmos de detección de altura para diferentes propósitos, tales como los de Hess (1983); Roads (1996); Romero y Cerdá (1997); Kostek (2004); de Cheveigné (2005).

Gran cantidad de literatura se ocupa del análisis de grabaciones de audio monofónico[26] con el objeto de estimar su frecuencia fundamental. La primera solución que se dio fue adaptar algunas de las técnicas propuestas para el habla, como la que presentó Hess (1983). Más tarde, otros métodos fueron específicamente diseñados para tratar con música.

Todos los algoritmos de estimación de frecuencia fundamental nos dan una medida correspondiente a una porción de la señal (ventana de análisis). De acuerdo con McKinney (citado en Hess, 1983), el proceso de detección de la frecuencia fundamental puede subdividirse en tres pasos principales que se ejecutan sucesivamente: el pre-procesador, el extractor básico y el post-procesador (Figura 4.1). El extractor básico ejecuta la tarea principal de medida: convierte la señal de entrada en una serie de estimaciones de frecuencia fundamental. El principal cometido del pre-procesador es facilitar la extracción de la frecuencia fundamental (por ejemplo, eliminar el ruido o normalizar). Finalmente, el post-procesador es el bloque que realiza tareas más diversas, como la detección y corrección de errores o el suavizado del contorno obtenido.

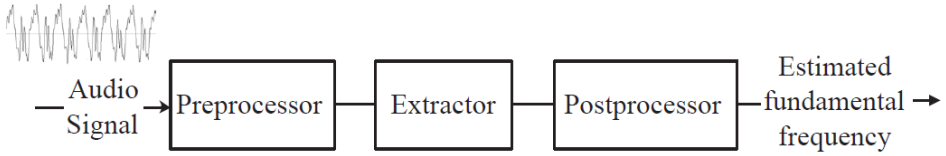


Figura 4.1. Etapas del proceso de detección de frecuencia fundamental (Gómez, 2006a).

Hay multitud de formas de clasificar los distintos algoritmos. Pueden clasificarse de acuerdo con el dominio en el cual se procesa. Siguiendo esta regla, podemos diferenciar entre algoritmos en el dominio del tiempo, que tratan con formas de onda, y algoritmos en el dominio de la frecuencia, que parten de los espectros de las señales. Esta distinción entre algoritmos de dominio temporal o frecuencial no siempre es clara, puesto que algunos de los algoritmos pueden ser expresados en ambos dominios, como el método de la función de autocorrelación (ACF). Otra manera de clasificar los diferentes métodos, más adaptada al dominio espectral, puede ser distinguir entre algoritmos de espacio espectral o algoritmos de intervalo espectral, de acuerdo con Kostek (2004). Los algoritmos de espacio espectral, tales como el método ACF y el análisis del cepstrum[27], ponderan las

[26] Para resolver la ambigüedad de lenguaje que se crea en este caso al hablar de audio musical, emplearemos el término *monofónico* para referirnos al carácter eminentemente melódico del contenido musical, y reservaremos la palabra *mono* para referirnos a un archivo de audio de un solo canal.

[27] El cepstrum de una señal es el resultado de calcular la transformada de Fourier (FT) del espectro de una señal en escala logarítmica. En ocasiones se le llama el espectro del espectro, y aporta información sobre el ritmo de cambio de cada una de las bandas de éste. Se emplea en análisis de señal musical para detección de las

componentes espectrales de acuerdo con su localización en el espectro. Otros sistemas, como los basados en la periodicidad del espectro o el procesamiento de la autocorrelación del espectro, utilizan la información correspondiente a los intervalos entre componentes espectrales. Así pues, el espectro puede ser desplazado arbitrariamente sin afectar al valor de salida. Estos algoritmos funcionan relativamente bien sobre sonidos que presentan inarmonicidad, puesto que los intervalos entre armónicos gozan de mayor estabilidad que la situación espectral de los parciales.

Los **algoritmos de dominio temporal** tratan de encontrar la periodicidad de una señal sonora de entrada en el dominio del tiempo. La **tasa de cruces por cero** (ZCR) es de las primeras y más simples técnicas orientadas a estimar el contenido frecuencial de una señal en el dominio del tiempo. Este método es muy simple y nada costoso, pero no demasiado preciso. El valor de la ZCR ha demostrado también correlacionar notablemente con el centroide espectral, de modo que puede tener que ver más con el timbre que con la altura.

Los algoritmos basados en la función de **autocorrelación** (ACF) en el dominio temporal han sido de los más frecuentemente empleados para estimar la frecuencia fundamental (ver Medan, et al., (1991); Talkin, 1995; de Cheveigné y Kawahara, 2002). La autocorrelación puede también ser aplicada en el dominio frecuencial, ya que la ACF es la FFT inversa de la densidad espectral de potencia. De acuerdo con Kostek (2004), los sistemas basados en ACF pueden denominarse estimadores de la frecuencia fundamental de tipo posición espectral. Los detectores de frecuencia fundamental basados en ACF han sido reseñados como relativamente inmunes al ruido (Romero y Cerdá, 1997) pero sensibles a formantes y peculiaridades espectrales del sonido analizado (Kostek, 2004).

Otros métodos están basados en el análisis de la **periodicidad de la envolvente** (EP). Están considerados como orientados al paradigma de intervalo espectral (Kostek, 2004), pues analizan las diferencias en frecuencia entre las diferentes componentes. Los modelos más recientes de percepción humana de la altura calculan la periodicidad de la envolvente separadamente en distintas bandas de frecuencia y combinan los resultados entre canales (Meddis y Hewitt, 1991). Estos métodos tratan de estimar la altura percibida, y no la periodicidad puramente física, en señales acústicas de diversa índole. El algoritmo de Terhardt es uno de los primeros y valiosos modelos que se han propuesto (Terhardt, 1979; Terhardt, et al., 1981). Exceptuando los algoritmos más simples, que tan solo se centran en encontrar la periodicidad de la señal, los estimadores de "altura percibida" emplean algún tipo de conocimiento acerca del sistema de audición humana en sus diferentes etapas, en el pre-procesado, extracción o post-procesado de los datos. Por tanto, pueden ser considerados estimadores de altura. En cualquier caso, dado que la psicoacústica tan solo se aplica a la hora de mejorar la

frecuencias fundamentales, como se explica en el texto, y en análisis del habla para identificación de voz, entre otras aplicaciones.

estimación de la periodicidad y no se emplea un modelo completo de percepción de altura, algunos fenómenos de la audición no reciben explicación.

Los **enfoques de procesado en paralelo** están basados en numerosos procesos ejecutados en paralelo, como en el caso del detector de frecuencia fundamental definido por Gold y modificado por Rabiner (Gold y Rabiner, 1969; Rabiner y Schafer, 1978), mostrado en la Figura 4.2. Esta vía no ha sido todavía muy explorada, pero es plausible desde el punto de vista de la percepción humana y puede ser fructífera. Bregman (1998) señala que debe de haber un alto grado de redundancia en los mecanismos responsables de muchos de los fenómenos de la percepción. Numerosos procesos distintos analizan el mismo problema, y donde uno fracasa otro tiene éxito.

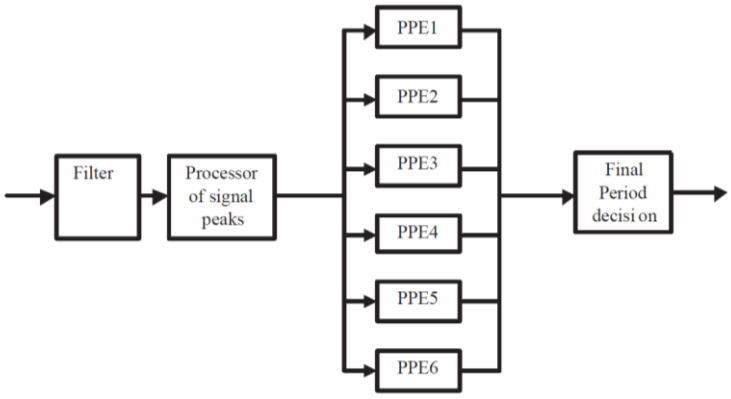


Figura 4.2. Enfoque de procesamiento en paralelo (Gómez, 2006a).

Los **algoritmos de dominio frecuencial** utilizan la información del espectro de la señal, obtenida a través de una STFT u otra transformación. El algoritmo de detección de altura basado en el **cepstrum** (Noll, 1967) fue el primer algoritmo que utilizaba un análisis frecuencial a corto plazo (ST) que demostró ser realizable en un ordenador. El cepstrum, como ya se apuntaba, es la transformada inversa de Fourier del logaritmo de la densidad espectral de potencia de la señal (aplicar dos DFT consecutivas es muy similar a aplicar una DFT inversa sobre un espectro). La secuencia de pulsos producida por la periodicidad reaparece en el cepstrum como un pico evidente en la frecuencia logarítmica (en inglés, "*quefrency*") T_0, que es fácilmente reconocible por la lógica del extractor básico. La detección de la frecuencia fundamental en el cepstrum tiene un alto grado de parecido con los sistemas ACF. A diferencia de éstos, los estimadores de altura basados en cepstrum tienen un comportamiento poco deseable en presencia de ruido pero obtienen buenos resultados con formantes y peculiaridades espectrales de los sonidos analizados (Kostek, 2004; Romero y Cerdá, 1997).

Los métodos de **autocorrelación del espectro** se basan en detectar el periodo del espectro de amplitud utilizando su función de autocorrelación. Encontramos una buena implementación de este principio en Lahat, et al., 1987.

Los métodos de **comparación armónica** tratan de extraer un periodo de un conjunto de máximos del espectro de amplitud de la señal. Una vez estos picos espectrales son identificados, pueden ser comparados con los armónicos predichos para cada una de las notas candidatas, para ver en qué medida coinciden. Este enfoque ha sido ampliamente utilizado en Piszczalski and Galler (1979), Maher and Beauchamp (1993) y Doval and Rodet (1991). La solución presentada en Maher and Beauchamp (1993) consiste en un cálculo de doble vía del error de coincidencia, tal y como está ilustrado en la Figura 4.3. La primera está basada en la diferencia en frecuencia entre cada parcial en la secuencia medida y su vecino más cercano en la secuencia predicha, y la segunda se basa en el error de coincidencia entre cada armónico en la secuencia predicha y su parcial vecino más cercano en la secuencia medida (método TWM, *Two-Way Mismatch error*).

.

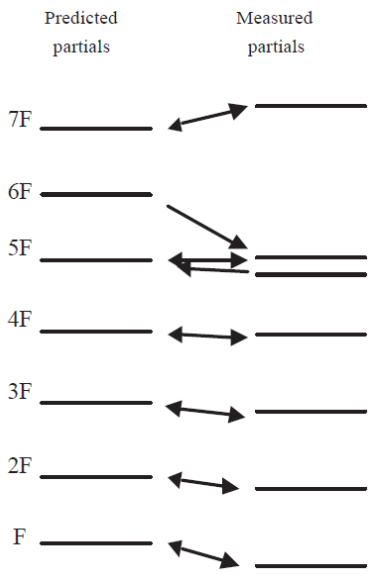


Figura 4.3. Procedimiento para calcular el error de coincidencia del espectro de amplitud empleando dos vías (Gómez, 2006a).

La transformada *wavelet* (WT) es una herramienta de análisis multi-escala y multi-resolución que ha demostrado ser muy útil para procesamiento de música por su parecido con la forma en que el oído humano procesa el sonido. En contraste con la STFT, que utiliza una única ventana de análisis, la WT utiliza ventanas pequeñas en frecuencias altas y ventanas grandes a baja frecuencia. Se han propuesto algunos algoritmos de frecuencia fundamental basados en la WT para analizar señales de voz (por ejemplo, en Jehan, 1997). Siguiendo la idea del análisis en frecuencia con factor Q constante, Klapuri (2000b) propuso un algoritmo para el análisis de la periodicidad que calcula estimaciones de frecuencias fundamentales independientes en bandas de frecuencia separadas. Estos valores son entonces combinados para ofrecer una estimación global. De esta manera se solucionan diversos

problemas; se proporciona robustez en el caso de señales gravemente deterioradas, en las que tan solo un fragmento de todo el rango frecuencial tiene la calidad suficiente para ser utilizado.

En la Tabla 4.1 mostramos un resumen de las características de los algoritmos discutidos. La información acerca de los resultados de su ejecución se ha extraído de diferentes fuentes como Kostek (2004); Romero y Cerdá (1997) y del propio trabajo de cada autor. Ciertos algoritmos se comportan mejor en diferentes situaciones, pero no hay todavía una solución para todas las fuentes sonoras en todas las condiciones.

Los algoritmos de detección de frecuencia fundamental suponen que dicha frecuencia fundamental está presente, pero no siempre es así. Puede haber segmentos en los que no haya ninguna altura reconocible, como por ejemplo los tramos de silencio, solos de percusión o segmentos ruidosos. Un proceso de segmentación debe distinguir entre periodos "afinados" (en los que hay una altura discernible) o no, de modo que la detección de frecuencia fundamental tan solo sea realizada en las partes en que suene un sonido dotado de altura. Sin embargo, muchas de las técnicas utilizadas para segmentación de altura/no altura ya usan la frecuencia fundamental estimada para decidir si la información es válida o corresponde a una señal en la que no puede encontrarse ningún tono, como añadidura a otras características extraídas de la señal (ver, por ejemplo, Cano, 1998).

Method	**Domain**	**Spectral Place/ Interval**	**Simplicity**	**Noise**	**Spectral Peculiarities**
ZCR	Time	SP	Very simple		
ACF	Both	SP	Simple	Relatively immune	Sensitive
EP	Time	SI	Simple		
Rabiner (parallel processing)	Time	SP	Relatively simple		
Cepstrum	Frequency	SP	Simple	Poor Performance	Relatively immune
Spectrum AC	Frequency	SI	Simple		
Harmonic Matching Method	Frequency	both	Quite complex	Relatively immune	Relatively immune
Wavelet based method	Frequency (WT)		Quite complex	Immune	
Bandwise Klapuri	Frequency	both	Quite complex	Relatively immune	Relatively immune

Tabla 4.1. Resumen de los diferentes métodos para estimar la frecuencia fundamental (Gómez, 2006a).

Los **métodos de pre-procesado** están dirigidos a suprimir el ruido y enfatizar las características útiles para la estimación de la frecuencia fundamental. Algunos de los métodos de pre-procesado empleados en procesamiento del habla se detallan en Hess (1983), incluyendo técnicas para aislar el primer parcial, filtrar paso bajo moderadamente para eliminar la influencia de los formantes superiores, implementar filtrados inversos para eliminar la influencia del tracto vocal y estimar la fuente de la voz (excitación), implementar filtros peine para enfatizar las estructuras armónicas, procesar de forma no lineal en el dominio espectral, producir saturación central (que destruye la estructura de formantes sin arruinar las estructuras periódicas de la señal), distorsionar la señal utilizando una función no lineal uniforme, detección de envolvente, detección de envolvente instantánea, etc. Estos algoritmos pueden también utilizarse para detectar la frecuencia fundamental de señales musicales. Algunos métodos de pre-procesado han sido definidos desde un principio para procesar música. El método propuesto por Klapuri aplica los principios del procesado espectral RASTA; trata de eliminar tanto el ruido aditivo como el convolutivo simultáneamente (ver Klapuri, et al., 2001).

El contorno de las frecuencias fundamentales que se obtiene como salida de los diferentes algoritmos es normalmente ruidoso y puede estar afectado por errores aislados, por lo que han sido definidos diferentes **métodos de post-procesado** para corregirlos. La forma más habitual de suavizar una función es la convolución de la señal de entrada con la respuesta al impulso de un filtro paso bajo. Tal y como se presenta en Hess (1983), la aplicación de filtros paso bajo elimina en gran medida el *jitter* local y el ruido, pero no los grandes errores locales y, además, difumina las discontinuidades intencionadas en las zonas de transición altura/no altura. Por tanto, algún tipo de suavizado no lineal puede ser más apropiado. En un artículo escrito por Rabiner, et al., (1975) se propone el suavizado de mediana como método no lineal. Vemos esto aplicado a la detección de acordes en la Figura 4.4.

Encontramos otro enfoque en Laroche (1995). El procedimiento consiste en almacenar un elevado número de posibles valores para la frecuencia fundamental en cada instante de análisis, asignándoles una puntuación que representa la bondad de la estimación. El objetivo es encontrar una sucesión que, siguiendo una de las estimaciones por cada instante temporal, tenga la mejor puntuación. Otros métodos de post-procesado son dependientes del algoritmo utilizado para el análisis sonoro y la detección de frecuencia fundamental. Un ejemplo es el enfoque adoptado en Serra (1996). La frecuencia fundamental se estima valiéndose de información de máximos en el espectro. Los picos espectrales son procesados *frame* a *frame* utilizando enventanado y análisis FFT. En el procedimiento del enventanado, el tamaño de la ventana se actualiza en función de la frecuencia fundamental. Si se produce un error en la estimación, el tamaño de la ventana en los instantes siguientes no será escogido correctamente. Con el fin de corregir este tipo de errores, se lleva a cabo un nuevo análisis sobre un grupo de muestras comenzando por la última y finalizando en la primera.

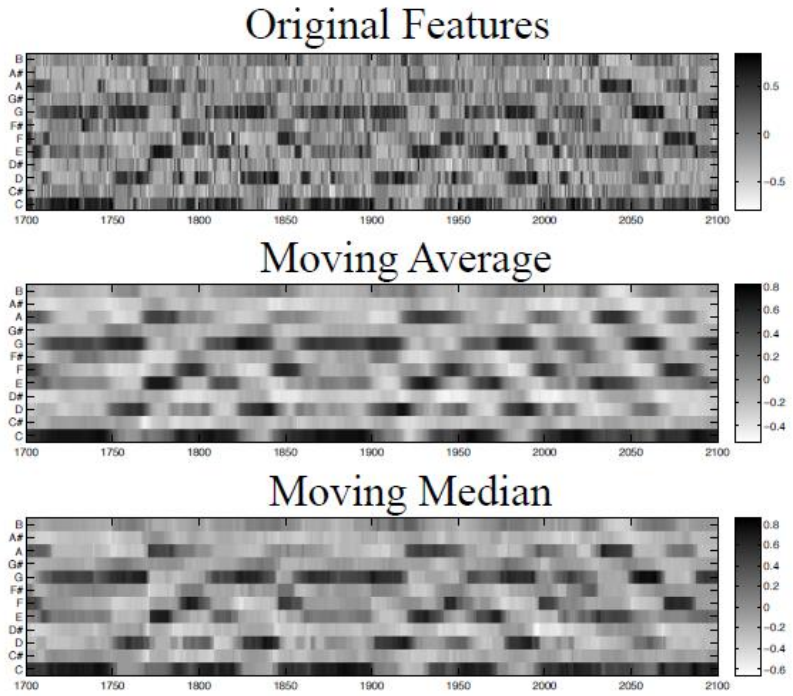


Figura 4.4. Métodos de procesado no lineal aplicados sobre un cromagrama ruidoso: promediado (en medio) y filtro de mediana (abajo). El primero es más limpio aunque difumina excesivamente los bordes, y por tanto las transiciones entre acordes. El segundo es una solución de compromiso (Bello, s.f.).

El histórico de frecuencias fundamentales y los instantes futuros también se aprovechan para elegir entre los candidatos siguiendo asimismo el TWMe (ver Cano, 1998), que tiene una evolución suave entre muestras vecinas. La fase de los picos puede ser útil finalmente para modificar la búsqueda entre frecuencias fundamentales candidatas. En este caso, se suaviza el contorno de altura de acuerdo con las propiedades del sonido, al contrario que con las técnicas de mediana.

Seguimiento de la frecuencia fundamental empleando otras fuentes de información. También podemos utilizar otros metadatos disponibles para dirigir la detección de la frecuencia fundamental y el proceso de seguimiento. Se ha usado la información acerca del contenido elaborando modelos internos para caracterizar la fuente sonora (Kashino, et al., 1995). Martín (1996) también utilizó reglas musicales para transcribir piezas polifónicas de piano a cuatro voces. Cuando se realiza alguna hipótesis acerca del tipo de señal o cuando el algoritmo se adapta a alguna propiedad de la señal, también estamos sacando provecho de una información que puede considerarse de contexto.

Por ejemplo, podemos emplear algunos métodos de pre-procesado en función del instrumento que interpreta la pieza. Goto considera los diferentes rangos de altura de la melodía en contraposición con los de la línea del bajo y decide entre ambas clasificaciones aplicando diferentes filtros paso banda (Goto, 1999, 2000). Otra posibilidad sería adaptar algunos de los parámetros de un algoritmo

al instrumento en cuestión. Algunas de estas ideas son la base de ciertos detectores de altura multitímbricos (Goto 1999, 2000; Anderson, 1997).

4.1.2. Estimación polifónica de altura

Se admite en general que los métodos de estimación monofónica de altura no son apropiados para aplicarlos al caso polifónico (Kostek, 2004), aunque algunos de los algoritmos utilizados en detección monofónica de altura pueden adaptarse a situaciones polifónicas simples. En Anderson (1997) se describe cómo algunos métodos aplicados a la estimación monofónica de frecuencia fundamental pueden adaptarse a la polifonía. También el procedimiento TWMe puede extenderse a separación por pares, tal y como se explica en Maher y Beauchamp (1993), tratando de encontrar las dos frecuencias fundamentales que mejor definan los picos espectrales medidos.

La estimación polifónica está orientada al análisis del escenario de la audición y la separación de fuentes: si un algoritmo puede encontrar la altura de un sonido sin que otros sonidos simultáneos confundan al sistema, la información de altura puede usarse para separar los parciales de un sonido de la mezcla. De hecho, la mayoría de los métodos de estimación polifónica de altura que han tenido éxito aplican estos principios aprendidos de la organización que demuestra la audición humana.

Kashino, et al., (1995) implementaron estos principios en una red probabilística bayesiana, en la que el análisis *bottom-up* de la señal podía integrarse con las predicciones temporales y musicales. Un ejemplo reciente que sigue los mismos principios es el de Walmsley, et al., (1999), que utilizaron un marco probabilístico bayesiano para estimar conjuntamente los parámetros del modelo armónico para un cierto número de muestras. Otros enfoques estadísticos incluyen el trabajo de Davy y Godsill (2003) y Cemgil (2004). Godsmark y Brown (1999) desarrollaron un modelo que era capaz de extraer líneas melódicas de música polifónica a través de la integración de diversas fuentes de información. El sistema propuesto por Goto (1999, 2000) es más pragmático, y es capaz de detectar melodía y líneas de bajo en grabaciones polifónicas reales basándose en la hipótesis de que ambas se encuentran en regiones diferentes del espectro. En Klapuri (2000a) se reseñan otros métodos y se describe un sistema siguiendo un método iterativo enfocado a la separación. Este algoritmo es razonablemente preciso para un amplio rango de frecuencias fundamentales y variedad de fuentes sonoras.

Los estimadores polifónicos de altura de última generación son razonablemente precisos sobre señales limpias, incrementándose progresivamente el error a nivel de muestra del 2%, para señales a dos voces, al 20% de muestras equivocadas en fragmentos a cinco voces. En todo caso, la eficiencia se reduce de forma significativa en presencia de ruido, y el número de voces que concurren no suele detectarse correctamente. Además, si se quiere obtener una estimación fiable de la altura en un contexto polifónico es necesario utilizar ventanas de análisis más grandes (de en torno a 100 ms) que en la estimación monofónica (Klapuri, 2000a; Kostek, 2004).

Recientemente, algunos sistemas centrados en la estimación de la *altura predominante* han sido evaluados en el contexto de las conferencias del ISMIR en 2004 y 2005. El objetivo de esta evaluación ha sido comparar algoritmos de última generación que realicen la estimación de la altura predominante de señales polifónicas. En Gómez y otros (2006) se presenta un resumen de los resultados de la evaluación. De acuerdo con estos experimentos, la precisión global se encuentra en torno al 71%. Este rendimiento puede, no obstante, reducirse al tratar de estimar múltiples voces en el caso de una señal polifónica. El error de estimación afecta entonces a la precisión de las subsiguientes descripciones tonales de alto nivel que puedan realizarse tras la estimación polifónica de altura.

4.2. Transcripción y descripción tonal

De acuerdo con el estado actual de la tecnología en la tarea de transcripción automática a partir de audio, todavía no es posible conseguir una representación fiable de una pieza musical en formato de audio. Se plantea la pregunta de si es realmente necesario obtener esta representación. Una de las hipótesis de partida de Gómez (2006a) es que no es en absoluto necesario disponer de una transcripción para obtener una descripción tonal completa a distintos niveles, incluyendo las distribuciones de clases de alturas, acordes y tonalidades, y asimismo permitiendo el procesamiento de similitud tonal y la organización de colecciones musicales en función de la tonalidad. Numerosos autores se han basado en esta idea, tal y como se expone en la sección 4.3.

Como se muestra en Gómez (2006a), las características de distribución de PC pueden extraerse de las señales de audio sin pérdida alguna de precisión debido a la estimación. El esquema seguido en este enfoque se presenta en la Figura 4.5. El uso de características del PCP solventa el problema de la transcripción automática, al ocuparse de la descripción de los aspectos tonales sin necesidad de disponer de una transcripción perfecta. Por otro lado, introduce un problema adicional al respecto de la aplicación de los modelos tonales a estas nuevas representaciones, pues han sido diseñados para trabajar con una representación en partitura.

Finalmente, la tonalidad estimada podría usarse para mejorar el proceso de transcripción automática. Por ejemplo, la información de tonalidad o escala podría emplearse para dar mayor peso a los valores de frecuencia fundamental que pertenezcan a la escala. La distinción entre transcripción y descripción ya fue señalada en Herrera (2006) para diferentes aspectos del sonido.

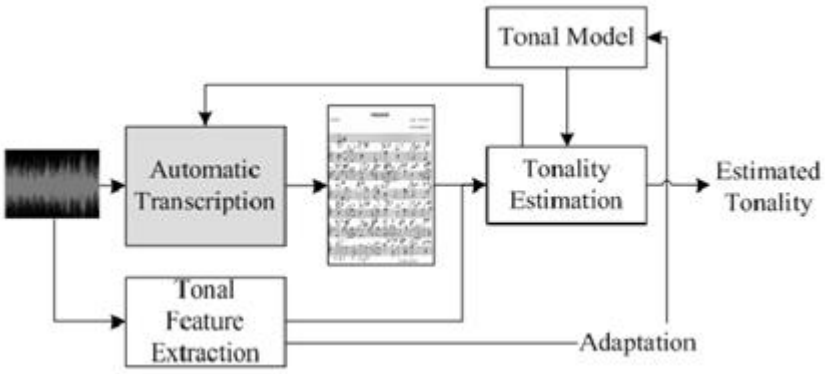


Figura 4.5. Esquema del enfoque adoptado en Gómez (2006a), en el que no es necesario disponer de una transcripción de la grabación para obtener una descripción tonal de la pieza en formato de audio.

4.3. Características de distribución de croma

Se han dedicado muchos esfuerzos al análisis de secuencias de acordes y de la tonalidad en representaciones MIDI de piezas de música clásica, pero en cambio pocos autores han tratado directamente con la señal de audio o han analizado piezas de otros géneros. El uso de métodos orientados al MIDI haría necesario un paso previo para la transcripción automática de audio polifónico, tarea que ha demostrado ser realmente dificultosa.

Como ya se ha mencionado, el hecho de no disponer de sistemas de transcripción automática provoca que muchos trabajos se orienten directamente a manejar grabaciones en formato de audio, con el fin de extraer información de la distribución de croma de la música. Ésta tiene, en cierta forma, una relación directa con los acordes y la tonalidad de la pieza bajo estudio. Los acordes pueden reconocerse a partir de la distribución de croma sin haber detectado de forma precisa qué notas están sonando. La tonalidad puede también estimarse a partir de la distribución de croma sin un procedimiento previo de detección de acordes. Para garantizar la fiabilidad de los descriptores de distribución de croma se deben satisfacer los siguientes requerimientos:

1. Representar la distribución de croma tanto de señales monofónicas como polifónicas.
2. Considerar la presencia de armónicos. Los primeros armónicos de un tono complejo pertenecen a la tonalidad mayor definida por la nota musical de la frecuencia fundamental, y todos excepto el 7º armónico pertenecen a su tríada tónica.
3. Robustez frente al ruido coexistente: ruido ambiental (por ejemplo, el de grabación en directo), sonidos percusivos, etc.
4. Independencia del timbre y el instrumento utilizado, de forma que la misma pieza tocada en diferentes instrumentos tenga la misma descripción tonal.
5. Independencia de sonoridad y dinámica.

6. Independencia de la afinación, de modo que la frecuencia de referencia pueda ser diferente del LA_4 estándar a 440 Hz.

Todas las propuestas para procesar la evolución instantánea de las distribuciones de croma siguen el mismo esquema, mostrado en la Figura 4.6. Además, apuntamos que es necesario un procesamiento adicional de los descriptores instantáneos cuando se pretende describir un segmento mayor de una pieza.

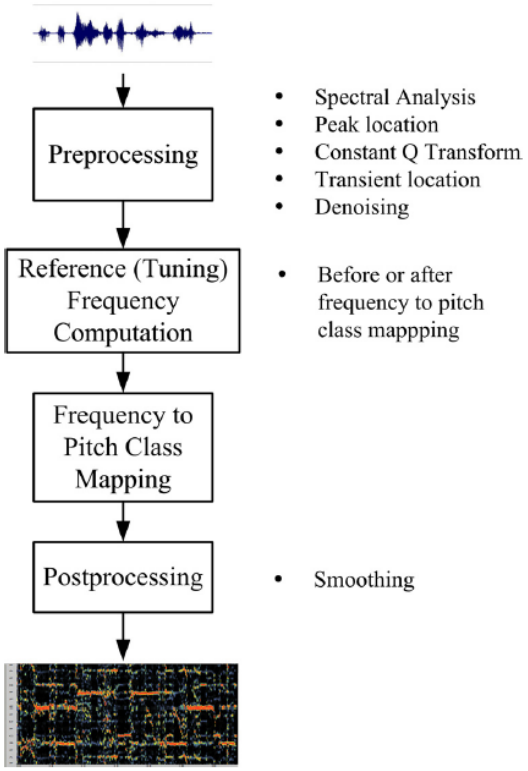


Figura 4.6. Diagrama de bloques general para métodos de procesado de distribución de croma en audio (Gómez, 2006a).

Se resumen en la Tabla 4.2 las diferentes propuestas para cada una de las etapas del esquema general de extracción de información tonal del audio. La nomenclatura de este tipo de características es realmente variada. La primera incursión en la inducción de tonalidad a partir de audio fue propuesta por Leman (1991, 1995b, 2000); Martens, et al., 2004). El enfoque adoptado por Leman (2000) consiste en extraer un conjunto de *patrones de altura* siguiendo un procedimiento similar a algunos métodos de estimación polifónica ya mencionados en la sección 4.1.2, empleando un modelo del sistema de audición humana. El uso de *histogramas de altura* aparece posteriormente en Tzanetakis (2002). Fujishima (1999) propuso un sistema de reconocimiento de acordes basado en el perfil de croma (en adelante PCP, *Pitch-Class Profile*), una característica de bajo nivel. El PCP, tal

como lo definió Fujishima, es un vector de doce dimensiones que representa las intensidades de los doce semitonos del conjunto cromático. Su sistema de reconocimiento de acordes compara este vector con una colección de acordes tipo expresados igualmente como patrones de intensidades de croma para estimar qué acorde está sonando. En Gómez (2006a) se propone una extensión del PCP denominada perfil armónico de croma o HPCP (*Harmonic Pitch-Class Profile*), que utilizaremos en este proyecto. Los perfiles de Q constante también se han empleado para caracterizar el contenido tonal del audio: el primero, de nombre homónimo, definido por Purwins, et al., (2000) y el llamado perfil de alturas por Zhu, et al., (2005). Otra herramienta propuesta recientemente es el cromagrama (Pauws, 2004), que también ha sido utilizado para estimación de tonalidad.

4.3.1. Pre-procesado

La tarea principal de la etapa de pre-procesado es preparar la señal para la descripción de su distribución de croma y enfatizar los rasgos más relevantes para este tipo de herramientas. El pre-procesado puede ayudar a satisfacer el tercer requerimiento antes mencionado, proporcionar robustez frente al ruido: ruido ambiental, sonidos percusivos, sonidos del habla, etc.

Todas las propuestas analizadas en la literatura están basadas en realizar un análisis frecuencial sobre la señal de audio. En el trabajo de Fujishima (1999, 2000), el sonido de entrada se transforma a un espectro DFT (definiendo ventanas de análisis de 2048 muestras a 5.5 kHz, con una duración de 400 ms). La DFT también se utiliza en la propuesta de Gómez (ver también Gómez, 2004, 2006; Gómez y Herrera, 2004), y en Pauws (2004); Chuan y Chew (2005); Izmirli (2005), utilizando una duración de ventana similar.

También es común restringir el análisis a una región frecuencial dada, con el fin de eliminar frecuencias no audibles y seleccionar las frecuencias más relevantes para los descriptores de distribución de croma. Fujishima (1999) selecciona una región espectral ente 63.5 y 2032 Hz. En el trabajo de Gómez (2006a) se emplea también una sola región de la señal entre 100 Hz y 500 Hz, Pauws (2004) utiliza frecuencias entre 25 y 5000 Hz, Zhu, et al., (2005) entre 27.5 y 3520 Hz, Chuan y Chew (2005) entre 32 y 1975 Hz y Izmirli (2005) de 50 a 2000 Hz.

En lugar de escoger una única banda de frecuencias, algunos métodos consideran un enfoque multibanda. En Leman (2000), un modelo de audición descompone la señal de entrada en diferentes bandas de frecuencia representadas como patrones de nervios, también conocidos como imagen nerviosa de la audición. Se hace uso de quince canales con frecuencias centrales entre 141 y hasta 5173 Hz (ver Martens, et al., 2002).

La transformada de Q constante fue presentada por Brown (1991), y Brown y Puckette (1992) propusieron más tarde un algoritmo para su ejecución eficiente. Se ha utilizado esta transformada en los trabajos de Purwins, et al., (2000) y más tarde por Zhu, et al., (2005) para descripción tonal de

bajo nivel. La transformada de Q constante puede verse como un banco de filtros con frecuencias centrales f_i geométricamente espaciadas. La letra "Q" está relacionada con la relación constante entre la frecuencia central y el ancho de banda de cada filtro (en ocasiones se denomina a esta relación factor de calidad, *Quality factor*). Las frecuencias f_i se definen como:

$$f_i = f_{ref} \cdot 2^{\frac{i}{M}}$$

y la relación constante frecuencia central-ancho de banda como:

$$Q = \frac{f_i}{\Delta_i} = \left(2^{\frac{1}{M}} - 1\right)^{-1}, \quad i = 0, \dots, M-1$$

donde M determina el número de filtros o componentes por octava (12, 24, 36, etc.). Este parámetro se fija a $M = 120$ (resolución de 0.1 semitonos) in Zhu, et al., (2005), con el fin de mejorar el cálculo de la frecuencia de afinación. El procesado de la transformada Q radica en que, para cada componente frecuencial calculada, se escoge un tamaño de ventana N_i diferente. Para valores enteros de Q, la componente i-ésima de la transformada de Q constante es idéntica a la componente Q-ésima del espectro (DFT) con un tamaño de ventana de:

$$N_i = Q \cdot \frac{f_s}{f_i}$$

La ejecución de la transformada Q se estructura como sigue: en primer lugar se escoge la frecuencia más pequeña f_{ref} y el número de bandas por octava (M) y se consideran las ecuaciones de Q y N_i. Entonces, la i-ésima componente de la transformada de Q constante $QTX(i)$ es igual a la componente Q-ésima de la DFT $X_{N_i}(Q)$ considerando una longitud de ventana de N_i, y se define como:

$$QTX(i) = X_{N_i}(Q) = \frac{1}{N_i} \sum_{n<N_i} x[n] \cdot w_{N_i}[n] \cdot e^{-j2\pi n \frac{Q}{N_i}},$$

donde $x[n]$ es la señal muestreada en el dominio del tiempo y $w_{N_i}[n]$ es la ventana de análisis escogida. Este método no es muy eficiente computacionalmente, ya que deben realizarse varias FFT.

Además de un análisis frecuencial de la señal de entrada, Fujishima (1999) menciona otras etapas de pre-procesado, incluyendo el escalado no lineal y la detección del silencio y el ataque para evitar obtener características ruidosas; no las explica ni evalúa en mayor profundidad en dicho trabajo. La detección de transitorio se utiliza en Gómez (2006b) como etapa de pre-procesado.

Otra técnica de pre-procesado interesante para reducir el ruido es la inclusión de una rutina de selección de picos considerando únicamente los máximos locales del espectro (Gómez, 2004, 2006b). Chuan y Chew (2005) también presentan este procedimiento. Finalmente, Pauws (2004) también

menciona un procedimiento para realzar las componentes espectrales con el fin de cancelar picos espurios, aunque en su artículo tampoco explica este procedimiento. Cho y Bello (2014) hacen una evaluación de los diferentes métodos de pre-procesado en el contexto de los sistemas de detección acordes.

4.3.2. Extracción de la frecuencia de referencia

Tal y como explicábamos en la sección 3.5, la nota LA_4 a 440 Hz está considerada como la frecuencia de referencia estándar para la definición de las clases de alturas. De acuerdo con esto, la mayoría de los trabajos que buscan extraer la distribución de croma utilizan una frecuencia fija como referencia (Fujishima, 1999; Purwins, et al., 2000; Pauws, 2004; Martens, et al., 2002; Chuan y Chew, 2005; Izmirli, 2005), que normalmente se establece siguiendo el criterio que comentamos.

Sin embargo, no podemos suponer que las orquestas y bandas siempre estarán afinadas a esta frecuencia. Algunas piezas pueden no estar afinadas a 440 Hz, de modo que hemos de estimar la frecuencia de referencia de la afinación con el fin de que el sistema sea robusto frente a eventualidades en la afinación (el sexto requerimiento de la lista que presentábamos). Este procedimiento puede ejecutarse bien antes o bien después de la extracción de la distribución de croma, y el vector de características final debe ajustarse a esta frecuencia de referencia.

Fujishima (2000) incluyó un procedimiento para ajustar los valores del PCP a la frecuencia de referencia después la obtención del mismo. En este procedimiento, el perfil de octava, obtenido empleando una resolución frecuencial de 1 centésima, se divide en doce partes iguales de un semitono de ancho, y estas doce partes conforman un perfil semitonal, que se encuentra desplazado. Para cada uno de los perfiles semitonales se obtienen los valores de media y varianza. La mínima varianza identifica la posición del pico buscado, y se obtiene la frecuencia de referencia utilizando el valor de la media para esta posición y la cantidad de desplazamiento del perfil semitonal.

El procedimiento para determinar la frecuencia de la afinación propuesto por Zhu, et al. (2005) se realiza previamente a la extracción de la distribución de croma, y es empleado como frecuencia de referencia para realizar la asignación de alturas a partir de frecuencias logarítmicas. El método está basado en el análisis estadístico de las posiciones en frecuencia de los picos prominentes de la transformada de Q constante, y se compone de cuatro etapas:

- Detección de máximos locales de la transformada de Q constante que tengan un nivel de energía superior a un umbral
- Agrupación de los picos detectados de acuerdo a sus desviaciones (módulo 12) respecto de la frecuencia de referencia de 440 Hz
- Elección del grupo más grande para representar la frecuencia de afinación en la ventana de análisis dada, debido a que su valor sea lo suficientemente prominente

- Construcción de un histograma de las frecuencias de afinación de todas las ventanas de análisis y elección del máximo punto como frecuencia de referencia

La frecuencia de referencia se extrae con una resolución de 0.1 semitonos (10 centésimas, un orden de magnitud más grande que en el caso de Fujishima), y tan solo se necesita un fragmento corto de una pieza (30 segundos) para estimar la frecuencia de afinación, en base a la suposición de que la frecuencia de referencia normalmente no varía a lo largo de la pieza. La propuesta de Gómez (2006a) emplea un procedimiento similar para la determinación de la frecuencia de afinación. Respecto a métodos de pre-procesado, no estamos al tanto de la existencia de evaluaciones formales de métodos orientados a la extracción de la frecuencia de referencia.

4.3.3. Determinación de la frecuencia y asignación de valores a las alturas

Una vez se conoce la frecuencia de referencia y la señal se convierte en un espectrograma por medio de una DFT o un análisis de Q constante, existe un procedimiento para determinar los valores de las diferentes clases de alturas a partir de los valores de frecuencia.

Los trabajos de Leman (2000) y Tzanetakis (2002) estuvieron orientados a la estimación polifónica de altura, y aplican un análisis de la periodicidad a la salida del banco de filtros empleando la función de autocorrelación, con el fin de extraer un conjunto de K frecuencias predominantes f_{pk}, donde $k = 1, \dots, K$, que son las utilizadas en la descripción tonal. Leman (2000) utiliza entonces las frecuencias predominantes y las compara con el conjunto cromático valiéndose de una correspondencia logarítmica con respecto a la frecuencia de referencia:

$$PCD(n) = \sum_{f_{pk} \;\; s.t. \;\; M(f_{pk})=n} 1 \quad ,$$

donde PCD representa la distribución de características de croma (*Pitch Class Distribution*). En esta ecuación, $n = 1, \dots, 12$, f_{pk} representa las frecuencias predominantes y $M(f_{pk})$ es una función que correlaciona el valor de frecuencia con el índice de la PCD, siguiendo una correspondencia logarítmica:

$$M(f_{pk}) = round\left(12 \cdot \log_2\left(\frac{f_{pk}}{f_{ref}}\right) mod\ 12\right) \quad ,$$

donde f_{ref} es la frecuencia de referencia que se encuentra en $PCD(0)$. Aunque la nota de clase DO es asignada comúnmente a esta posición, consideramos aquí la clase LA, de forma que f_{ref} sería 440 Hz si la pieza estuviera afinada con referencia a este valor.

En lugar de utilizar tan solo alturas predominantes, Fujishima (1999) considera todas las frecuencias de la DFT, donde el peso que aporta cada frecuencia a la clase de altura correspondiente se obtiene como el cuadrado del espectro de amplitud:

$$PCD(n) = \sum_{i \ s.t. \ M(i)=n} |X_N(i)|^2$$

donde $n = 1, \dots, 12$, $|X_N(i)|$ es la amplitud en escala lineal de la componente espectral i, $i = 0, \dots, N/2$ donde N es el tamaño de la DFT. $M(i)$ es, pues, una tabla que correlaciona los coeficientes espectrales con sus índices del PCP, siguiendo una distribución logarítmica:

$$M(i) = \begin{cases} -1 & si \ i = 0 \\ round\left(12 \cdot \log_2\left(\frac{\frac{f_s \cdot i}{N}}{f_{ref}}\right) mod \ 12\right) & si \ i = 1,2, \dots, \frac{N}{2} \end{cases}$$

donde f_s es la frecuencia de muestreo, f_{ref} es de nuevo la frecuencia de referencia que se encuentra en $PCD(0)$. El término $\frac{f_s \cdot i}{N}$ representa la frecuencia de la componente i del espectro.

Otros enfoques como los de Purwins, et al., (2000); Chuan y Chew (2005); Izmirli (2005) no consideran el cuadrado $|X_N(i)|^2$ como coeficiente de cada frecuencia sino $|X_N(i)|$:

$$PCD(n) = \sum_{i \ s.t. \ M(i)=n} |X_N(i)|$$

En Gómez (2006a) se presenta un esquema de asignación de pesos que emplea una función coseno. Pauws (2004) también presenta una ponderación dependiente de la posición en frecuencia (siguiendo una función exponencial decreciente) y de la sonoridad percibida (función arcotangente). Se define la contribución mediante un procedimiento de probabilidad máxima.

Izmirli (2005) propone añadir una etapa de filtrado utilizando la SFM (*Spectral Flatness Measure*). El espectro está dividido en bandas de frecuencia de media octava y por cada una se calcula la SFM. Sólo aquellas bandas que posean un valor superior a 0.6 de esta medida (esto es, que contengan información relevante sobre los picos) se añaden al vector PCD.

Los métodos de obtención de la PCD deberían considerar la presencia de armónicos. El método presentado en Gómez (2006a) lo hace cuando se adapta el modelo tonal empleado, al igual que en Izmirli (2005). Pauws (2004) también considera la presencia de armónicos en cada sonido, considerando hasta 15 armónicos por cada clase de altura que contribuye en el cálculo. Zhu, et al.,

(2005) también consideran este efecto mediante un método de filtrado, denominado filtrado de consonancia, tras aplicar la transformada Q. En esta etapa, extraen tan solo los parciales consonantes, de acuerdo con la escala diatónica. Esto podría también considerarse una etapa de post-procesado, dado que no tienen en cuenta los armónicos durante la extracción de la PCD. Cuando se considera la presencia de armónicos nos acercamos al enfoque de la transcripción automática, como el adoptado en Leman (2000) o en Tzanetakis (2002). En cuanto a otras etapas, no estamos al corriente de que exista una evaluación sistemática de los diferentes esquemas de ponderación.

4.3.4. Resolución interválica

Un parámetro importante para los descriptores de distribución de croma es la resolución en frecuencia utilizada. Un semitono (12 bandas PCD, 100 centésimas) es la resolución que normalmente se escoge para representar la distribución de croma, tal y como vemos en Pauws (2004); Chuan y Chew (2005); Izmirli (2005).

Incrementar esta resolución puede ser beneficioso para mejorar la robustez frente a variaciones en la afinación y otras oscilaciones en frecuencia. Fujishima define 12 bandas para la estimación de acordes, aunque emplea una resolución de 1 centésima (1200 valores) en las primeras etapas del algoritmo para realizar el plegamiento de frecuencias y revisar la afinación. Este es también el caso de Zhu, et al., (2005), donde se escoge una resolución interválica de 1 semitono (12 valores) para generar el perfil de alturas en la tarea de estimación de la tonalidad, aunque se fija la resolución a 10 centésimas (120 valores) para determinar la frecuencia de afinación.

Igualmente, Purwins, et al., (2000) y Gómez (2004, 2006b) definen 36 valores (1 tercio de semitono) para mejorar la resolución de la PCD. La resolución suele rebajarse a 12 valores cuando se compara con un modelo tonal. Para obtener estos 12 valores suelen sumarse todas las amplitudes en la región de cada semitono (como en Fujishima, 2000).

4.3.5. Métodos de post-procesado

Como es habitual en los métodos de estimación de frecuencia fundamental, algunos métodos de post-procesado se aplican tras extraer la distribución de croma.

Uno de los requerimientos mencionados al principio de este capítulo para las características de distribución de croma es la robustez frente a variaciones en la dinámica. Se suele conseguir esto con un proceso de normalización. Gómez (2004, 2006b) propone normalizar el vector PCD dividiendo cada componente por su valor máximo. Chuan y Chew (2005) también normalizan por el valor máximo y la suma de todos los valores del segmento completo del vector PCD. Esta normalización también puede estar implícita en el proceso de extracción de características o incluida como etapa de post-procesado.

Leman (2000) propone sumar a cada vector futuro (al que llama imagen) una pequeña cantidad del vector antiguo, especificada por un tiempo de decaimiento medio. Este es el tiempo que tarda la amplitud de una señal impulsiva en decaer hasta la mitad de su valor original (Martens, et al., 2004).

Fujishima (2000) propone un procedimiento de realce de picos: en primer lugar, el PCP se desplaza una cantidad de semitonos n ($n = 1, \dots, 11$), y se obtiene la correlación entre el PCP original y la versión desplazada para cada valor de n. Todos estos perfiles correlacionados se combinan para obtener un PCP mejorado. De acuerdo con Fujishima (2000), este perfil tiene picos más agudos y prominentes que un PCP normal. Algunos autores comentan que este procedimiento puede omitirse por simplicidad, pero no se ha realizado ninguna evaluación al respecto para probar su utilidad.

Otras ideas mencionadas en Fujishima (1999), aunque no desarrolladas, incluyen el suavizado (promedio) y la detección de cambio de acorde (monitorizar la dirección del vector PCP). Izmirli (2005) también propone un proceso de suma en segmentos de 5 segundos.

4.3.6. Descriptores de segmentos

La distribución de croma, como medida instantánea, se ha utilizado para estimación de acordes, como en el sistema descrito por Fujishima (1999). La mayoría de los trabajos en torno a la estimación de la tonalidad global se basan en la acumulación de la distribución instantánea de croma en un vector global, como en Purwins, et al., (2000); Martens, et al., (2002); Gómez (2004, 2006b); Pauws (2004); Zhu, et al., (2005). En la tabla 4.2 mostramos un resumen de estos trabajos. También encontramos en Zhu, et al., (2005) un procedimiento de normalización tal que la suma de los valores resulta en la unidad.

Method	Pre-processing	Reference Frequency Computation	Frequency to Pitch Class Mapping	PCD Resolution	Post-processing	Segment description	Application
Pitch patterns Leman (2000)	Filter bank (141-5173 Hz), periodicity analysis (correlation)	No	Equal weight	Frequency analysis	Echoic memory module	Sum	Key induction
PCP Fujishima (2000)	DFT (63.5-2032 Hz), non-linear scaling, silence and attack detection	PCP shifting procedure	Square of spectral magnitude	$\frac{1}{2}$ tone (1 cent for tuning frequency)	Peak enhancement, smoothing (average), chord change sensing	None	Chord recognition
Constant-Q profiles Purwins et al. (2000)	Constant Q transform	No	Spectral magnitude	$\frac{1}{6}$ tone	None	Sum	Key estimation and tracking of classical music, comparative analysis of composers
Chromagram Pauws (2004)	DFT (25-5000 Hz), spectral peaks enhancement	No	Likelihood, Spectral magnitude, 15 harmonics (decreasing contribution), arc-tangent weighting according to loudness	$\frac{1}{2}$ tone (12 bins)	None	Sum and normalization	Key estimation from piano classical music

Pitch profile Zhu et al. (2005)	Constant Q transform (27.5-3520 Hz)	Analysis of peak deviations	Equal weight	$\frac{1}{2}$ tone (10 cents for tuning frequency)	None	Consonance filtering	Key estimation
Pitch class and strength Chuan and Chew (2005)	DFT (32-1975 Hz), peak estimation	No	Spectral magnitude	$\frac{1}{2}$ tone	Normalization	None	Key recognition from audio generated from MIDI, classical
Chroma summary vector Izmirli (2005)	DFT (50-2000 Hz), filtering based on SFM over half octave bands	No	Spectral magnitude	$\frac{1}{2}$ tone	Summation of 5 seconds windows	Sum	Key estimation from audio, classical
HPCP Gómez (2006, 2004), Chapter 3	DFT (100-5000 Hz), transient detection	Analysis of peak deviations	Square of spectral magnitude and weighting scheme	$\frac{1}{6}$ tone (frequency resolution for tuning frequency)	Normalization	Sum	Key estimation

Tabla 4.2. Cuadro resumen de los diferentes métodos para obtener la distribución cromática de alturas a partir de audio.

Capítulo 5

Diseño del sistema de estimación automática de acordes

En este capítulo describiremos el sistema que hemos desarrollado, sus componentes específicos y las herramientas utilizadas, así como la colección de audio sobre la que hemos evaluado el sistema y los resultados obtenidos.

En primer lugar, siguiendo el orden lógico del diagrama de bloques que veremos que tiene nuestro sistema ACE, hablaremos de la obtención de las características de croma de las señales y las herramientas para ello usadas, Sonic Annotator y Sonic Visualiser, en las secciones 5.1 y 5.2. En la sección 5.3 veremos las características de los plug-ins de cromagrama utilizados y su evaluación comparada.

La sección 5.4 servirá para dar cuenta de qué colecciones de datos de puesta a prueba hemos considerado más interesantes para evaluar el sistema y en la sección 5.5 hablaremos del vocabulario armónico que se ha escogido abarcar, en base a un análisis de la problemática asociada a la detección de acordes a partir de audio.

Dedicaremos la sección 5.6 a describir la versión inicial del sistema de detección de acordes diseñado: el diagrama de bloques del prototipo y sus diferentes módulos (definición de patrones, similitud y decisión, principalmente), así como las decisiones de diseño que se han tomado y su justificación. Obviamos en este punto la extracción de características de croma por haberla cubierto ya pormenorizadamente en 5.1 y 5.3.

La versión final del sistema la presentaremos en el apartado 5.7. Hablaremos de las novedades que hemos introducido a raíz de observaciones en el desempeño del prototipo, los conceptos en los que se basan las nuevas características (entre otras cosas, tareas de pre y post-procesado) y cómo afectan éstas a la tarea de estimación de acordes. Hemos tenido que realizar una última revisión bibliográfica para dar cobertura técnica al problema, nos referiremos a estos nuevos autores de manera resumida.

El método de evaluación para medir la eficacia del sistema diseñado será establecido en la sección 5.8, en base a estrategias de evaluación anteriormente aplicadas a extracción de información musical en las convocatorias del MIREX.

Finalmente, hablaremos en la sección 5.9 de algunos problemas de implementación que hemos encontrado en el desarrollo del sistema y que no han sido cubiertos en otros apartados.

5.1. Sonic Annotator

Sonic Annotator es una herramienta de extracción sistemática de características de ficheros de audio. Es el medio para generalizar los procesos que también realiza Sonic Visualiser para poder aplicarlos a miles de archivos de una sola vez. Puede aplicar las transformaciones de la señal (de cualquiera de los Vamp plug-ins instalados) tanto a ficheros en local como a ficheros en un servidor remoto HTTP o FTP. Se ejecuta en línea de comandos y su uso es realmente sencillo.

5.1.1. Vamp plug-ins

Los algoritmos de análisis frecuencial que emplearemos para obtener los cromagramas de las señales se basan en un sistema de procesamiento de audio llamado Vamp, que sirve como plataforma para la ejecución de diferentes plug-ins (Figura 5.1). Éstos son programas de terceros que realizan una tarea específica, asociada generalmente a la extracción de información descriptiva de ficheros de audio.

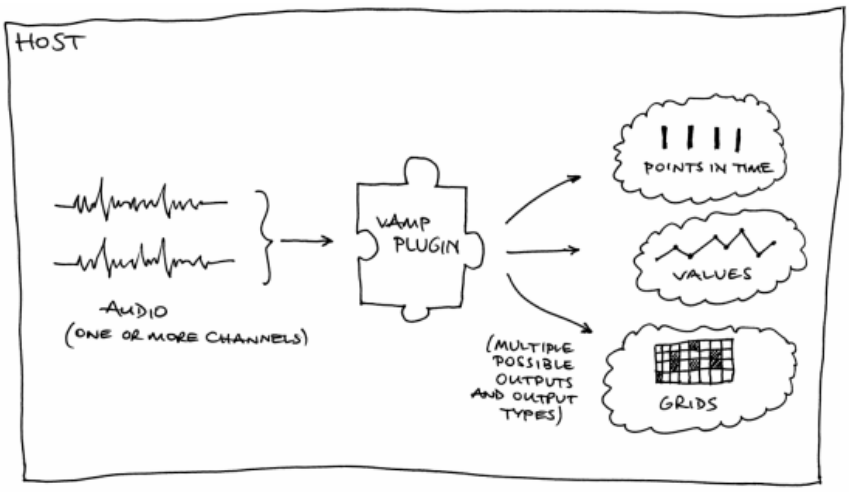


Figura 5.1. Diagrama de bloques que presenta el funcionamiento básico del sistema Vamp (www.vamp-plugins.org).

Al igual que los plug-ins de efectos (como los VST), un plug-in Vamp es un módulo software que puede lanzarse integrado en una aplicación *host*. Los VST pueden recibir y devolver información de audio. Sin embargo, a diferencia de los plug-ins de efectos, un plug-in Vamp no genera otro fichero de audio, sino algún tipo de información simbólica.

Entre las cosas que pueden calcular se incluyen la localización de los *onsets* o tiempos de inicio de las notas, representaciones visuales del audio como los espectrogramas, o curvas de densidad espectral de potencia o seguimiento de frecuencia fundamental. En nuestro caso, nos interesa calcular

los cromagramas de las señales de audio de la colección de ficheros de prueba que hemos confeccionado.

5.1.2. Interfaz de usuario

La ejecución de Sonic Annotator se realizará a través de la consola de comandos de Windows. En primer lugar, hemos de instalar los Vamp plug-ins necesarios. Podemos listar los plug-ins disponibles mediante la orden siguiente (ver Figura 5.2):

```
sonic-annotator.exe -l
```

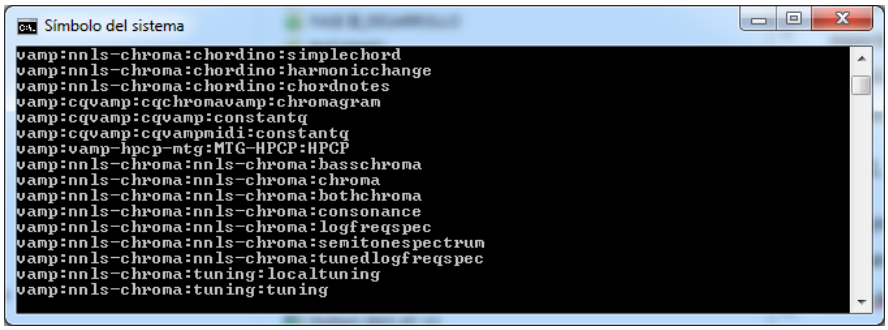


Figura 5.2. Lista de Vamp plug-ins instalados.

Para ejecutar un plug-in, primero hemos de extraer su fichero de configuración, que modificaremos para establecer los parámetros de la transformación antes de realizarla. Extraemos dicho fichero con la siguiente acción:

```
sonic-annotator.exe -s nombre_completo_del_plugin > conf.n3
```

el cual volcará la información de configuración en un fichero llamado `conf.n3`, que podremos modificar con cualquier editor de texto (en la siguiente sección veremos esto en más detalle).

A continuación tenemos dos opciones:

- iniciar la transformación sobre un único archivo de audio (Figura 5.3),
- o aplicarla recursivamente a todos los archivos de una ruta especificada.

En el primer caso, la orden será:

```
sonic-annotator.exe -t conf.n3 nombre_del_audio.ext -w csv
```

y en el segundo, para explotar la función de sistematización:

```
sonic-annotator.exe -t conf.n3 C:/ruta_local -w csv -r
```

Nótese que estamos empleando el fichero de configuración del plug-in que se ha seleccionado, modificado por nosotros, y le estamos pidiendo a Sonic Annotator que la salida sea en formato .csv (*comma separated vector*).

Al ejecutarse la transformación, la consola nos mostrará generalmente una lista con los parámetros implicados en ella y un informe de errores (Figura 5.3), para facilitar al usuario la corrección de cualquier anomalía que pudiera darse en la ejecución.

Figura 5.3. Aplicación de un plug-in de extracción de características de croma sobre un fichero de audio. En línea de comandos: parámetros del plug-in e informe de errores.

5.1.3. El fichero de configuración

Editar el fichero de configuración del plug-in implica fijar los parámetros involucrados en la transformación (Figura 5.4). Por tanto, hemos de estar familiarizados con los conceptos descritos en esta memoria en los Capítulos 2, 3 y 4, en especial este último.

En primer lugar, nos interesa saber cuál es el intervalo temporal Δt a emplear, esto es, cada cuánto se calcularán las muestras del cromagrama. Dependiendo del uso que queramos hacer de la información de croma, tendrá más o menos sentido definir un intervalo de análisis fino. Recordemos que la música opera a una escala de tiempo diferente en función de si hablamos de la detección de las notas individuales (desde 50 ms) o del ritmo armónico (el tiempo medio de cambio de acorde, grosso modo, se sitúa en torno a 2 s). El intervalo temporal es dependiente de la frecuencia de muestreo f_s y el salto de la ventana. Para una frecuencia estándar de $f_s = 44.1$ kHz y un salto de 2048 muestras:

$$\Delta t = \frac{step\_size}{f_s} = \frac{2048}{44100} \cong 0.0464\ s$$

Esta precisión es de menos de una décima de segundo, más que suficiente en nuestro caso. Modificando el salto de ventana a 256 muestras obtendríamos 8 veces más *frames* en el cromagrama, pero carece de sentido en el contexto de la segmentación armónica, pues no se producirán cambios relevantes en esos intervalos de tiempo.

```
@prefix xsd:      <http://www.w3.org/2001/XMLSchema#> .
@prefix vamp:     <http://purl.org/ontology/vamp/> .
@prefix :         <#> .

:transform a vamp:Transform ;
    vamp:plugin <file:///C:/Program%20Files%20(x86)/Vamp%20Plugins/vamp-hpcp-mtg#MTG-HPCP> ;
    vamp:step_size "2048"^^xsd:int ;
    vamp:block_size "4096"^^xsd:int ;
    vamp:parameter_binding [
        vamp:parameter [ vamp:identifier "HF" ] ;
        vamp:value "5000"^^xsd:float ;
    ] ;
    vamp:parameter_binding [
        vamp:parameter [ vamp:identifier "HarmonicsPerPeak" ] ;
        vamp:value "8"^^xsd:float ;
    ] ;
    vamp:parameter_binding [
        vamp:parameter [ vamp:identifier "LF" ] ;
        vamp:value "50"^^xsd:float ;
    ] ;
    vamp:parameter_binding [
        vamp:parameter [ vamp:identifier "nbins" ] ;
        vamp:value "12"^^xsd:float ;
    ] ;
    vamp:parameter_binding [
        vamp:parameter [ vamp:identifier "non_linear" ] ;
        vamp:value "0"^^xsd:float ;
    ] ;
    vamp:parameter_binding [
        vamp:parameter [ vamp:identifier "peakMagThreshold" ] ;
        vamp:value "-100"^^xsd:float ;
    ] ;
    vamp:parameter_binding [
        vamp:parameter [ vamp:identifier "reff0" ] ;
        vamp:value "440"^^xsd:float ;
    ] ;
    vamp:parameter_binding [
        vamp:parameter [ vamp:identifier "two_bands" ] ;
        vamp:value "1"^^xsd:float ;
    ] ;
    vamp:parameter binding [
```

Figura 5.4. Fichero de configuración del Vamp plug-in HPCP de Emilia Gómez.

Configuraremos el salto de ventana a 2048 muestras y el tamaño de ventana a 4096 en todos los cálculos. Fijadas estas variables, podremos observar la influencia de otros parámetros en los resultados.

5.2. Sonic Visualiser

Sonic Visualiser es una aplicación multiplataforma escrita en C++ que emplea un interfaz de usuario diseñado con Qt. Su valor reside en que proporciona una interfaz visual para las tareas de análisis de audio y extracción de información musical y permite gestionar una biblioteca de plug-ins de análisis desarrollados por multitud de investigadores en la materia.

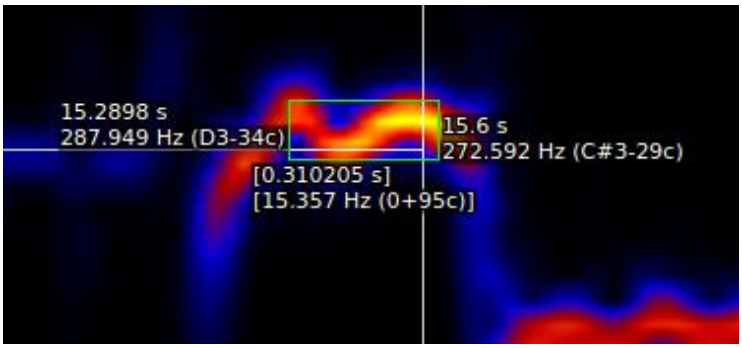


Figura 5.4. Espectrograma estudiado de manera visual en Sonic Visualiser. Detalle de la medida del *vibrato* en un espectro de voz (https://code.soundsoftware.ac.uk).

Los plug-ins están clasificados por Categoría, Nombre y Autor. Añadiendo múltiples ventanas pueden realizarse diferentes transformaciones sobre la misma señal de audio y comparar sus resultados, como se verá en la sección 5.3.3.

5.3. Algoritmos de cálculo de cromagrama empleados

A continuación, como parte de la fase exploratoria, se pretende realizar la evaluación comparada del comportamiento de dos plug-ins de cálculo de cromagrama ante las mismas señales de entrada. Se valorará principalmente la energía estimada por nota en cada instante (prominencia de las clases de alturas).

Mediante la suma de todos los histogramas del cromagrama (es decir, los vectores columna de croma) obtendremos también una estimación de la tonalidad de la señal, para poder comparar el comportamiento de cada plug-in.

Los algoritmos escogidos son

- NNLS *chroma* de Matthias Mauch.
- HPCP de Emilia Gómez.

En primer lugar, estudiaremos detalladamente las características de cada uno de los algoritmos escogidos para realizar una evaluación más completa de sus puntos fuertes y débiles.

5.3.1. Algoritmo NNLS

Uno de los algoritmos escogidos por su importancia para calcular los cromagramas es el NNLS *chroma* de Matthias Mauch y Simon Dixon (Mauch y Dixon, 2010a). La investigación en torno a la mejora del rendimiento de los algoritmos de transcripción de acordes muchas veces se ha centrado en encontrar buenos patrones para compararlos con los vectores de croma. Mauch y Dixon plantean con el algoritmo NNLS un enfoque opuesto, tratando de encontrar vectores de croma más aptos para emplearlos en modelos orientados al procesamiento de música. Realizan esta tarea por medio de una

transcripción aproximada inicial utilizando una técnica ya existente para resolver problemas de mínimos cuadrados no negativos (*Non-Negative Least Squares*, NNLS). Los vectores de croma NNLS resultantes son evaluados mediante su uso como datos de entrada en un modelo de alto nivel, de última generación, para la transcripción de acordes. De esta manera, se alcanzan resultados de hasta el 80% de precisión empleando las mismas anotaciones, colecciones de datos y métodos de evaluación que en el MIREX 2009, y superando el mejor resultado de dicha convocatoria (74%).

La naturaleza de determinados acordes dificulta su identificación, haciéndola susceptible de error al tratar de distinguir los parciales de la frecuencia fundamental. Encontramos esta situación, por ejemplo, en las inversiones del acorde triada mayor: los parciales de las voces tercera y quinta introducen notas extrañas a la armonía, no incluidas en el acorde, con influencia especialmente importante si se encuentran en la voz del bajo (primera y segunda inversión). La quinta y la tercera mayor de cada factor del acorde están muy presentes en el perfil armónico. Por ejemplo, la primera inversión de Do Mayor (nota MI en el bajo) lleva asociada la presencia notable de las notas SI y SOL♯ en el PCP. El método NNLS mejora de manera notoria la detección de estos acordes problemáticos a través de la mencionada transcripción inicial aproximada, antes de descartar la información de altura absoluta del espectro. Se considera que el espectro puede representarse aproximadamente como una combinación lineal de perfiles de nota de un diccionario o una matriz de perfiles E, ponderada por un vector de activación x, siendo $x \geq 0$:

$$Y \approx E \cdot x$$

El sistema de Mauch, para obtener los llamados cromagramas NNLS (*NNLS chroma*), calcula el espectrograma con eje de frecuencias logarítmico, continua con un pre-procesado del mismo y realiza la transcripción aproximada empleando el algoritmo NNLS. Esta transcripción es entonces transformada en un cromagrama y sincronizada con el pulso musical. Emplean por defecto, y recomiendan utilizar, una DFT de 4096 muestras, ventana de tipo Hamming y salto de ventana de 2048 muestras (solapamiento del 50%), lo cual es suficiente para detectar intervalos de segunda mayor en la región de baja frecuencia con intervalos temporales de aproximadamente 0.05 s, adecuándose así a la inmensa mayoría de aplicaciones musicales. Se utiliza un mapa de frecuencias con precisión de tercio de semitono en el que la frecuencia central de cada banda está linealmente espaciada en el eje de frecuencias logarítmico, es decir, se corresponde con las alturas musicales. Se aplica sobremuestreo sobre el espectro de la DFT como paso previo al ajuste logarítmico deseado para la representación del espectrograma. Asumiendo igual temperamento, se estima entonces la afinación global de la pieza a partir del espectrograma y, en lugar de ajustar los PCP en función de ésta, se actualiza el espectrograma logarítmico utilizando interpolación lineal, de manera que la banda central de cada semitono se corresponda con la frecuencia correcta según la afinación que se ha estimado (Mauch, 2010). El espectrograma logarítmico actualizado Y tiene 256 componentes de

$^1/_3$ de semitono (7 octavas aproximadamente), y es por tanto mucho más pequeño que el espectrograma original. No obstante, el tamaño reducido permite modelarlo eficientemente como la suma de notas idealizadas, como se explicará.

El pre-procesado que se aplica sobre el espectro logarítmico varía: el método *o* consiste en utilizar el espectrograma original, sin procesar; el método sub actúa eliminando el espectro de fondo (*background spectrum*, Catteau, et al., 2007) y el método std hace lo mismo que el anterior y lo completa dividiendo por la desviación típica móvil. Para estimar el espectro de fondo se utiliza la media móvil $\mu_{k,m}$, que es la media de una vecindad, en el ámbito de una octava, enventanada con una ventana Hamming (de la componente $k-18$ a la $k+18$). Los valores en los bordes del espectrograma, donde la ventana completa no está disponible, se asimilan a los de la componente más cercana. Entonces se le resta $\mu_{k,m}$ a $Y_{k,m}$, y los valores negativos son descartados (método sub). Además, si se divide por la respectiva desviación típica móvil $\sigma_{k,m}$, se obtiene una reducción móvil (método std). Esto es similar a lo que ocurre con el blanqueado espectral (*spectral whitening*, Klapuri, 2006) y sirve para descartar la información tímbrica. El espectro logarítmico resultante de la aplicación de ambos métodos de pre-procesado puede calcularse como

$$Y_{k,m}^{\rho} = \begin{cases} \dfrac{Y_{k,m} - \mu_{k,m}}{\sigma_{k,m}^{\rho}} & si\ Y_{k,m} - \mu_{k,m} > 0 \\ 0 & en\ otro\ caso, \end{cases}$$

donde $\rho = 0$ o $\rho = 1$ para los casos sub y std, respectivamente.

Con el fin de convertir un vector del espectrograma logarítmico en las diferentes notas que lo generan, se necesitan dos ingredientes básicos: un diccionario de notas E, que describe el perfil que se asume que tienen las notas (idealizadas), y un procedimiento de inferencia para determinar el patrón de activación de notas que mejor se ajuste a cada vector del espectrograma. Se genera un diccionario de perfiles de nota idealizados en el dominio frecuencial logarítmico empleando un modelo en el que las amplitudes de los sobretonos decrecen geométricamente (Gómez, 2006a),

$$a_k = s^{k-1}$$

donde el parámetro $s \in (0,1)$ tiene influencia sobre la forma del espectro: cuanto más pequeño sea el valor de s, más débiles serán los parciales de alta frecuencia. Gómez (2006a) favorece el parámetro $s = 0.6$ para la generación de su croma, mientras que en (Mauch, et al., 2009) se utilizó $s = 0.9$. Una tercera posibilidad es emplear dichos valores de s para la nota más alta y la más baja, respectivamente, y utilizar valores linealmente espaciados para todas las demás. Esto está motivado por el hecho de que las frecuencias de resonancia de los instrumentos musicales son fijas, y los

parciales de notas con frecuencia fundamental mayor es menos probable que correspondan a una resonancia, y por tanto decaerán antes puesto que no tienen un sostenimiento privilegiado debido a las características del instrumento. En cada uno de los tres casos, se crean patrones de nota a lo largo de siete octavas, con doce notas por octava: un conjunto de 84 perfiles de nota. Las frecuencias fundamentales de estos tonos van desde LA_0 (a 27.5 Hz) a $SOL\sharp_6$ (a aproximadamente 3322 Hz). Cada perfil de nota está normalizado de modo que la suma de todas las bandas resulta en la unidad. Juntas forman una matriz E, en la que cada columna corresponde a una nota.

A continuación se asume nuevamente que las componentes individuales del espectrograma de frecuencia logarítmica Y se generan de forma aproximada como una combinación lineal $Y_{\cdot,m} \approx E \cdot x$ de los 84 perfiles tonales. El problema reside en encontrar un patrón de activación de tono x que minimice la distanica euclidea

$$\|Y_{\cdot,m} - E \cdot x\|$$

entre la combinación lineal y los datos, teniendo en cuenta que $x \geq 0$, es decir, todas las activaciones deben ser positivas. Este es un conocido problema matemático denominado mínimos cuadrados no negativos (NNLS). Lawson y Hanson (1974) han propuesto un algoritmo para encontrar una solución, y dado que en este caso la matriz E tiene rango completo y más filas que columnas, la solución es también única. El método propuesto en Mauch y Dixon (2010) utiliza una implementación en MATLAB de este algoritmo. De nuevo, todas las muestras temporales se procesan por separado, y finalmente obtenemos una transcripción NNLS del espectro, S, en la que cada columna corresponde a un instante del audio de entrada y cada fila a un semitono. Por otro lado, es posible omitir la etapa de transcripción aproximada y copiar la banda central de cada semitono de Y en la banda correspondiente de S (Peeters, 2006).

El método NNLS utiliza un cromagrama de propósito general, un cromagrama exclusivo de la línea del bajo y aplica un procesado de sincronización con el pulso musical, para mejorar la detección y transición de alturas.

Emplearemos el mismo método que utiliza Mauch para estimar las activaciones de notas, es decir, el cálculo de la distancia euclídea entre patrones ideales y perfiles armónicos reales, para calcular en nuestro caso la activación de los acordes.

5.3.2. Algoritmo HPCP

El algoritmo HPCP es un Vamp plug-in que extrae la evolución instantánea del perfil armónico de croma (HPCP, *Harmonic Pitch Class Profile*) de una señal.

El HPCP es una propuesta para estimar el contenido cromático que representa las alturas en las señales musicales polifónicas, presentando esta información condensada en una única octava. Ha

sido utilizado ampliamente en numerosas aplicaciones como la estimación de acordes, inducción de tonalidad, identificación de versiones y clasificación de música.

Nos remitimos al Capítulo 4 de este trabajo para mayor detalle del algoritmo HPCP (Gómez, 2006a).

5.3.3. Representación comparada

Hemos empleado una serie de señales de prueba, sintetizadas con el lenguaje Csound, para obtener unas representaciones de cromagramas y valorar el comportamiento de HPCP y NNLS. Utilizaremos una progresión tonal de sinusoides con diferentes características tímbricas:

- ***Progresión tonal de tonos sintéticos*** (Do Mayor: C:maj, F:maj, G:maj, A:min, E:min, G:maj, C:maj[28]) con tres timbres diferentes. Cada acorde tiene una duración de 2 s, y los timbres son:

  [Audio: elaborado con Csound (Anexo B)]

  1. Tono puro.
  2. Tono con armónicos de amplitud decreciente
  3. Tono percusivo (algoritmo de Karplus-Strong)

Mostramos a continuación las representaciones de ambos cromagramas para las tres señales **instr-1, instr-2,** y **instr-3** (haciendo referencia a los diferentes instrumentos definidos en Csound, esto es, la especificación de sus timbres), para ver comparativa y gráficamente los resultados (Figura 5.5, 5.6 y 5.7). Más abajo lo haremos cuantitativamente. Hemos calculado los cromagramas en todos los casos con 4096 muestras de tamaño de ventana y 2048 de avance de ventana (solapamiento del 50%), los mismos parámetros que empleamos como parámetros del sistema de estimación de acordes.

Trabajando con sinusoides tenemos el caso más controlado posible para ver el comportamiento de los cromagramas. En el caso del instr-1 (Figura 5.5) se trata de senos puros; podemos observar que aparecen algunos defectos debido al propio análisis, pero generalmente sólo tienen energía las notas pertenecientes al acorde. En cambio, vemos cómo al introducir armónicos en la señal (instr-2, Figura 5.6) aparece "ruido" más evidente en otras clases de alturas diferentes de las propias de los acordes.

[28] En la sección 5.4.2 justificaremos esta sintaxis para los acordes.

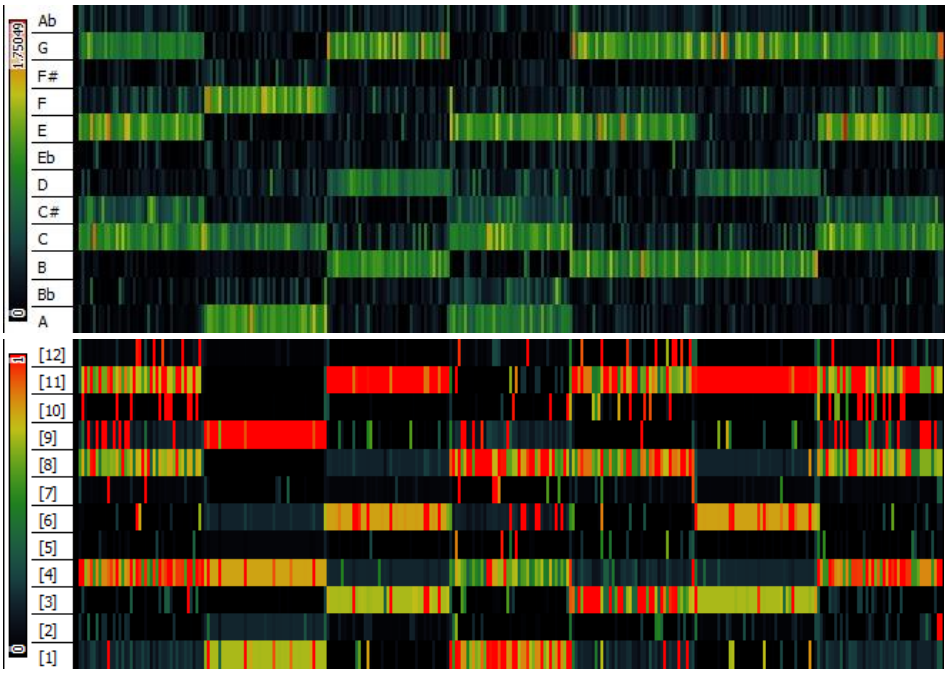


Figura 5.5. Comparativa de cromagramas calculados con el algoritmo **NNLS Chroma** de Matthias Mauch (arriba) y el **HPCP** de Emilia Gómez (abajo) para la señal de prueba[29] con el **instr-1** (tono puro).

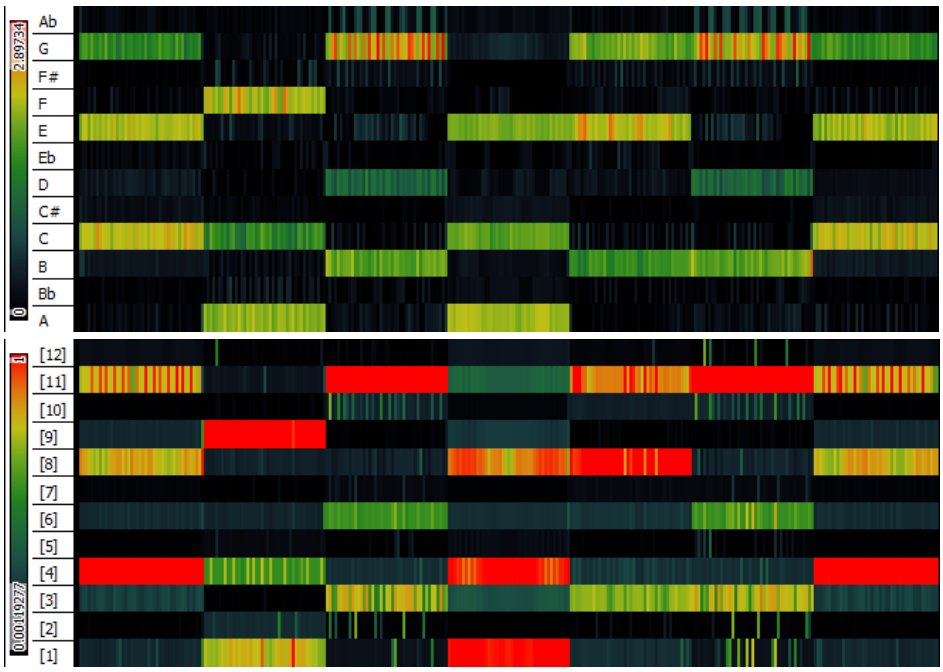


Figura 5.6. Comparativa de cromagramas calculados con el algoritmo **NNLS Chroma** de Matthias Mauch (arriba) y el **HPCP** de Emilia Gómez (abajo) para la señal de prueba con el **instr-2** (tono con armónicos de amplitud decreciente).

[29] Progresión: C:maj, F:maj, G:maj, A:min, E:min, G:maj, C:maj (en todos los casos).

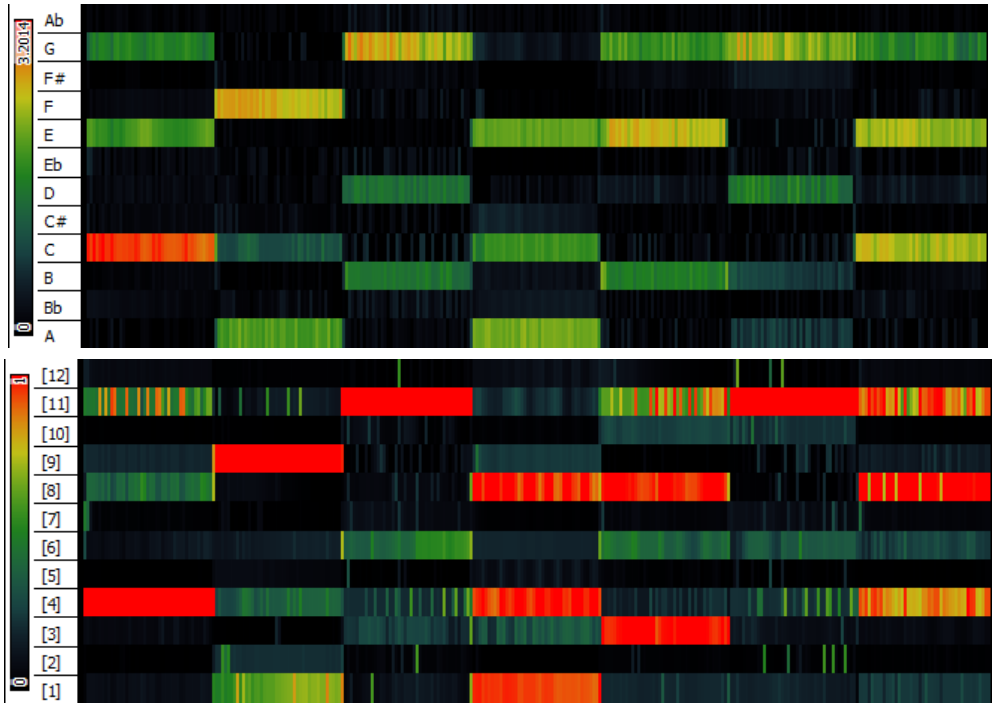


Figura 5.6. Comparativa de cromagramas calculados con el algoritmo **NNLS Chroma** de Matthias Mauch (arriba) y el **HPCP** de Emilia Gómez (abajo) para la señal de prueba con el **instr-3** (tono percusivo armónico).

En el caso percusivo (instr-3, Figura 5.6), vemos el HPCP especialmente saturado en las fundamentales de los acordes (C, F, G…) y tiene un comportamiento algo errático en otros instantes. Sí es cierto que el NNLS produce valores débiles para las mismas clases de alturas e instantes que el HPCP, pero en general apreciamos mayor estabilidad en el análisis de éste.

Si colapsamos los vectores de croma de todos los instantes de tiempo, estamos realizando una estimación de tonalidad y podemos ver de esta manera el comportamiento de cada plug-in. Para este experimento utilizaremos igualmente esta progresión de acordes con centro tonal en C, con los diferentes timbres de las señales instr-1, instr-2, instr-3.

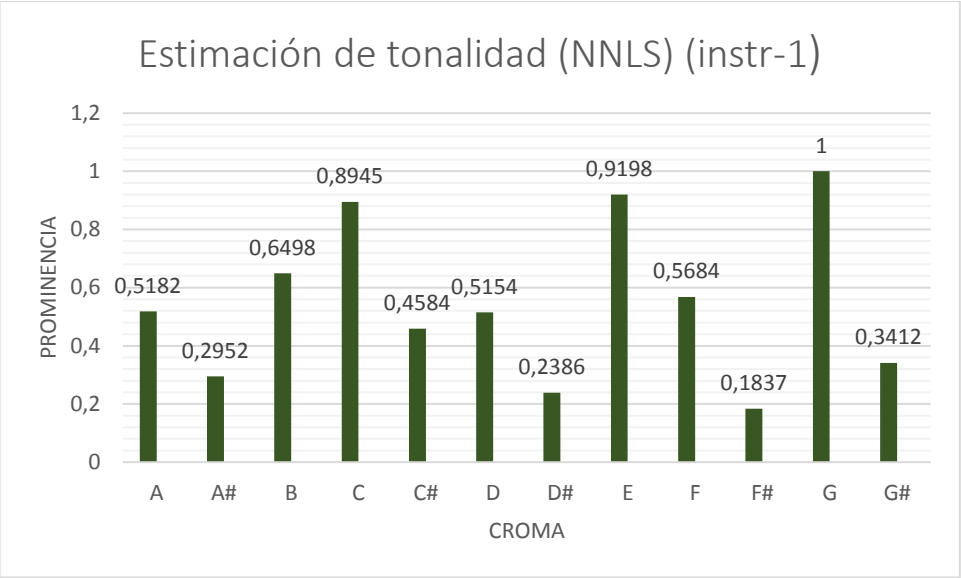
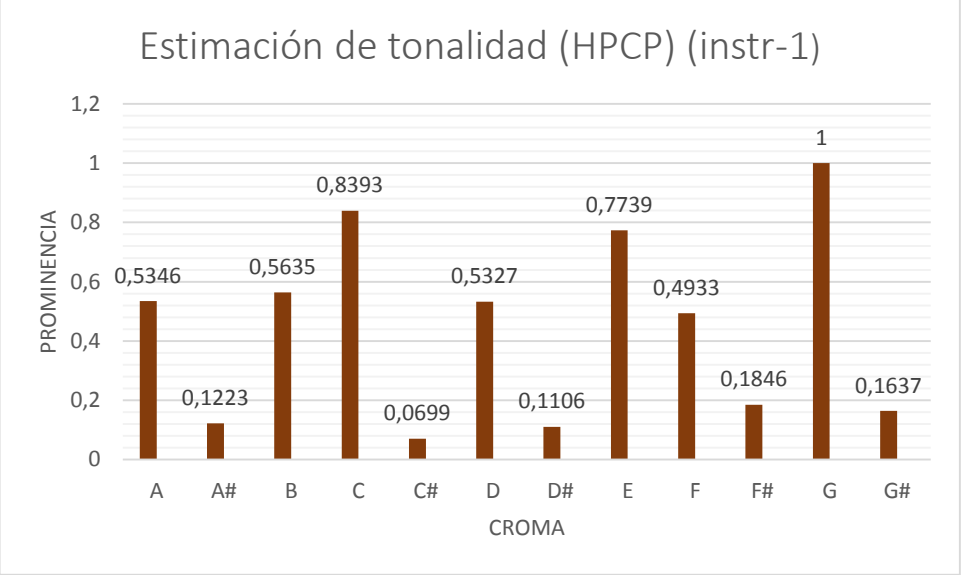


Figura 5.7. Estimación de tonalidad realizada sumando y normalizando todos los vectores de croma del audio de prueba con el **instr-1** (tono puro), obtenidos con el **NNLS Chroma** (arriba) y el **HPCP** (abajo).



Las notas que intervienen en la progresión aparecen lógicamente más reforzadas. En el caso de la Figura 5.7, donde tan solo intervienen sinusoides puras, esto es todavía más claro, puesto que no hay armónicos y tan solo toman partido las alturas de los factores de acorde. Quienes aparecen con mayor frecuencia, por así decirlo, son las notas propias de la tónica mayor, C:maj.

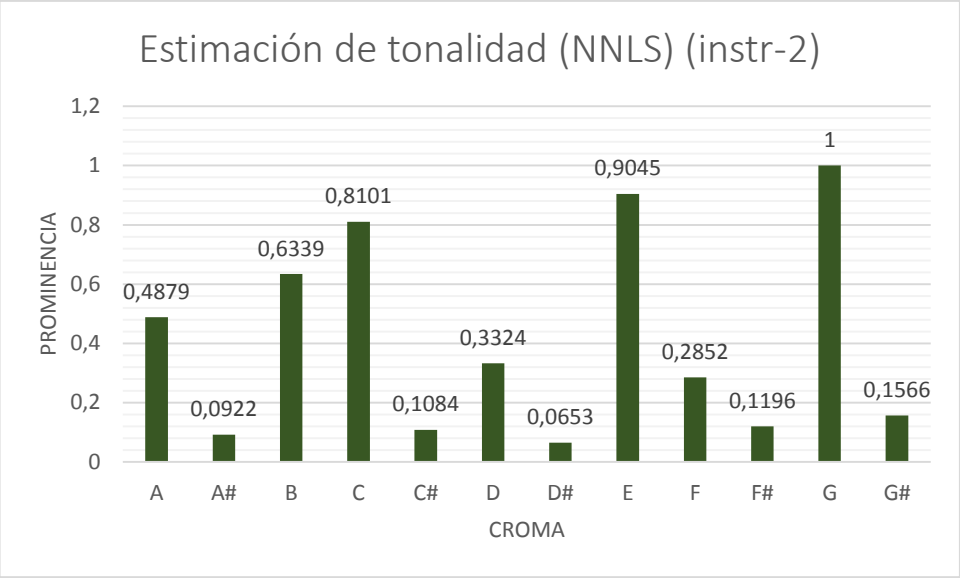
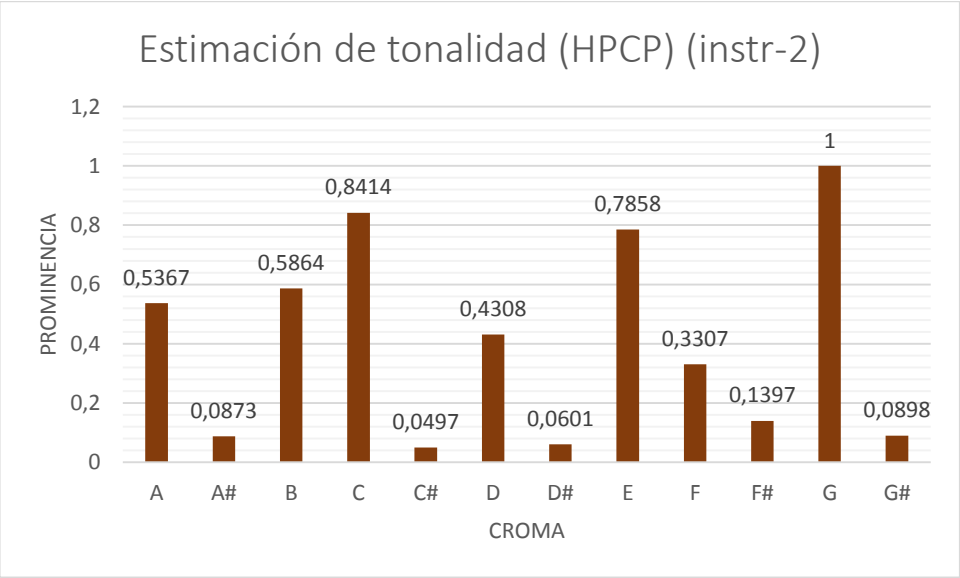


Figura 5.8. Estimación de tonalidad realizada sumando y normalizando todos los vectores de croma del audio de prueba con el **instr-2** (tono con armónicos de amplitud decreciente), obtenidos con el **NNLS Chroma** (arriba) y el **HPCP** (abajo).



Podemos observar cómo el relativo menor (A:min) cobra fuerza en la Figura 5.8, donde se han introducido los armónicos en la señal (intr-2), y pierden presencia clases de altura menos reforzadas por los sobretonos de los factores de los acordes (recordamos que hemos normalizado las representaciones, por lo que porcentualmente participan menos). Aparece también reforzado B, que recordamos que es la quinta de E, tercera del acorde de tónica en esta tonalidad; B es una nota muy importante en la serie armónica de E (3er y 6º armónicos), y además pertenece a la triada de G:maj, que es la dominante de la tonalidad (el "elemento tensor" que lleva al centro y por ello tiene mucha presencia en una progresión tonal). HPCP y NNLS detectan mejor unos u otros grados, pero en este caso no hay diferencias muy acusadas.

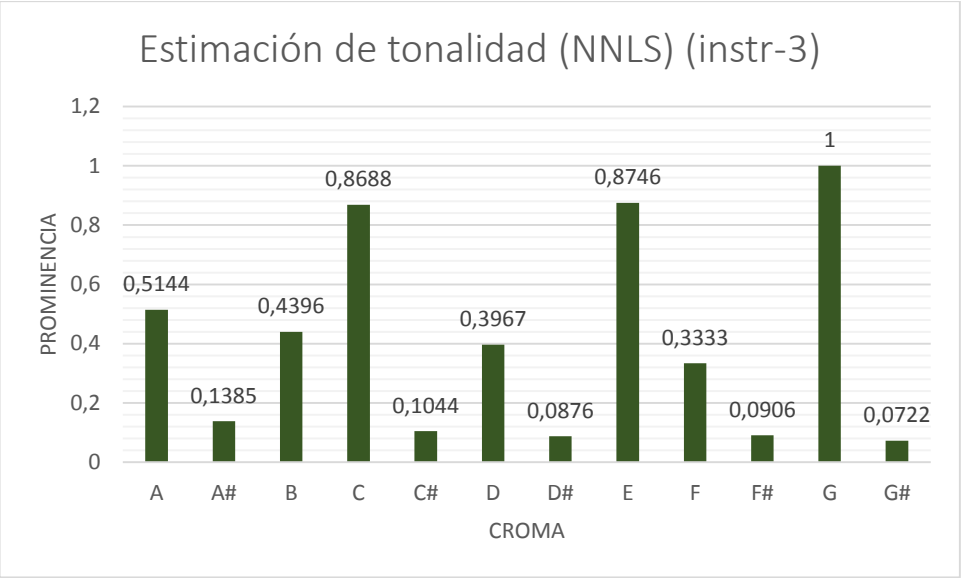
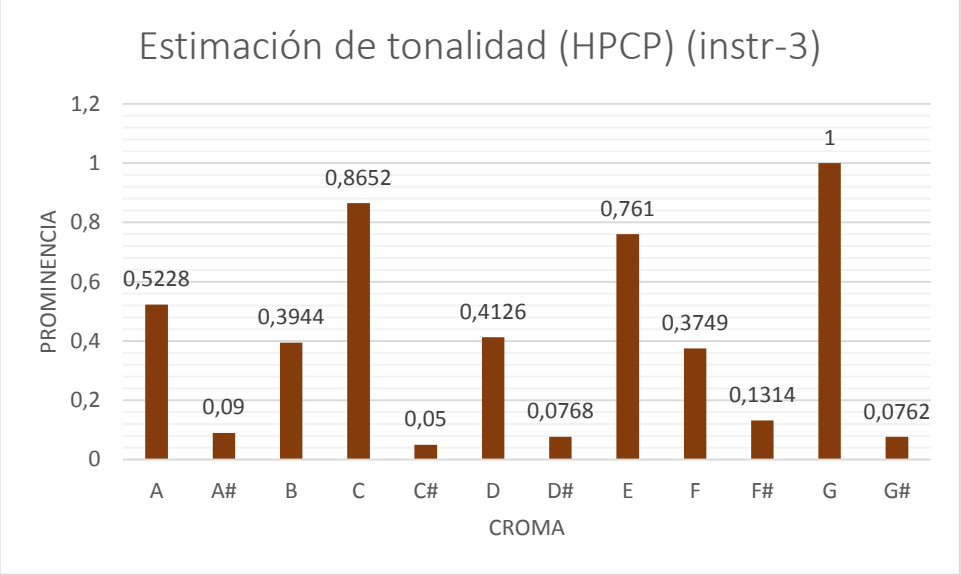


Figura 5.9. Estimación de tonalidad realizada sumando y normalizando todos los vectores de croma del audio de prueba con el **instr-3** (tono percusivo armónico), obtenidos con el **NNLS Chroma** (arriba) y el **HPCP** (abajo).



En este caso, para la señal percusiva generada por el instr-3 (Figura 5.9), vemos cómo se impone ligeramente el NNLS a la hora de detectar las clases de altura propias de la fundamental mayor. Tal vez no sea un dato demasiado importante, pero salta a la vista que el algoritmo NNLS fue presentado en 2010 (Mauch y Dixon, 2010) mientras que el HPCP vio la luz en el 2006 (Gómez, 2006a). Esto es, Matthias Mauch y Simon Dixon (QMUL) pudieron contar con el trabajo de Emilia Gómez (UPF) como referencia (de hecho, la citan) para depurar en cierta medida el análisis de información de croma. Aunque las diferencias no son tan evidentes con este breve análisis, utilizaremos el NNLS para la extracción de características durante el resto de este trabajo.

5.4. Vocabulario armónico

5.4.1. Análisis del problema. Elección de un conjunto de acordes

Considerando todas las notas de la escala cromática como posibles factores intervinientes, pueden darse un total de $2^{12} = 4096$ posibles acordes. Lo ideal sería considerar todos ellos por no dejar escapar ninguna posibilidad, pero resulta totalmente prohibitivo. Por tanto, lo razonable para evaluar un sistema de estimación de acordes es limitar el vocabulario armónico que se va a emplear.

Hemos de tener en cuenta que la práctica musical es basta y compleja, dada la variedad de estilos y tendencias existente a día de hoy en el repertorio musical. Como decimos, la amplitud del vocabulario armónico es fuertemente dependiente del estilo. El jazz parte de una concepción muy creativa de la música; siendo optimista, puede esperarse que las cuatriadas presentadas en la sección 3.3 sean los acordes más habituales, pero puede aparecer casi cualquier tipo de combinación de clases de alturas que el compositor o el intérprete decida introducir. En pop y rock el vocabulario suele ser más predecible y limitado, existe un conjunto de acordes más probables sobre el que se puede trabajar.

Si nos remitimos a las primeras propuestas en este campo, Fujishima consideraba 27 tipos de acordes, incluyendo ejemplos avanzados como A:(1, 3, 5♯, 7)/G (Fujishima, 1999). En 2003 se dieron algunos pasos hacia un alfabeto más manejable, con la aportación de Sheh y Ellis, quienes consideraron siete tipos de acordes (maj, min, maj7, min7, dom7[30], aug, dim) (Sheh y Ellis, 2003), aunque otros autores han explorado las posibilidades de utilizar únicamente las cuatro triadas principales: maj, min, aug y dim (Yoshioka, et al., 2004; Burgoyne, et al., 2007). Sumi y Mauch identificaron los acordes suspendidos, y en estudios recientes se ha incluido de manera adicional un símbolo para indicar "ausencia de acorde", silencio, fragmentos hablados o imposibilidad de detección (*no chord*, en definitiva). Mauch identificó un extenso alfabeto de hasta diez tipos de acordes, incluyendo inversiones (Mauch y Dixon, 2010b). A medida que avanza la sofisticación de los sistemas ACE sí que se exhorta a los investigadores a ampliar el vocabulario armónico más allá de las triadas maj/min para no dar hinchar desproporcionadamente las evaluaciones de sus algoritmos y nublar así el estado de la cuestión (McVicar, et al., 2014).

Como decimos, es común en la puesta a prueba de nuevos algoritmos y sistemas ACE limitar el contexto armónico a triadas mayores y menores en estado fundamental. En el prototipo inicial que hemos desarrollado se ha seguido este criterio. Este se debe a que el acorde de triada mayor, el más común indiscutiblemente en la música occidental, se considera difícil de detectar (Mauch y Dixon, 2010a; Barbancho, et al., 2013), porque la serie armónica de cada uno de los factores del acorde

[30] El término 'dom', ya aparezca solo o como 'dom7', suele hacer referencia a un acorde de séptima de dominante, diferenciándose de 'maj7' en el tipo de intervalo de séptima (menor / mayor).

(especialmente de la tercera y la quinta) tiene unas particularidades con las que el sistema se tiene que enfrentar, que ya hemos adelantado en la sección 5.3.1.

En la Figura 5.10 se muestran los espectros de la triada $DO_4SOL_5MI_6$ y sus factores, y podemos observar que existe una gran relación armónica entre estas tres notas. Por tanto, este acorde resulta difícil de transcribir, porque la relación entre las notas puede hacer que el sistema de transcripción considere un número inferior de ellas (solamente DO_4 y MI_6) o capture notas de diferentes octavas (como SOL_4 y MI_5, que son algunos de los parciales). Así pues, el acorde de triada mayor, con cualquier número de notas, clases de altura duplicadas en distintas octavas y variación del orden de las notas, se considera difícil de detectar.

Nótese que la separación entre las notas también influye en éxito de la detección. Por ejemplo, consideremos un acorde triada mayor como $DO_1MI_4SOL_7$; este acorde es más fácil de detectar que $DO_4MI_4SOL_4$, porque en el primero las notas están más separadas en frecuencia que en el segundo.

Otro factor es que el espectro de un instrumento cambia según el rango de frecuencias o tesitura en el que toque. En el caso del piano (Barbancho, et al., 2013), la armonicidad disminuye conforme nos acercamos a los extremos de la tesitura del instrumento. La detección de $DO_1MI_1SOL_1$ tiene que lidiar con el hecho de que la frecuencia fundamental de las diferentes voces es muy pequeña o, en ocasiones, no existe. Por otro lado, el acorde de piano $DO_7MI_7SOL_7$ tiene las frecuencias fundamentales de sus notas muy arriba en el espectro, tanto que en ocasiones caen en un rango no considerado en el análisis espectral.

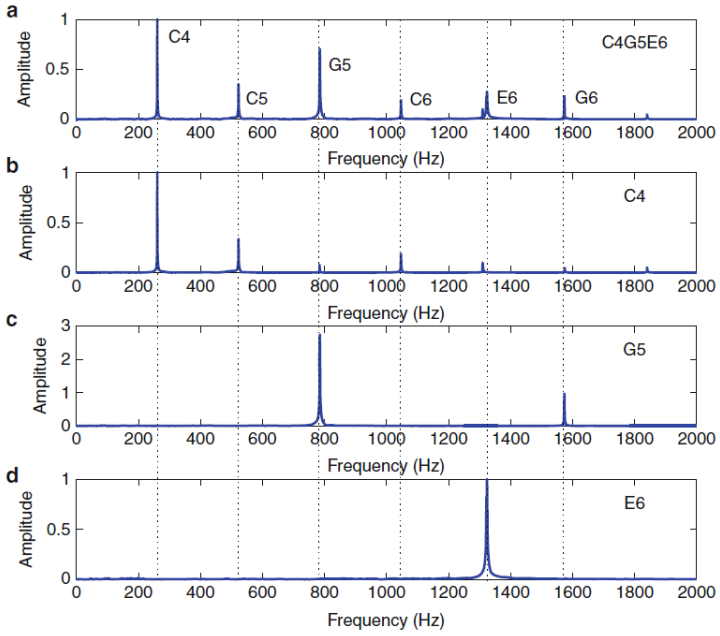


Figura 5.10. Comparación del espectro de (a) el acorde de triada mayor $DO_4MI_6SOL_5$ (C4G5E6) y sus notas individuales (b) DO_4, (c) SOL_5 y (d) MI_6 (Barbancho, et al., 2013).

Paradójicamente, los acordes disminuidos, de carácter disonante, son más fáciles de detectar para los sistemas de transcripción automática. A un músico le resulta más complicado identificarlos, porque son menos frecuentes en la música occidental, y sin embargo está muy acostumbrado a escuchar acordes mayores y menores y diferenciarlos. El espectro de un acorde disminuido no tiene apenas parciales coincidentes, por lo que el análisis arroja una colección de clases de alturas claramente diferenciada (Figura 5.11).

En primer lugar, por tanto, asumimos la gran complejidad que presenta la detección de acordes triada a partir del análisis de grabaciones de audio. En consecuencia, limitaremos el vocabulario armónico del prototipo del sistema a triadas mayores y menores en estado fundamental (nivel 2 de complejidad según Pauwels and Peters, 2013).

A la hora de poner a prueba el algoritmo definitivo, ampliaremos este vocabulario para abarcar otros tipos de triadas, como la disminuida, la aumentada y el acorde de cuarta suspendida, para ver cómo se comporta nuestro sistema en estos casos y sacar algunas conclusiones.

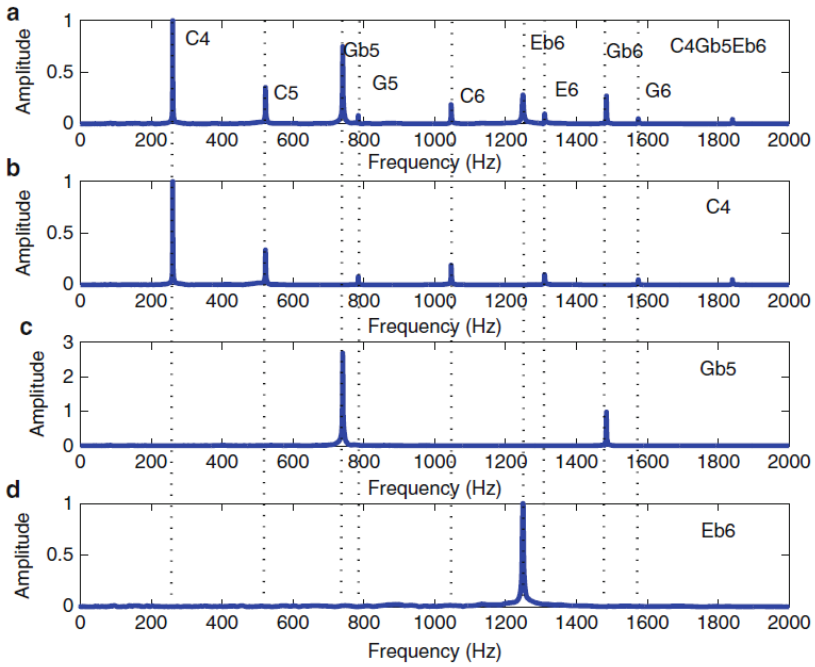


Figura 5.11. Comparación del espectro de (a) el acorde disminuido $DO_4Mi♭_6SOL♭_5$ y sus notas individuales (b) DO_4, (c) $SOL♭_5$ y (d) $MI♭_6$ (Barbancho, et al., 2013).

5.4.2. Sintaxis

En cuanto a la nomenclatura de dichos acordes, nos sumamos a la sugerencia de Harte (Harte, et al., 2005) de que las anotaciones de acordes se describan con una sintaxis estándar para toda la comunidad investigadora en MIR, a la vez intuitiva para individuos con preparación musical y adecuada para su tratamiento computacional (Tabla 5.1).

```
<chord>        ::=   <note> ":" <shorthand> ["("<degree-list>")"]["/"<degree>]
                   | <note> ":" "("<degree-list>")" ["/"<degree>]
                   | <note> ["/"<degree>]  | "N"

<note>         ::=  <natural>|<note> <modifier>

<natural>      ::=  A|B|C|D|E|F|G

<modifier>     ::=  b|#

<degree-list>  ::=  ["*"] <degree> ["," <degree-list>]

<degree>       ::=  <interval>|<modifier> <degree>

<interval>     ::=  1|2|3|4|5|6|7|8|9|10|11|12|13

<shorthand>    ::=  maj|min|dim|aug|maj7|min7|7|dim7|hdim7
                   |minmaj7|maj6|min6|9|maj9|min9|sus4
```

Tabla 5.1. Sintaxis de notación musical según la forma Backus-Naur (BNF) (Harte, et al., 2005).

Chord Type		Shorthand Notation	Components List
Triad Chords:			
	Major	maj	(3,5)
	Minor	min	(b3,5)
	Diminished	dim	(b3,b5)
	Augmented	aug	(3,#5)
Seventh Chords:			
	Major Seventh	maj7	(3,5,7)
	Minor Seventh	min7	(b3,5,b7)
	Seventh	7	(3,5,b7)
	Diminished Seventh	dim7	(b3,b5,bb7)
	Half Diminished Seventh	hdim7	(b3,b5,b7)
	Minor (Major Seventh)	minmaj7	(b3,5,7)
Sixth Chords:			
	Major Sixth	maj6	(3,5,6)
	Minor Sixth	min6	(b3,5,6)
Extended Chords:			
	Ninth	9	(3,5,b7,9)
	Major Ninth	maj9	(3,5,7,9)
	Minor Ninth	min9	(b3,5,b7,9)
Suspended Chords:			
	Suspended 4th	sus4	(4,5)

Tabla 5.2. Definición de acordes comunes y su cualidad (Harte, et al., 2005).

Todo acorde constará de una raíz o nota fundamental y una cualidad, separados por el carácter ':'. De momento el campo de cualidad o tipo de acorde se limitará a las etiquetas mayor y menor. Queda, pues, el formato `nota_fundamental:tipo_de_acorde`. Por ejemplo, los acordes de Do Mayor y Do menor se describirán como C:maj y C:min, respectivamente. En la Figura 5.12 se muestra la colección completa de triadas que se usará y su nomenclatura.

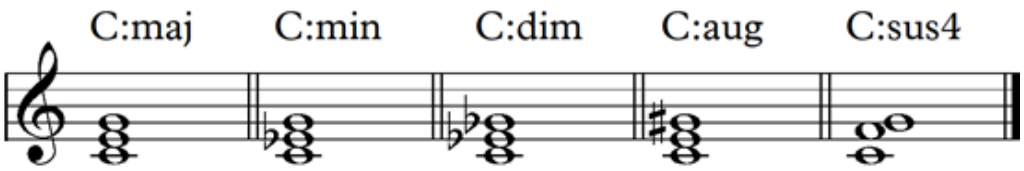


Figura 5.12. Vocabulario armónico del presente proyecto y su sintaxis.

Usaremos, por tanto, acordes mayores y menores, acordes aumentados y disminuidos y acordes de cuarta suspendida, para aumentar el espectro de triadas básicas.

5.5. Colección de datos de puesta a prueba

Ya que este proyecto se orienta en gran medida a la transcripción polifónica monotímbrica, hemos hecho especial énfasis en este punto a la hora de configurar nuestra colección de datos. El *data set* se compondrá de 60 acordes de piano, escogidos de forma que cubramos un rango de casos interesantes para poner a prueba el algoritmo, y añadiremos a éstos 10 canciones de The Beatles. Si bien nuestro sistema no tiene por qué entender plenamente estos temas pop-rock, puesto que nuestro objetivo original es otro distinto y esas señales están muy lejos de ser audio monotímbrico, queremos comprobar hasta dónde llega aprovechando el módulo de pre-procesado que hemos decidido emplear.

5.5.1. Fuentes

Principalmente emplearemos dos fuentes de información para obtener los audios o las anotaciones manuales de acordes con las que verificar la estimación de nuestro sistema.

Base de datos de acordes de piano de la Universidad de Málaga (UMA)

En cuanto a la colección de audios de entrada empleada para evaluar el sistema, hemos escogido, en primer lugar, la base de datos de acordes de piano de la Universidad de Málaga, que es la mayor en su especie actualmente y de acceso abierto a toda la comunidad investigadora.

Presenta acordes de piano con hasta 7 niveles de polifonía, es decir, 7 factores de acorde, que bien pueden ser clases de alturas diferentes o clases duplicadas en otra octava. Los acordes están interpretados en tres niveles dinámicos diferentes (*forte, mezzo* y *piano*) y en tres estilos de interpretación (normal¸ *staccato* y pedal).

Como documentación sobre esta base de datos, hacemos referencia, nuevamente, al libro que han publicado estos investigadores sobre la base de datos que han creado [31] (Barbancho, et al., 2013).

Isophonics

La colección de datos de Beatles, Queen, Michael Jackson y otros, del Centre for Digital Music de la Universidad Queen Mary de Londres[32], ha sido empleada durante muchos años en trabajos de ACE presentados al MIREX.

Remitimos al lector a la tesis de Matthias Mauch (Mauch, 2010) y al artículo introductorio de Christopher Harte (Harte, et. al, 2005) como documentación de esta base de datos. La web de Isophonics, además, hace referencia a la tesis de Harte (Harte, 2010) como introducción a las anotaciones de The Beatles que hemos empleado en este proyecto.

5.5.2. Colección de archivos de audio

A continuación presentamos esquemáticamente los archivos que hemos decidido emplear para evaluar el sistema una vez implementado (Tabla 5.3 y 5.4). Los archivos se encuentran almacenados en el directorio 'Data_wav', incluido en la carpeta del programa desarrollado en MATLAB.

- ***60 acordes de piano*** de diferentes tipos, dinámicas y tesitura.

[Audio: Database of piano chords (Barbancho, et al., 2013)]
[Anotaciones: elaboración propia]

Nº	Nombre de archivo	Descripción
1	UMAPiano-DB-C3E3G3-NO-F.wav	maj; rango medio; *forte*
2	UMAPiano-DB-C3E3G3-NO-M.wav	maj; rango medio; *mezzo*
3	UMAPiano-DB-C3E3G3-NO-P.wav	maj; rango medio; *piano*
4	UMAPiano-DB-C1C3E3-NO-F.wav	maj; rango grave; *forte*
5	UMAPiano-DB-C1C3E3-NO-M.wav	maj; rango grave; *mezzo*
6	UMAPiano-DB-C1C3E3-NO-P.wav	maj; rango grave; *piano*
7	UMAPiano-DB-G#6C7Eb7-NO-F.wav	maj; rango agudo; *forte*
8	UMAPiano-DB-G#6C7Eb7-NO-M.wav	maj; rango agudo; *mezzo*

[31] Introduciendo en http://extras.springer.com el ISBN del libro (978-1-4614-7475-3) puede descargarse la base de datos. El extracto que hemos empleado lo facilitamos en la documentación adicional.
[32] http://www.isophonics.net/

9	UMAPiano-DB-G#6C7Eb7-NO-P.wav	maj; rango agudo; *piano*
10	UMAPiano-DB-C3Eb3G3-NO-F.wav	min; rango medio; *forte*
11	UMAPiano-DB-C3Eb3G3-NO-M.wav	min; rango medio; *mezzo*
12	UMAPiano-DB-C3Eb3G3-NO-P.wav	min; rango medio; *piano*
13	UMAPiano-DB-C1C3Eb3-NO-F.wav	min; rango grave; *forte*
14	UMAPiano-DB-C1C3Eb3-NO-M.wav	min; rango grave; *mezzo*
15	UMAPiano-DB-C1C3Eb3-NO-P.wav	min; rango grave; *piano*
16	UMAPiano-DB-G6Eb7C8-NO-F.wav	min; rango agudo; *forte*
17	UMAPiano-DB-G6Eb7C8-NO-M.wav	min; rango agudo; *mezzo*
18	UMAPiano-DB-G6Eb7C8-NO-P.wav	min; rango agudo; *piano*
19	UMAPiano-DB-Eb4G#4C6-NO-F.wav	maj/3; rango medio-alto; *forte*
20	UMAPiano-DB-Eb4G#4C6-NO-M.wav	maj/3; rango medio-alto; *mezzo*
21	UMAPiano-DB-Eb4G#4C6-NO-P.wav	maj/3; rango medio-alto; *piano*
22	UMAPiano-DB-C3A3E4-NO-F.wav	min/3; rango medio; *forte*
23	UMAPiano-DB-C3A3E4-NO-M.wav	min/3; rango medio; *mezzo*
24	UMAPiano-DB-C3A3E4-NO-P.wav	min/3; rango medio; *piano*
25	UMAPiano-DB-E3A3C#4-NO-F.wav	maj/5; rango medio; *forte*
26	UMAPiano-DB-E3A3C#4-NO-M.wav	maj/5; rango medio; *mezzo*
27	UMAPiano-DB-E3A3C#4-NO-P.wav	maj/5; rango medio; *piano*
28	UMAPiano-DB-E3A3C4-NO-F.wav	min/5; rango medio; *forte*
29	UMAPiano-DB-E3A3C4-NO-M.wav	min/5; rango medio; *mezzo*
30	UMAPiano-DB-E3A3C4-NO-P.wav	min/5; rango medio; *piano*
31	UMAPiano-DB-C3Eb3Gb3-NO-F.wav	dim; rango medio; *forte*
32	UMAPiano-DB-C3Eb3Gb3-NO-M.wav	dim; rango medio; *mezzo*
33	UMAPiano-DB-C3Eb3Gb3-NO-P.wav	dim; rango medio; *piano*
34	UMAPiano-DB-C1Eb2Gb2-NO-F.wav	dim; rango grave; *forte*
35	UMAPiano-DB-C1Eb2Gb2-NO-M.wav	dim; rango grave; *mezzo*
36	UMAPiano-DB-C1Eb2Gb2-NO-P.wav	dim; rango grave; *piano*
37	UMAPiano-DB-C5Gb5Eb6-NO-F.wav	dim; rango agudo; *forte*
38	UMAPiano-DB-C5Gb5Eb6-NO-M.wav	dim; rango agudo; *mezzo*
39	UMAPiano-DB-C5Gb5Eb6-NO-P.wav	dim; rango agudo; *piano*

40	UMAPiano-DB-C3E4G#4-NO-F.wav	aug; rango medio; *forte*
41	UMAPiano-DB-C3E4G#4-NO-M.wav	aug; rango medio; *mezzo*
42	UMAPiano-DB-C3E4G#4-NO-P.wav	aug; rango medio; *piano*
43	UMAPiano-DB-C2E2G#2-NO-F.wav	aug; rango grave; *forte*
44	UMAPiano-DB-C2E2G#2-NO-M.wav	aug; rango grave; *mezzo*
45	UMAPiano-DB-C2E2G#2-NO-P.wav	aug; rango grave; *piano*
46	UMAPiano-DB-C5G#6Eb7-NO-F.wav	aug; rango agudo; *forte*
47	UMAPiano-DB-C5G#6Eb7-NO-M.wav	aug; rango agudo; *mezzo*
48	UMAPiano-DB-C5G#6Eb7-NO-P.wav	aug; rango agudo; *piano*
49	UMAPiano-DB-C4F4G4-NO-F.wav (*)[33]	sus4; rango medio; *forte*
50	UMAPiano-DB-C4F4G4-NO-M.wav (*)	sus4; rango medio; *mezzo*
51	UMAPiano-DB-C4F4G4-NO-P.wav (*)	sus4; rango medio; *piano*
52	UMAPiano-DB-C2F2G2-NO-F.wav (*)	sus4; rango grave; *forte*
53	UMAPiano-DB-C2F2G2-NO-M.wav (*)	sus4; rango grave; *mezzo*
54	UMAPiano-DB-C2F2G2-NO-P.wav (*)	sus4; rango grave; *piano*
55	UMAPiano-DB-C6F6G6-NO-F.wav (*)	sus4; rango agudo; *forte*
56	UMAPiano-DB-C6F6G6-NO-M.wav (*)	sus4; rango agudo; *mezzo*
57	UMAPiano-DB-C6F6G6-NO-P.wav (*)	sus4; rango agudo; *piano*
58	UMAPiano-DB-C1Bb1G2E3-NO-F.wav	dom7; rango grave; *forte*
59	UMAPiano-DB-C2Bb3E4G4-NO-F.wav	dom7; rango medio; *forte*
60	UMAPiano-DB-C3Bb4E5G5-NO-F.wav	dom7; rango agudo; *forte*

Tabla 5.3. *Data set* extraído de la base de datos de acordes de piano de la Universidad de Málaga (UMA).

[33] Los ficheros marcados con asterisco, correspondientes a las triadas de cuarta suspendida (sus4) no están contenidos en la base de datos de la UMA como tal; los hemos generado mezclando el intervalo CxFx con la nota Gx, contenidos en los directorios de número de polifonía 2 y 1, respectivamente.

- ***10 temas de The Beatles*.**

[Audio: álbumes en CD]
[Anotaciones: Isophonics]

Nº	Nombre de archivo
1	The_Beatles_A_Taste_of_Honey.wav
2	The_Beatles_Across_the_Universe.wav
3	The_Beatles_Act_Naturally.wav
4	The_Beatles_All_My_Loving.wav
5	The_Beatles_All_You_Need_Is_Love.wav
6	The_Beatles_And_I_Love_Her.wav
7	The_Beatles_Here_Comes_The_Sun.wav
8	The_Beatles_I_Me_Mine.wav
9	The_Beatles_I_Saw_Her_Standing_There.wav
10	The_Beatles_In_My_Life.wav

Tabla 5.4. *Data set* extraído de los álbumes de The Beatles.

5.6. Versión inicial del sistema de detección de acordes

5.6.1. Diagrama de bloques del sistema

Se ha implementado un sistema básico de reconocimiento de acordes, cuyo esquema de bloques se presenta a continuación (Figura 5.13). Consta de una etapa de extracción de características de croma, valiéndose del software Sonic Annotator y el plug-in HPCP, y otra de modelado mediante comparación de patrones, desarrollada en lenguaje MATLAB. Como puede observarse, se ha prescindido de etapas de pre-procesado y post-procesado en este sistema inicial. A continuación se describe en detalle el sistema realizado.

El algoritmo HPCP ha de ejecutarse a través de Sonic Annotator, en la aplicación de consola de comandos tradicional. Debe generarse en primer lugar el fichero de configuración del plug-in, donde principalmente se deben ajustar los parámetros $block\_size$ y $step\_size$. La resolución temporal dependerá de estos parámetros asociados a la transformada Q del algoritmo. Una vez ejecutada la transformación sobre el fichero de audio objetivo, se obtendrá un archivo .csv (*comma separated vector*) que contendrá el cromagrama calculado. Será una matriz de M filas y 13 columnas, siendo

M el número de muestras temporales; la primera columna corresponderá a los tiempos de inicio de muestra y las doce restantes a la prominencia de cada uno de los semitonos en ese instante de tiempo.

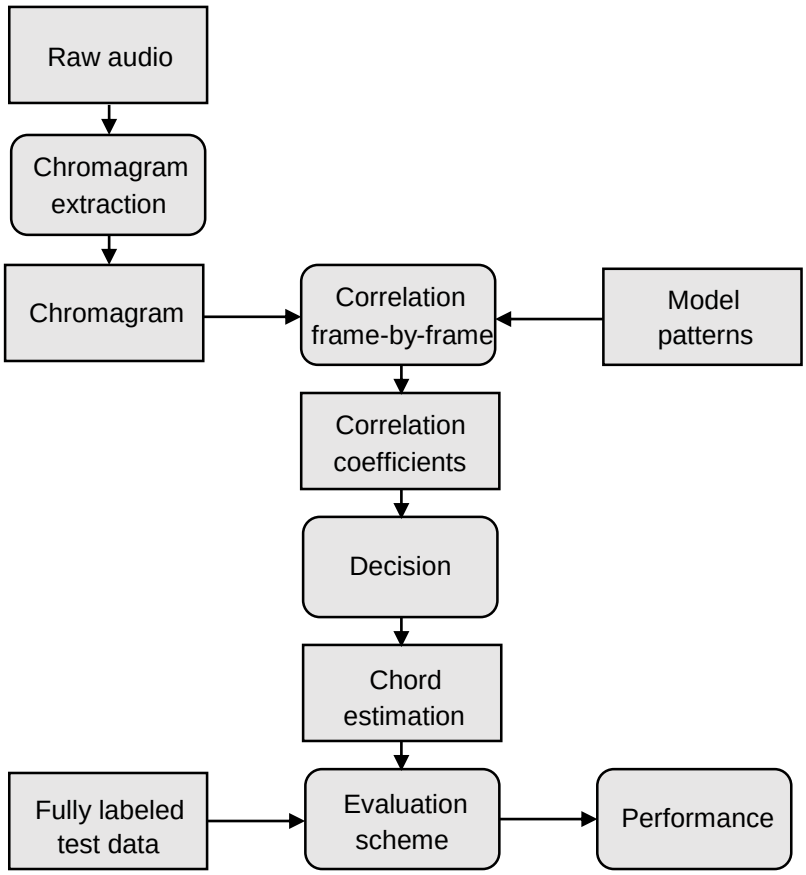


Figura 5.13. Diagrama de flujo del sistema. Los datos se muestran como rectángulos y los procesos como rectángulos redondeados. Los parámetros del modelo para el caso inicial son los patrones de vector de croma correspondiente a los acordes tríada mayor y menor ideales.

5.6.2. Patrones de acordes ideales

Como se ha comentado anteriormente, el enfoque adoptado para la etapa de modelado ha sido el más sencillo a priori de entre las técnicas desarrolladas en la actualidad, que es la comparación con patrones predefinidos (*template matching*) (Izmirli, 2005; McVicar, et al., 2014).

Estos patrones serán ideales, es decir, la distribución de las notas en el acorde se modelará con una secuencia binaria que indicará la presencia o ausencia de cada factor. Los acordes analizados son tríadas mayores y menores, cuyo vector teórico tendrá niveles altos únicamente en fundamental, tercera y quinta (Figura 5.14).

El acorde triada mayor se representará con el vector

$$\boldsymbol{p_{maj}} = [1\ 0\ 0\ 0\ 1\ 0\ 0\ 1\ 0\ 0\ 0\ 0].$$

Realizaremos operaciones sencillas de rotación en el vector para generar la colección completa de triadas mayores:

$$\boldsymbol{p}_{\{LA\}} = [1\,0\,0\,0\,1\,0\,0\,1\,0\,0\,0\,0]$$

$$\boldsymbol{p}_{\{LA\#\}} = [0\,1\,0\,0\,0\,1\,0\,0\,1\,0\,0\,0]$$

$$\boldsymbol{p}_{\{SI\}} = [0\,0\,1\,0\,0\,0\,1\,0\,0\,1\,0\,0]$$

$$etc.$$

A medida que se realizan las rotaciones, almacenaremos los vectores generados en una matriz cuadrada $\boldsymbol{P_{maj}}$ (12 tónicas × 12 elementos de croma):

$$\boldsymbol{P_{maj}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Podemos ver aquí los perfiles armónicos ideales de las 12 triadas mayores organizados por filas (primera fila, LA mayor; segunda fila, LA# mayor; …; última fila, SOL# mayor).

Hemos organizado las clases de alturas de forma que el vector generador $\boldsymbol{p_{maj}}$, sin rotar, corresponde al acorde de LA mayor, siguiendo la propuesta de Gómez, 2006a. Es decir, las clases están indexadas en `A`:

$$\boldsymbol{P_{maj}}(i, 1) \Rightarrow \mathrm{A}\ ,$$

y cada columna corresponde a una clase de altura de la lista `{A, A♯, B, C, C♯, D, D♯, E, F, F♯, G, G♯}`.

Podría no ser necesario gastar memoria en manejar una matriz y generar los vectores desplazando in situ el vector básico $\boldsymbol{p_{maj}}$ para conseguir todos los acordes mayores cuando se los necesite. En este proyecto no tenemos un requerimiento de optimización de memoria. No obstante, adoptamos esta primera aproximación por resultar conceptual y visualmente sencilla de entender. En la versión final del proyecto planteamos adoptar una versión alternativa.

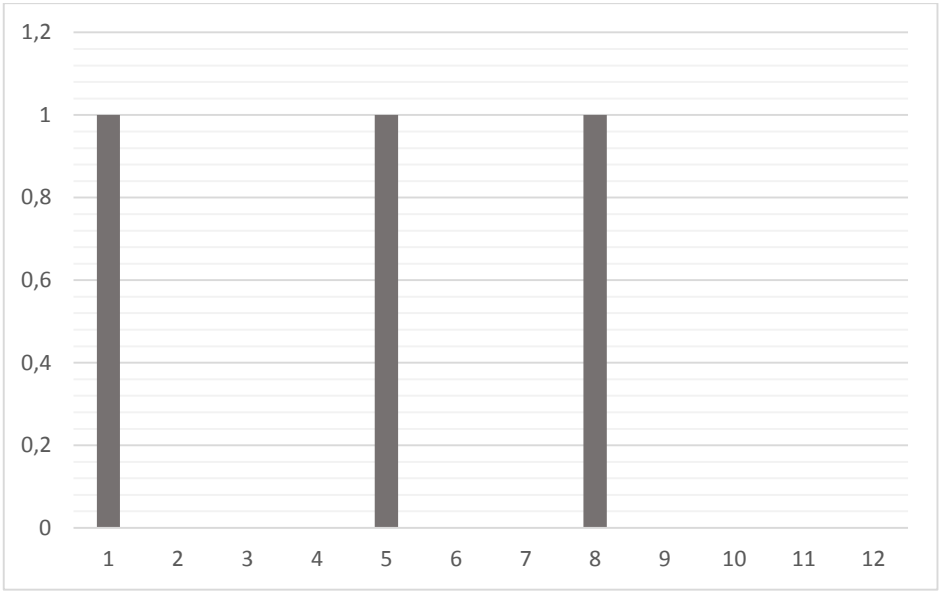


Figura 5.14. Patrón ideal del acorde de triada mayor; amplitud normalizada frente a clases de alturas. Empleamos esta vez un eje numérico para remarcar el carácter general de este patrón.

Procedemos de manera totalmente análoga para el caso de los patrones de triada menor (Figura 5.15). El patrón generador en este caso es

$$\boldsymbol{p_{min}} = [1\ 0\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0]\ ,$$

y mediante rotaciones obtenemos la matriz con los 12 patrones triada menor

$$\boldsymbol{P_{min}} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

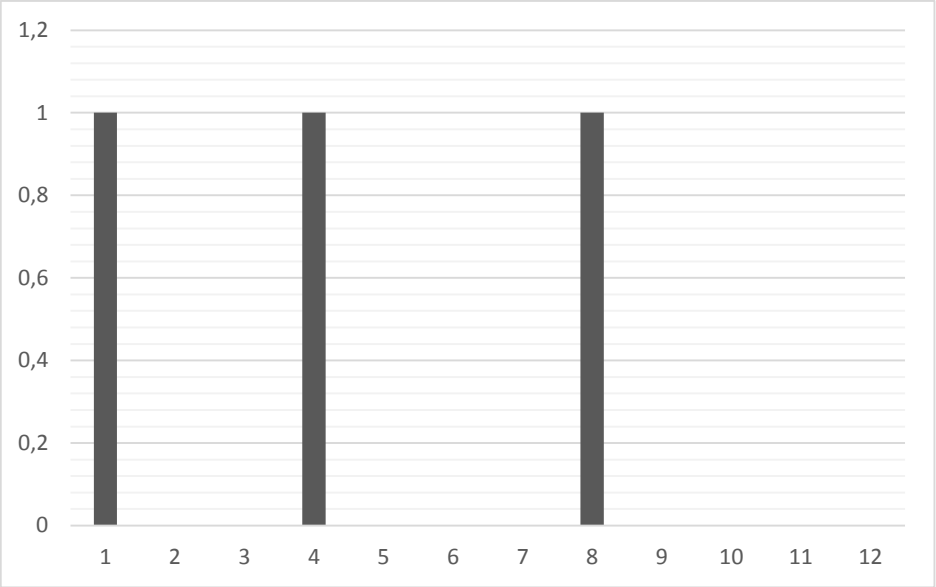


Figura 5.15. Patrón ideal del acorde de triada menor; amplitud normalizada frente a clases de alturas. Empleamos esta vez un eje numérico para remarcar el carácter general de este patrón.

5.6.3. Similitud: distancia euclídea

Como método de similitud, se ha escogido por sencillez la distancia euclídea o norma L2 entre dos vectores **p** y **h**, definida como:

$$d_E(\boldsymbol{p}, \boldsymbol{h}) = \sqrt{\sum_{i=1}^{N} (p_i - h_i)^2}$$

Por cada instante temporal y cada vector de croma $\boldsymbol{x}$, procedente del cromagrama $\boldsymbol{X}$, se obtendrán 24 distancias euclídeas, 12 correspondientes a los acordes mayores y los otros 12 a los menores.

En el momento en que se amplíe el vocabulario armónico, se ampliará lógicamente el número de distancias a calcular. Se calculará una distancia por cada acorde patrón c_i, esto es, una por cada posible clase de altura que actúa como voz fundamental.

Se separará la funcionalidad de similitud en dos etapas: el cálculo de las distancias euclídeas y la comparación de éstas entre sí, que tendrá lugar en la fase de decisión.

5.6.4. Decisión

5.6.4.1. Lógica de menor distancia

En definitiva, lo que se quiere es saber qué acorde c_i es el que tiene mayor probabilidad de estar sonando de cada instante de tiempo. De manera sencilla, través de una serie de instrucciones lógicas,

determinaremos para cada *frame* qué patrón de acorde $\boldsymbol{p}$ del vocabulario c_j guarda una distancia menor con el vector de croma i-ésimo $\boldsymbol{x}$. El acorde estimado será:

$$\widehat{c_i} = argmin_i\big(d_E(\boldsymbol{p}, \boldsymbol{x})\big)\ ,$$

es decir, aquel que minimice la distancia entre estos vectores.

5.6.4.2. Umbral de detección de acorde

En este proyecto, nuestro interés se centra en ser capaces de realizar la segmentación armónica de un archivo de audio, las zonas en las que predomina cierta armonía, sin importar demasiado los tiempos exactos de inicio y fin de cada uno de los acordes.

No obstante, es necesario al menos ser capaces de diferenciar entre una situación monofónica y una polifónica de manera automática, para que el algoritmo no pretenda detectar acordes cuando no los hay. El resultado de calcular la distancia euclídea entre los perfiles armónicos ideal y real para el caso de una señal monofónica deberá ser radicalmente diferente, a priori, que en el caso polifónico.

Para corroborar esta hipótesis, realizamos un breve experimento calculando el cromagrama NNLS de un fragmento de *Promenade II* de la suite *Cuadros de una exposición* (Mussorgsky), en su versión original a piano[34]. Resulta interesante en este contexto porque presenta una línea monofónica que se alterna a intervalos regulares con la respuesta polifónica. Además, contiene acordes triada únicamente. Eso sí, en estado fundamental y también con inversiones. Nuestro interés está, en cualquier caso, en la diferencia entre los compases monofónicos y polifónicos (Figura 5.16).

El vector de croma i-ésimo $\boldsymbol{x}_i$, que en este caso denotamos con $\boldsymbol{h}$, se compara con el patrón de acorde triada k-ésimo $\boldsymbol{p}$; al ejecutar el sistema realizado en MATLAB resulta, para el caso monofónico,

$$\max(\boldsymbol{D_E}) = 1.7321\ ;$$

y en el caso polifónico

$$\min(\boldsymbol{D_E}) = 0.5695\ ,$$

siendo $\boldsymbol{D_E}$ la matriz que engloba todas las distancias, para todos los *frames* y tipos de acorde.

La alternancia regular entre línea monofónica y polifónica se ve reflejada en los valores máximo y mínimo de la distancia euclídea. Cuando suenan acordes, los patrones correlacionan mejor con los vectores de croma.

[34] Audio: Petrucci Music Library (IMSLP) - Intérprete:Alexander Ghindin - Boston: s.I.:Isabella Stewart Gardner Museum.

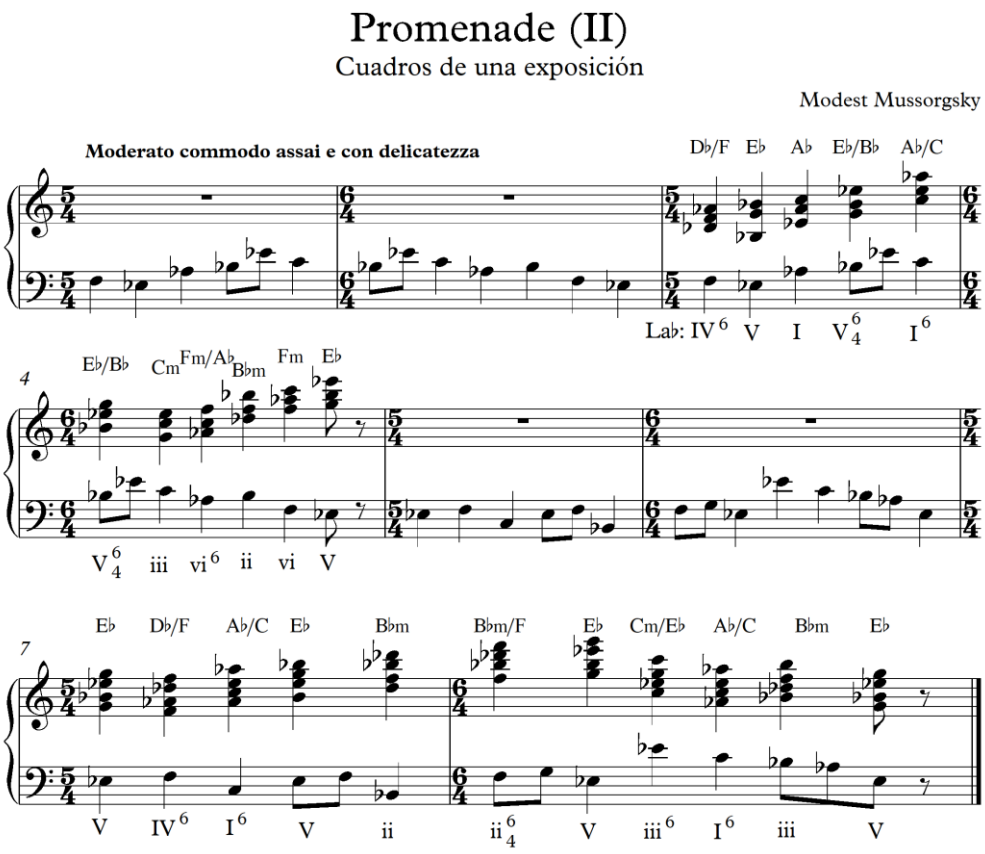


Figura 5.16. Partitura del fragmento analizado de *Promenade (II)* de la suite *Cuadros de una exposición* (Mussorgsky). Se indica sobre los pentagramas la anotación de los acordes y debajo el análisis funcional.

Se ha realizado esta operación utilizando los patrones triada mayor. La explicación reside en la serie armónica: todas las notas contienen todos los factores de su acorde mayor (con tónica en la nota dada en la melodía). Por lo tanto, los patrones mayores serán los que menor distancia euclídea arrojen y, en consecuencia, nos situamos en el peor caso, el que tiene mayor riesgo de error.

Como cabía esperar, vemos que es más débil la correlación del patrón triada con una línea melódica (monofónica), pues el algoritmo tan sólo está detectando los parciales del piano y no una armonía compuesta por varias voces sonando simultáneamente.

De los resultados podemos inferir un valor inicial que utilizaremos para establecer un **umbral de detección de acorde**, que permita diferenciar la situación monofónica de la polifónica y, por tanto, permitir que el algoritmo contemple la no-detección. En este caso, se verá reflejado en el fichero de salida con la etiqueta `'N'`. Este carácter puede implicar tanto situaciones de silencio como presencia de líneas monofónicas en los instantes en los que aparezca. Fijamos el valor del umbral de detección de acorde en

$$\alpha = 0.9 \ ,$$

por encima del cual se considerará que no está sonando ningún acorde. Más adelante consideraremos otros posibles valores para este umbral.

5.5.6. Formato del fichero de salida

La salida del algoritmo se escribirá en un fichero en formato .csv (*comma separated vector*), que es básicamente una lista de elementos, valores y cadenas de caracteres, separados por comas. El mismo formato en el que Sonic Annotator genera el cromagrama, aunque en ese caso tan solo contenía números.

La estructura del fichero será

```
instante_de_tiempo_1  etiqueta_de_acorde_1
instante_de_tiempo_2  etiqueta_de_acorde_2
          …                    …
instante_de_tiempo_M  etiqueta_de_acorde_M
```

para los *M* instantes de tiempo discreto que tenía el cromagrama.

Se le dará ese formato en dos columnas a los elementos del vector haciendo una pequeña operación para cambiar los puntos decimales (`.`) por comas (`,`) y las comas de separación del vector por puntos y coma (`;`). Luego veremos que esto puede resultar más adecuado para observar los datos en Excel, pero menos para que MATLAB procese las cadenas.

5.7. Versión final del sistema de detección de acordes

Para dotar al sistema de una mayor precisión, hemos incorporado una funcionalidad adicional que permite estabilizar la decisión del acorde más probable, basándonos en la estimación del tempo y los *onsets*. Junto con el umbral de detección de acorde, definido para el prototipo, esta condición de estabilidad supone un nuevo refinamiento que producirá modificaciones en la etapa de decisión (Grosche, et al., 2010; Grosche y Müller, 2011a, 2011b).

Como ya anunciábamos, en esta ocasión hemos ampliado el vocabulario armónico para incluir más tipos de acordes triada, además de redefinir los patrones o *templates* de cada acorde de forma que se acerquen más a la realidad. Así pues, consideraremos un número elevado de armónicos por cada clase de altura activada, que contribuirá a conseguir un mejor resultado a la hora de evaluar qué acorde tipo representa mejor el perfil armónico observado en un vector de croma.

En esta ocasión hemos modificado los patrones de acorde a únicamente vectores de 12 elementos definitorios del patrón de cada tipo de acorde, sin emplear matrices para almacenar todos los desplazamiento de dicho vector patrón (lo cual parecía algo desproporcionado en la práctica). Obtendremos las correlaciones rotando el propio vector de croma y manteniendo el PCP patrón, de forma equivalente a si rotásemos éste.

Como técnica de pre-procesado, se ha incluido la separación armónico/percusiva para considerar únicamente la componente armónica de la señal musical y facilitar así una mejor detección (Ono, et al., 2008; Fitzgerald, 2010; Valero, 2012). Utilizaremos la técnica sobre los temas de The Beatles para tratar de mejorar la estimación sobre estos audios.

Asimismo, en la versión final hemos integrado la extracción de características de croma en nuestro algoritmo en entorno MATLAB, al igual que la separación armónico/percusiva. Ya puede ejecutarse todo lo propuesto de una sola vez.

Se ha incluido la posibilidad de deshabilitar y controlar cada una de estas mejoras con una simple variable de activación, para así comparar su impacto real en la estimación de los acordes (cuestión ampliamente abordada por Cho y Bello, 2014).

5.7.1. Condición de estabilidad basada en los pulsos locales predominantes

Como hemos visto, uno de los problemas del sistema inicial era la poca estabilidad de la etapa de decisión. Al detectarse un acorde, la correlación con el instante siguiente podía ser prácticamente nula si los resultados de las distancias euclídeas entre patrón ideal y PCP real estaban muy reñidos, con la consiguiente basculación del estado aunque no hubiese comenzado a sonar un acorde distinto. Es más, no tiene sentido alguno que cambie el estado de la decisión en un intervalo de tiempo inferior al ritmo armónico medio de una pieza. Estimar un nuevo acorde cada $\Delta t = 0.0464$ s, que es la resolución temporal del sistema y ni tan siquiera sobrepasa el umbral de percepción de 50 ms, no es en absoluto razonable.

Para dar una solución a este problema se pensó en imponer una condición al sistema que fuera dependiente del tempo del audio de entrada. Planteamos la hipótesis inicial de que el ritmo armónico no podrá ser inferior a un tiempo o pulso musical, es decir, el cambio de acorde sólo podrá darse cuando haya transcurrido un tiempo completo. Era necesario realizar un seguimiento del pulso (o *beat tracking*).

El seguimiento del pulso se divide generalmente en dos etapas. Por un lado, la estimación de *onsets* o tiempos de inicio de nota, para lo cual la propuesta más popular en la actualidad consiste en estimar los cambios energéticos en el espectro mediante la llamada curva de novedad (*novelty curve*). En segundo lugar, y partiendo de dicha curva, se realiza una detección de patrones locales en forma de

trenes de pulsos cuasi-periódicos. Esta tarea se dificulta especialmente si se dan cambios de tempo (*accelerando*, *ritardando*, calderones), de manera directamente proporcional al grado de discontinuidad en el tempo. Puede llegar a hacerse inviable encontrar una periodicidad si la alteración de la frecuencia del pulso es excesiva (como el *rubato* tan marcado del estilo pianístico romántico).

Finalmente, revisando la literatura, decidimos partir de la propuesta de Grosche y Müller (2011a) para obtener una versión de la curva de novedad en la que se ha enfatizado el carácter periódico de la misma. Utilizan métodos basados en la DFT y en la función de autocorrelación para comparar la curva de novedad con patrones sinusoidales desplazados. Aplicando la técnica de solapamiento y suma y utilizando una herramienta conocida como tempograma (*tempogram*), se llega a la versión mejorada de la curva de novedad: la curva de pulsos locales predominantes (*Predominant Local Pulses*, PLP), que correlaciona en alto grado con la información de los *onsets*. Se está detectando en el dominio espectral el efecto ruidoso de banda ancha, de carácter impulsivo, que se produce cada vez ocurre un *onset*.

Utilizaremos el intervalo entre *onsets* (*Inter-Onset-Interval*, IOI) obtenido a partir de la curva PLP para imponer una condición de estabilidad en la detección de acordes. No podrá detectarse un nuevo acorde si no ha transcurrido un tiempo equivalente al IOI (Figura 5.17). Nos hemos valido para ello del Tempogram Toolbox de MATLAB diseñado por Grosche y Müller (2011b). Nos referimos al trabajo original de estos autores para el detalle matemático (Grosche, et al., 2010; Grosche y Müller, 2011a).

Alternativamente, podríamos haber empleado una detección de ritmo armónico, como la propuesta de Goto y Muraoka (1999), para dar solución al mismo problema de estabilidad. No obstante, entendemos que esta opción es más restrictiva, pues trabaja en una rejilla más fina de tiempo, incluso al nivel de *tatum*. Se corre el riesgo, en este caso, de confundirse la línea melódica, que contiene notas extrañas al acorde, con la armonía definitoria del pasaje. Un aspecto, por otra parte, altamente subjetivo: distintos oyentes responderían de manera diferente a qué notas están integradas o no en el acorde (en Mauch y Dixon, 2008, estos autores tratan de dar una solución a dicho problema concreto). Esta segmentación de pulsos locales predominantes es, no obstante, muy precisa, como demuestran los numerosos experimentos llevados a cabo en Grosche y Müller (2011a). Es posible detectar hasta los *onsets* energéticamente más débiles producidos por instrumentos cuya envolvente tiene un tiempo de ataque lento (como la familia de cuerda), e igualmente se obtiene una respuesta adecuada en presencia de cambios de tempo constantes producidos por los procedimientos expresivos más comunes.

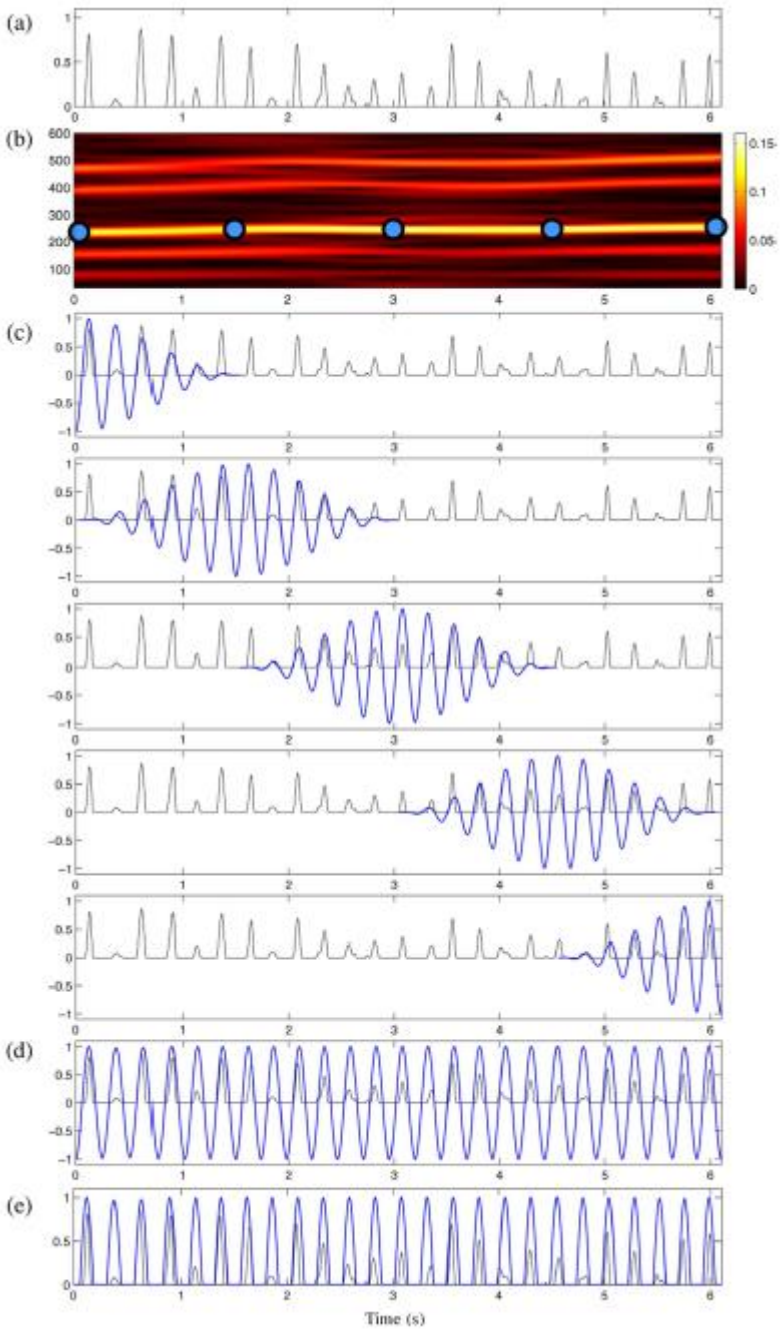


Figura 5.17. Obtención de la curva PLP. (a) Curva de novedad. (b) Tempograma de amplitud de una señal de tempo variable. (c) Patrones sinusoidales óptimos (tamaño de patrón de 3 s) con los que se buscan los máximos. (d) Solapamiento y suma de todos los patrones sinusoidales. (e) Curva PLP obtenida tras una rectificación de media onda (Grosche y Müller, 2011a).

5.7.2. Patrones de acordes no ideales

Para la versión final del algoritmo hemos querido considerar series armónicas de cada una de las notas presentes en los acordes. De esta manera se pueden obtener patrones más realistas ya que, de alguna manera, modelamos los armónicos presentes en instrumentos reales. Por ejemplo,

considerando el acorde de LA Mayor y utilizando los seis primeros armónicos para cada nota considerada, aparecen nuevos elementos en el patrón básico (Tabla 5.5).

	Armónico					
Nota inicial	**1**	**2**	**3**	**4**	**5**	**6**
LA	LA	LA	MI	LA	DO#	MI
DO#	DO#	DO#	SOL#	DO#	FA	SOL#
MI	MI	MI	SI	MI	SOL#	SI

Tabla 5.5. Serie de seis armónicos sobre la triada de LA Mayor.

Estos armónicos no tendrán todos la misma amplitud en el espectro, sino que irán decayendo. A continuación presentamos los PCP de los acordes triada incluidos en el vocabulario armónico adoptado: `{maj, min, dim, aug, sus4}`, indicando el peso de cada clase de altura en el PCP normalizado (Figuras 5.18-22).

Se han considerado hasta 11 armónicos de la serie armónica de cada voz de la triada, utilizado las relaciones del sistema de afinación justo para calcular las diferentes frecuencias (Tabla 3.6). El séptimo armónico, que no coincide de manera exacta con ninguna clase de altura de la escala cromática (diríamos que está ligeramente "desafinado"), ha sido asimilado al semitono superior. El error cometido por utilizar estas aproximaciones es realmente pequeño.

Sí debemos apuntar que estos patrones de acordes triada se encuentran en estado fundamental. La asignación de pesos cambiaría notablemente si el cálculo se hiciese con una nota diferente en la voz del bajo, es decir, empleando inversiones del acorde triada. Energéticamente tendría más importancia la altura de la voz del bajo y sus armónicos.

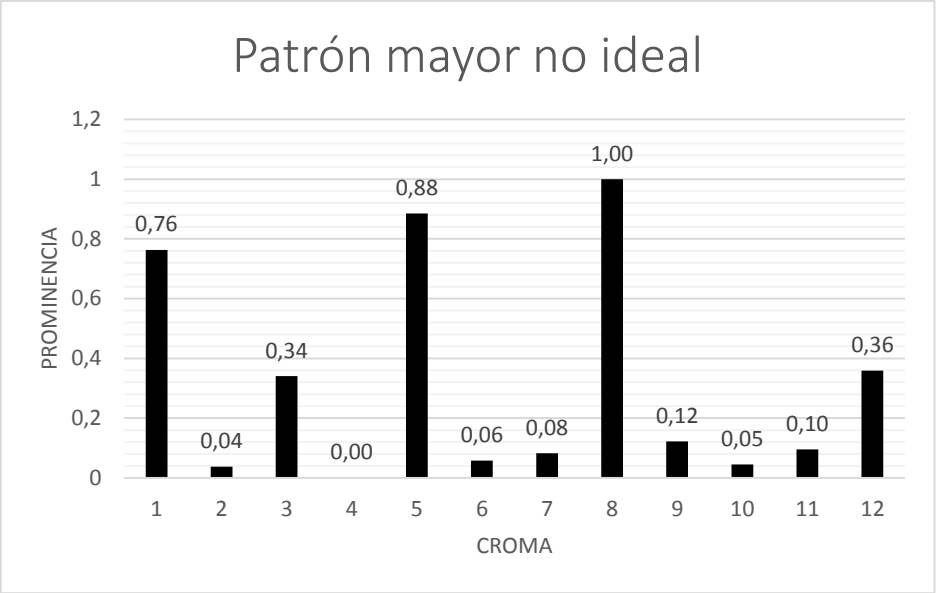


Figura 5.18. PCP del nuevo patrón mayor, formado por la superposición de 11 armónicos de amplitud decreciente por cada factor del acorde.

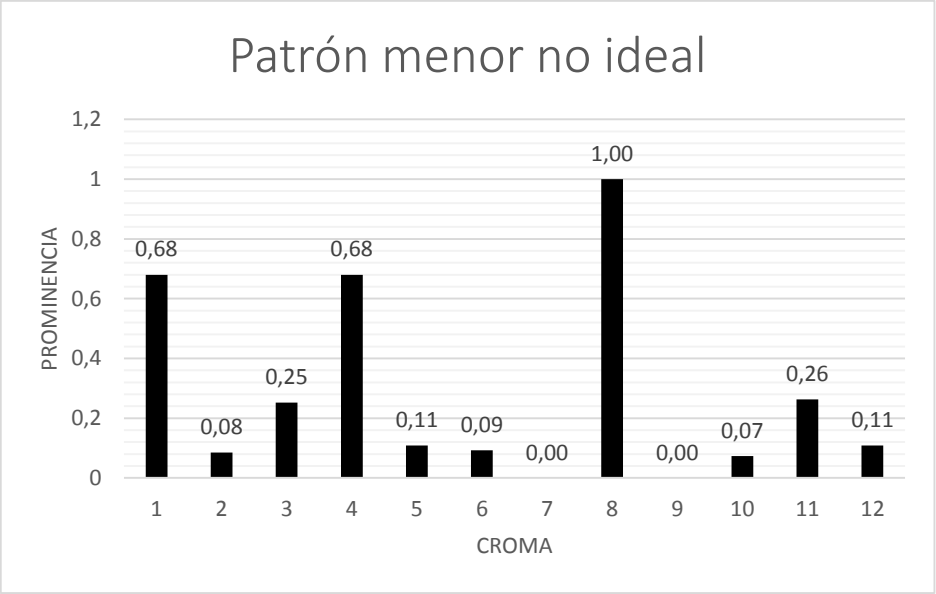


Figura 5.19. PCP del nuevo patrón menor, formado por la superposición de 11 armónicos de amplitud decreciente por cada factor del acorde.

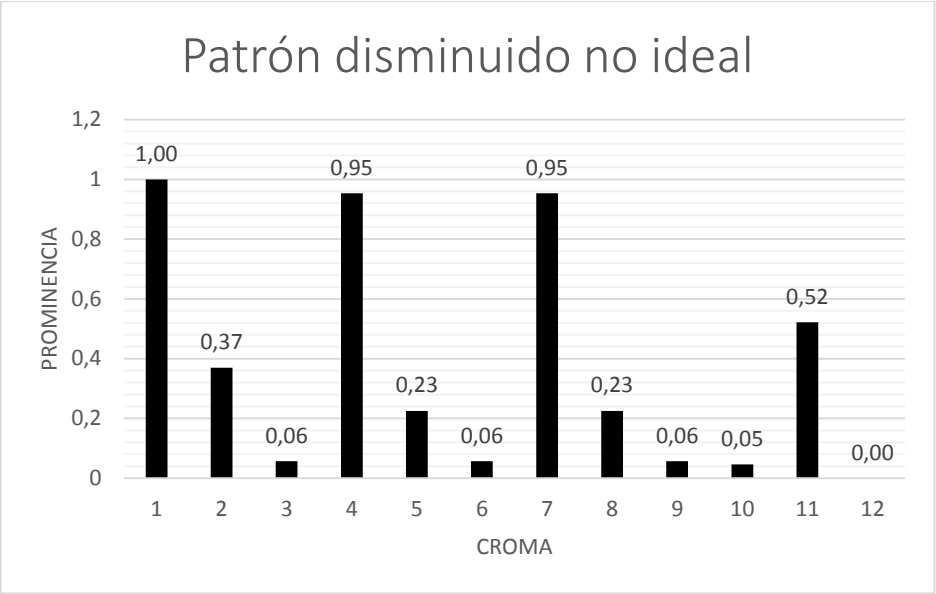


Figura 5.20. PCP del nuevo patrón disminuido, formado por la superposición de 11 armónicos de amplitud decreciente por cada factor del acorde.

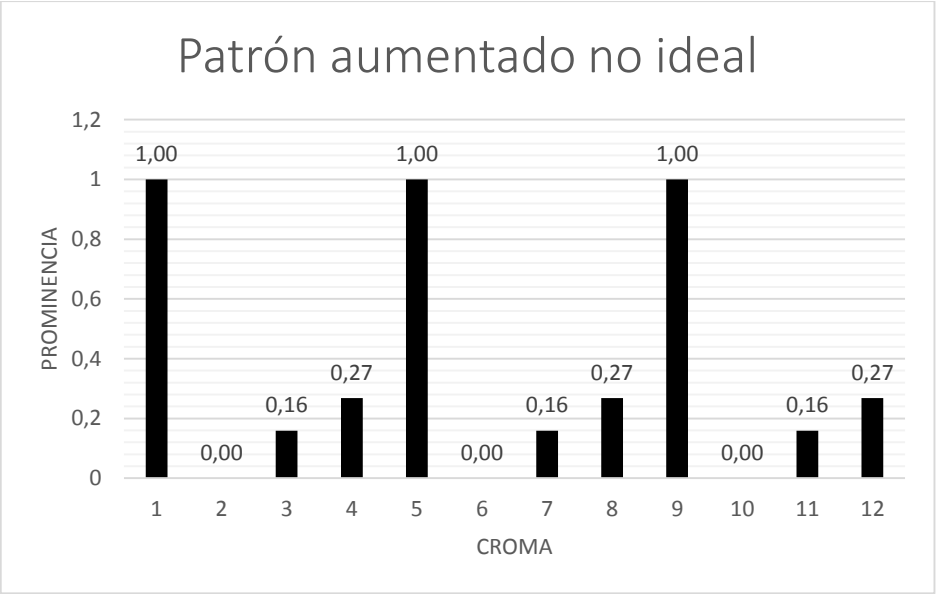


Figura 5.21. PCP del nuevo patrón aumentado, formado por la superposición de 11 armónicos de amplitud decreciente por cada factor del acorde.

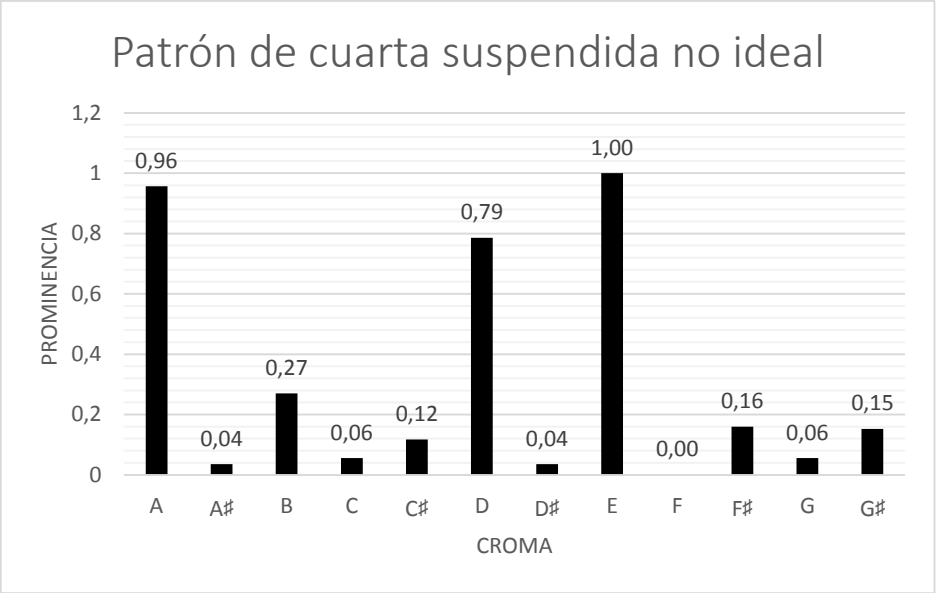


Figura 5.22. PCP del nuevo patrón de cuarta suspendida, formado por la superposición de 11 armónicos de amplitud decreciente por cada factor del acorde.

5.7.3. Separación armónico-percusiva

En cuanto al pre-procesado empleado para enfatizar las características de interés que desean segmentarse, como es el caso de la información armónica, hemos decidido aplicar a las señales un proceso de separación armónico-percusiva. Como veremos, es especialmente necesario en el caso de las músicas de género pop-rock, en las que la contribución inarmónica de la batería es casi omnipresente. Nuestro objetivo es utilizar esta componente armónica extraída de la señal como entrada optimizada para el cromagrama.

El algoritmo que hemos empleado se basa en una técnica no lineal, como es el filtro de mediana, aplicada en el dominio espectral (Fitzgerald, 2010). Puede ser clasificado como un algoritmo fiel a un modelo de tipo discriminativo, frente a modelos independientes de la fuente sonora (*blind-source*) o modelos basados en patrones percusivos en tiempo o frecuencia (*match and adapt*). El enfoque discriminativo asume que hay algún tipo de rasgo que diferencia la componente percusiva de la armónica, y varía en función de las hipótesis planteadas.

La hipótesis principal del algoritmo de Fitzgerald se basa en el enfoque de Ono, et al. (2008). En éste se dice que la componente armónica tiene normalmente una altura estable y describe lóbulos paralelos en el espectrograma con envolventes temporales suaves, mientras que la energía de un tono percusivo se concentra en un ventana muy corta y forma un lóbulo vertical con un gran ancho de banda de la envolvente espectral.

La afirmación es muy cercana a la realidad, como podemos comprobar en la Figura 5.23: la percusión se produce por impulsos e introduce información de banda ancha muy localizada temporalmente

(líneas verticales en el espectrograma) y el resto de instrumentos producen patrones armónicos que se extienden más en el tiempo (líneas horizontales en el espectrograma) (Valero, 2012).

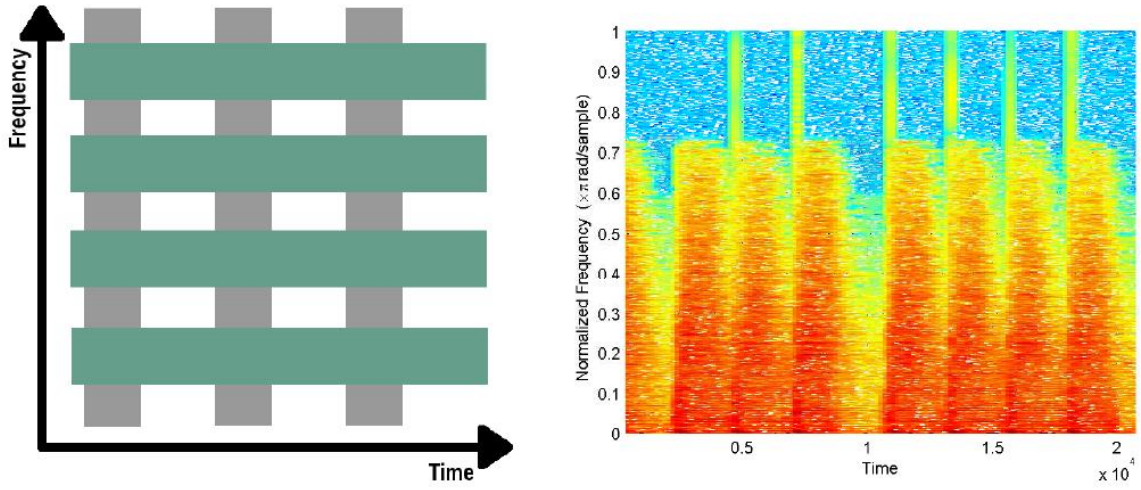


Figura 5.23. A la izquierda, esquema que plantea la hipótesis principal del algoritmo de Fitzgerald; a la derecha, un espectrograma real de un fragmento del tema *YYZ* de Rush (Valero, 2012).

Se ha realizado una pequeña modificación del algoritmo para se aplique recursivamente a todos los archivos de una carpeta en la que están contenidos los archivos de audio bajo estudio. En este caso, no nos interesa la componente percusiva, por lo que deshabilitaremos su escritura (matizaremos esto en breve). La función tan solo creará un fichero Harmonic_nombre_del_audio.wav para cada elemento de la colección de datos.

Podría ser interesante para músicas diferentes al pop-rock, y a otras músicas contemporáneas que hacen uso de batería y ritmo rígido, utilizar la componente percusiva extraída de la señal como parámetro de entrada de la función PLP. De esta manera conseguiríamos una doble ventaja: enfatizar la parte armónica de cara a extraer el cromagrama y también realzar el contenido rítmico de la señal de cara a segmentar los pulsos locales predominantes asociados a los *onsets*. El problema, creemos, es que en las músicas contemporáneas los PLP no están tan correlacionados con el ritmo armónico. Si lo que queremos segmentar es exclusivamente el tempo, y con ello tener una referencia para estabilizar el instante de decisión, desde luego sí es una opción interesante. Realizaremos un experimento al respecto cuando evaluemos el sistema.

En definitiva, aplicando el algoritmo de Fitzgerald obtenemos una versión optimizada de los audios que servirán de entrada para la extracción de características de croma (Figura 5.24). En este ejemplo inicial se realizaron los procesos se pre-procesado y extracción de cromagrama secuencial pero no automatizadamente.

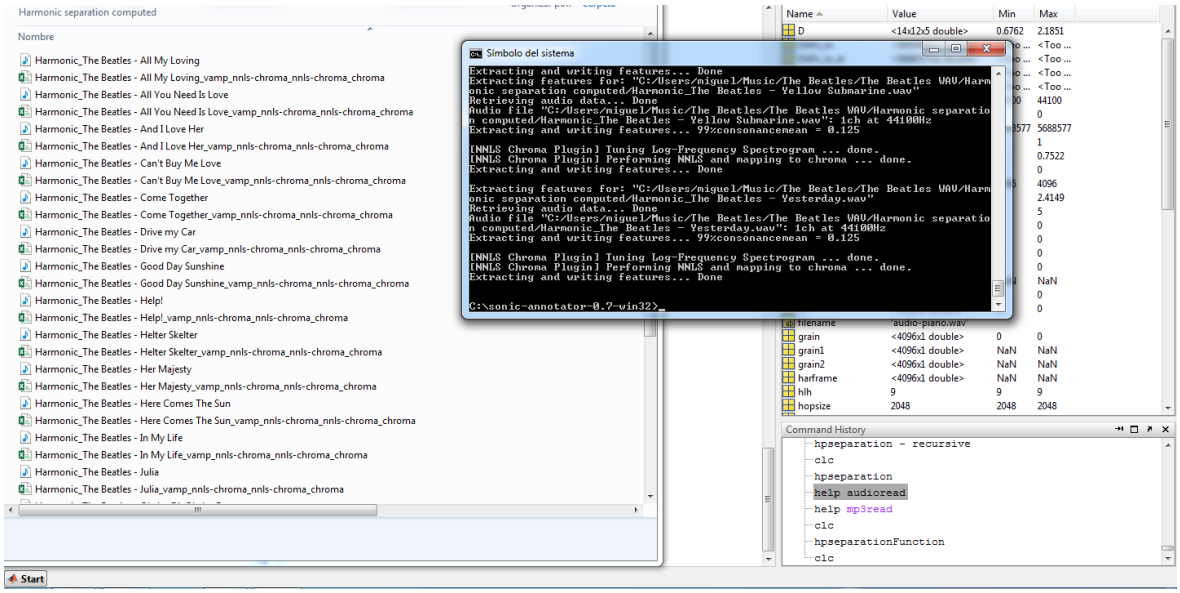


Figura 5.24. Obtención de los cromagramas NNLS a partir de las componentes armónicas extraídas de 20 temas de The Beatles, tras aplicar la separación armónico-percusiva.

5.7.4. Post-procesado de la estimación

Hemos decidido aplicar una técnica de post-procesado a la salida de la función de decisión, para suavizar los resultados obtenidos en la estimación de acordes. El objetivo es, en resumidas cuentas, evitar basculaciones extrañas o demasiado frecuentes en la decisión.

Un filtro de mediana se encargará de realizar esta tarea, una técnica no lineal de la que ya hemos venido hablando, usada comúnmente en procesado de señal para reducir ruido. En el ámbito de la estadística, la mediana representa el valor de la variable de posición central en un conjunto de datos ordenados, en nuestro caso, nombres de acordes.

No es una cuestión trivial aplicar un filtro de mediana a un vector de cadenas de caracteres (*char* o *string* de MATLAB). Es necesario elaborar alguna estrategia para conseguirlo. Asignaremos a cada posible etiqueta de acorde un valor numérico unívoco y esa será la información que se filtre, para luego volver a ser convertida en etiquetas.

Para ilustrar esta aplicación realizamos una llamada a la función de filtro de mediana que hemos diseñado con el vector ejemplo siguiente, que bien podría ser la salida de la función de decisión:

```
v =
   {'A:maj';'A:min';'A:maj';'G:maj';'A:maj';'E:min';'A:maj';'Bb:aug';'C#:
   maj'};
```

Con un valor de δ = 3 muestras (tamaño de ventana deslizante), obtenemos el siguiente resultado:

```
tags =
  {'A:maj';'A:maj';'A:maj';'A:maj';'A:maj';'A:maj';'A:maj';'A:maj';'A:ma
  j'};
```

Se trata, por tanto, de una nueva forma de depurar la estimación a la salida de la función de decisión, eliminando "ruido" del vector de etiquetas de acorde obtenido.

5.7.5. Integración de la extracción de características de croma en MATLAB

Anteriormente teníamos que realizar por separado la etapa de extracción de características de croma, utilizando la consola de comandos. Ocurría lo mismo en el caso de la separación armónico/percusiva, en las primeras pruebas que hicimos, la ejecutábamos en un fichero diferente de MATLAB.

Esto era un inconveniente, por una parte, porque era incómodo, pero automatizar y secuenciar todas las operaciones del sistema es especialmente importarse si va a tratarse con una base de datos de gran tamaño, para economizar tiempo y recursos.

MATLAB ofrece la posibilidad de conectar con el símbolo del sistema y encomendar la ejecución de tareas a la consola. Sonic Annotator puede ser controlado desde el *command window* utilizando el carácter `!` o desde una función o *script* recurriendo a la función `system`. El fichero de configuración conf.n3 debe estar en la carpeta de ejecución del programa en MATLAB.

Por ejemplo, en la línea del *command window* puede escribirse:

```
!C:\sonic-annotator-0.7-win32\sonic-annotator.exe  -t  conf.n3
     Data_wav\UMAPiano-DB-C3E3G3-NO-F.wav -w csv
```

Para ordenar la extracción del cromagrama de la señal `UMAPiano-DB-C3E3G3-NO-F.wav`.

Presentamos, de manera separada, una función en MATLAB que permite ejecutar una transformación de Sonic Annotator también sobre todos los archivos de audio de una carpeta completa y gestionar los directorios de origen y destino de los ficheros (.wav y .csv, respectivamente), para ofrecer diferentes opciones al usuario. Pueden encontrarse todas estas funciones adicionales en el Anexo B.

5.8. Evaluación del sistema

5.8.1. Método de evaluación

La estrategia de evaluación (*metrics*) diseñada se basa en el enfoque dado a los algoritmos de inducción de tonalidad en el MIREX 2005, con su correspondiente adaptación a la tarea de estimación de acordes. En dicha convocatoria se proponía una estrategia formal de evaluación ante

la falta de consenso hasta el momento, que fuese sencilla y fácilmente aplicable a la amplia variedad de algoritmos que se requería evaluar.

Se trata de asignar una puntuación a la calidad de la estimación muestra a muestra, sumar estos valores y hallar el promedio para obtener la precisión del sistema en porcentaje. Algunas evaluaciones han optado por dar exclusivamente valor a los verdaderos positivos, es decir, a las estimaciones que se corresponden exactamente con las anotaciones de referencia (*ground-truth*). En dicha conferencia se rechaza este enfoque, que se consideraba excesivamente rígido para englobar una colección heterogénea de algoritmos, y se propone el siguiente criterio de valoraciones:

Relación con la tonalidad correcta	**Puntos**
Mismo acorde	1
Quinta justa	0.5
Relativo mayor/menor	0.3
Paralelo mayor/menor	0.2

Tabla 5.6. Método de evaluación para inducción de tonalidad en MIREX 2005, usado como referencia para diseñar nuestro método.

La tarea que nos ocupa es distinta a la que motivó esta tabla en particular, nos ocupamos de la estimación de acordes y no de hallar el tono. Por lo tanto, realizaremos las adaptaciones que creemos oportunas para obtener un índice de rendimiento fiable en la tarea de estimación de acordes.

En el enfoque adoptado se valora positivamente que el algoritmo detecte el acorde paralelo mayor o menor, dado que tan solo se está equivocando en una de las tres notas que componen el acorde tríada, la tercera, que define el modo. Aglutinaremos este caso con otros y premiaremos en cierta medida que se detecte la voz fundamental correctamente (sin tener en cuenta la cualidad del acorde que figura en las anotaciones de referencia).

El relativo mayor o menor también se valora, pues de nuevo tan solo hay una nota errónea; se premia menos porque se entiende que es más grave que no se acierte la tónica, debido a que es un factor muy reforzado por los armónicos.

Ciertas decisiones en el método de evaluación se deben al evidente desajuste entre nuestro vocabulario armónico y el que se utiliza en las canciones de The Beatles que hemos incluido en nuestra colección. No puede hacerse evaluación alguna si no se es capaz de premiar, de algún modo, que se estime una triada mayor cuando en la anotación figuraba un acorde dominante. Por otro lado, lo mismo ocurre con las inversiones. Si el sistema consigue detectar una tónica pese a no estar en la voz del bajo, esto ha de tenerse en cuenta.

No consideraremos, por tanto, inversiones, tensiones o séptimas. Y recordamos que, debido a la sintaxis distinta que parece emplear *Isophonics* para notar los acordes, tendremos que dar dos pasos: despreciar la información adicional sobre los acordes que no puede comprender nuestro sistema (por no encontrarse en el vocabulario armónico) y adaptar la sintaxis de las anotaciones de referencia a la propuesta de estandarización de Harte (Harte, et al., 2005).

Volviendo a la tabla 5.4, creemos que no tiene sentido valorar para estimación de acordes que el algoritmo se decante por la quinta justa. Sí sería bueno hacerlo para la tarea de inducción de tonalidad puesto que se trata de un grado crucial para detectar el tono, pero entendemos que para la estimación de acordes esto no es positivo, ya que se estaría cometiendo un error de tono o semitono en dos factores de acorde.

Nuestro criterio de evaluación queda de la siguiente forma:

Relación con el acorde correcto	**Puntos**
Mismo acorde	1
Misma tónica	0.5
Relativo mayor/menor	0.3

Tabla 5.7. Método de evaluación adoptado.

5.8.2. Resultados

En primer lugar presentaremos los resultados de rendimiento de la estimación archivo por archivo, y más tarde un promedio global de las colecciones UMA Piano Data Base y The Beatles por separado (ya que no tendría sentido presentarlo de ambas colecciones globalmente, se trata de problemas distintos). No hemos considerado los acordes ideales porque en las pruebas que fuimos haciendo con prototipo y sistema final vimos que, en todos los casos, la precisión era inferior, y en cambio hemos preferido comparar otros factores que influyen en la estimación.

Colección de acordes de piano la UMA Piano Data Base

Parámetros de la estimación:

```
control.preProcess = 0;
control.chromagramComputation = 0;
control.idealTemplates = 0;
control.PLPComputation = 0;
control.postProcess = 0; %probamos ambos casos,
control.postProcess = 1; %con y sin filtro de mediana (delta = 3)
alpha = 0.95;
```

Nº	Nombre de archivo	Descripción	Precisión de la estimación (postProcess = 0)	Precisión de la estimación (postProcess = 1)
1	UMAPiano-DB-C3E3G3-NO-F.wav	maj; rango medio; *forte*	**95%**	**95%**
2	UMAPiano-DB-C3E3G3-NO-M.wav	maj; rango medio; *mezzo*	**92.5%**	**92.5%**
3	UMAPiano-DB-C3E3G3-NO-P.wav	maj; rango medio; *piano*	**95%**	**100%**
4	UMAPiano-DB-C1C3E3-NO-F.wav	maj; rango grave; *forte*	**76.7%**	**75.8%**
5	UMAPiano-DB-C1C3E3-NO-M.wav	maj; rango grave; *mezzo*	**69.4%**	**66.5%**
6	UMAPiano-DB-C1C3E3-NO-P.wav	maj; rango grave; *piano*	**82.7%**	**74.6%**
7	UMAPiano-DB-G#6C7Eb7-NO-F.wav[35]	maj; rango agudo; *forte*	**39.5%**	**37.5%**
8	UMAPiano-DB-G#6C7Eb7-NO-M.wav	maj; rango agudo; *mezzo*	**25.8%**	**22.5%**
9	UMAPiano-DB-G#6C7Eb7-NO-P.wav	maj; rango agudo; *piano*	**21.9%**	**21.9%**
10	UMAPiano-DB-C3Eb3G3-NO-F.wav	min; rango medio; *forte*	**95%**	**90%**
11	UMAPiano-DB-C3Eb3G3-NO-M.wav	min; rango medio; *mezzo*	**95%**	**90%**
12	UMAPiano-DB-C3Eb3G3-NO-P.wav	min; rango medio; *piano*	**97.5%**	**97.5%**
13	UMAPiano-DB-C1C3Eb3-NO-F.wav	min; rango grave; *forte*	**95.8%**	**93.8%**
14	UMAPiano-DB-C1C3Eb3-NO-M.wav	min; rango grave; *mezzo*	**94.8%**	**93.8%**
15	UMAPiano-DB-C1C3Eb3-NO-P.wav	min; rango grave; *piano*	**93.8%**	**93.8%**
16	UMAPiano-DB-C6G6Eb7-NO-F.wav	min; rango agudo; *forte*	**42.5%**	**45%**
17	UMAPiano-DB-G6G6Eb7-NO-M.wav	min; rango agudo; *mezzo*	**42.5%**	**45%**
18	UMAPiano-DB-G6G6Eb7-NO-P.wav	min; rango agudo; *piano*	**47.5%**	**45%**
19	UMAPiano-DB-E6C7G7-NO-F.wav	maj/3; rango medio-alto; *forte*	**43.8%**	**44.8%**
20	UMAPiano-DB-E6C7G7-NO-M.wav	maj/3; rango medio-alto; *mezzo*	**42.5%**	**46%**
21	UMAPiano-DB-E6C7G7-NO-P.wav	maj/3; rango medio-alto; *piano*	**25%**	**27.5%**
22	UMAPiano-DB-C3A3E4-NO-F.wav	min/3; rango medio; *forte*	**90%**	**95%**
23	UMAPiano-DB-C3A3E4-NO-M.wav	min/3; rango medio; *mezzo*	**90%**	**95%**

[35] Advertimos de que los nombres de los archivos que contienen notas alteradas con sostenido en el directorio 'Data_wav' tuvieron que ser modificados (carácter 's' en lugar de '#') debido a que Sonic Annotator no entiende este carácter en los nombres de ficheros y falla al extraer los cromagramas. En el código lo hemos previsto, de todas formas, y ahora se realiza esta sustitución automáticamente.

24	UMAPiano-DB-C3A3E4-NO-P.wav	min/3; rango medio; *piano*	**90%**	**95%**
25	UMAPiano-DB-E3A3C#4-NO-F.wav	maj/5; rango medio; *forte*	**35%**	**37.5%**
26	UMAPiano-DB-E3A3C#4-NO-M.wav	maj/5; rango medio; *mezzo*	**37.5%**	**37.5%**
27	UMAPiano-DB-E3A3C#4-NO-P.wav	maj/5; rango medio; *piano*	**35%**	**37.5%**
28	UMAPiano-DB-E3A3C4-NO-F.wav	min/5; rango medio; *forte*	**35%**	**37.5%**
29	UMAPiano-DB-E3A3C4-NO-M.wav	min/5; rango medio; *mezzo*	**35%**	**37.5%**
30	UMAPiano-DB-E3A3C4-NO-P.wav	min/5; rango medio; *piano*	**37.5%**	**37.5%**
31	UMAPiano-DB-C3Eb3Gb3-NO-F.wav	dim; rango medio; *forte*	**95%**	**90%**
32	UMAPiano-DB-C3Eb3Gb3-NO-M.wav	dim; rango medio; *mezzo*	**92.5%**	**90%**
33	UMAPiano-DB-C3Eb3Gb3-NO-P.wav	dim; rango medio; *piano*	**97.5%**	**97.5%**
34	UMAPiano-DB-C1Eb2Gb2-NO-F.wav	dim; rango grave; *forte*	**37.5%**	**35.4%**
35	UMAPiano-DB-C1Eb2Gb2-NO-M.wav	dim; rango grave; *mezzo*	**37.5%**	**35.4%**
36	UMAPiano-DB-C1Eb2Gb2-NO-P.wav	dim; rango grave; *piano*	**37.5%**	**35.4%**
37	UMAPiano-DB-C5Gb5Eb6-NO-F.wav	dim; rango agudo; *forte*	**55%**	**45%**
38	UMAPiano-DB-C5Gb5Eb6-NO-M.wav	dim; rango agudo; *mezzo*	**72.5%**	**67.5%**
39	UMAPiano-DB-C5Gb5Eb6-NO-P.wav	dim; rango agudo; *piano*	**45%**	**37.5%**
40	UMAPiano-DB-C3E4G#4-NO-F.wav	aug; rango medio; *forte*	**62.5%**	**60%**
41	UMAPiano-DB-C3E4G#4-NO-M.wav	aug; rango medio; *mezzo*	**70%**	**67.5%**
42	UMAPiano-DB-C3E4G#4-NO-P.wav	aug; rango medio; *piano*	**42.5%**	**37.5%**
43	UMAPiano-DB-C2E2G#2-NO-F.wav	aug; rango grave; *forte*	**37.5%**	**37.5%**
44	UMAPiano-DB-C2E2G#2-NO-M.wav	aug; rango grave; *mezzo*	**37.5%**	**37.5%**
45	UMAPiano-DB-C2E2G#2-NO-P.wav	aug; rango grave; *piano*	**37.5%**	**37.5%**
46	UMAPiano-DB-C5G#6Eb7-NO-F.wav	aug; rango agudo; *forte*	**37.5%**	**37.5%**
47	UMAPiano-DB-C5G#6Eb7-NO-M.wav	aug; rango agudo; *mezzo*	**27.5%**	**22.5%**
48	UMAPiano-DB-C5G#6Eb7-NO-P.wav	aug; rango agudo; *piano*	**37.5%**	**37.5%**
49	UMAPiano-DB-C4F4G4-NO-F.wav (*)[36]	sus4; rango medio; *forte*	**37.5%**	**30%**
50	UMAPiano-DB-C4F4G4-NO-M.wav (*)	sus4; rango medio; *mezzo*	**45%**	**45%**
51	UMAPiano-DB-C4F4G4-NO-P.wav (*)	sus4; rango medio; *piano*	**57.5%**	**60%**
52	UMAPiano-DB-C2F2G2-NO-F.wav (*)	sus4; rango grave; *forte*	**46.9%**	**51%**

[36] Los ficheros marcados con asterisco, correspondientes a las triadas de cuarta suspendida (sus4) no están contenidos en la base de datos de la UMA como tal; los hemos generado mezclando el intervalo CXFX con la nota GX, contenidos en los directorios de número de polifonía 2 y 1, respectivamente.

53	UMAPiano-DB-C2F2G2-NO-M.wav (*)	sus4; rango grave; *mezzo*	**53.1%**	**57.3%**
54	UMAPiano-DB-C2F2G2-NO-P.wav (*)	sus4; rango grave; *piano*	**44.8%**	**51%**
55	UMAPiano-DB-C6F6G6-NO-F.wav (*)	sus4; rango agudo; *forte*	**40%**	**37.5%**
56	UMAPiano-DB-C6F6G6-NO-M.wav (*)	sus4; rango agudo; *mezzo*	**31.3%**	**37.5%**
57	UMAPiano-DB-C6F6G6-NO-P.wav (*)	sus4; rango agudo; *piano*	**35%**	**37.5%**
58	UMAPiano-DB-C1Bb1G2E3-NO-F.wav	dom7; rango grave; *forte*	**20%**	**25%**
59	UMAPiano-DB-C2Bb3E4G4-NO-F.wav	dom7; rango medio; *forte*	**30%**	**32.5%**
60	UMAPiano-DB-C3Bb4E5G5-NO-F.wav	dom7; rango agudo; *forte*	**95%**	**85%**

A la vista de los resultados, vemos unas muy buenas estimaciones para acordes mayores y menores en todas las dinámicas y en los rangos grave y medio. En el registro agudo la inarmonicidad dificulta notablemente la estimación.

Vemos que en muchos casos los acordes de cuarta suspendida se detectan mejor con dinámica *mezzo* y *piano*. La interpretación que le damos es que se excitan menos armónicos y, al ser éste un acorde disonante, eso es positivo para la inteligibilidad del espectro. Algo parecido ocurre con los disminuidos y los aumentados, aunque éstos son todavía más disonantes (y más problemáticos) que el anterior.

Hay situaciones, también, insalvables. En los extremos grave y agudo de la tesitura del piano se da una notable disminución de la armonicidad, lo que se traduce en menor sensación de altura en los acordes. El sistema también tiene este problema de "percepción" para acordes aumentados y disminuidos en registros extremos. Con los de cuarta suspendida sucede otro tanto de lo mismo. El acierto mínimo de 22-30% a menudo se debe a que los instantes de silencio se han segmentado correctamente y, al tratarse de archivos tan cortos (apenas 2 s), porcentualmente pesan mucho.

Hacemos notar, aunque ya salta a la vista, que el post-procesado se comporta de manera bastante imprevisible, y en la mayoría de ocasiones empeora el resultado de la estimación.

Al final hemos incluido algunos casos de inversiones de acordes mayores y menores, y el sistema no responde demasiado bien, salvo la excepción notable del acorde menor en primera inversión (min/3). Se está desechando la información del bajo porque no está en este momento contemplada esa posibilidad, y se busca detectar únicamente el acorde, sin su inversión.

También hemos probado tres casos de séptimas de dominante, por ver la respuesta del sistema si le pedimos que trate de encontrar en ese audio una triada mayor. Tiene un resultado similar (o algo peor, como por otra parte es lógico) al de los acordes disonantes en los registros desfavorecidos.

Como esperábamos, las cuatriadas hacen muy difícil la tarea de correlación con acordes patrón, aunque nos encontramos con el caso sorprendente del registro agudo, donde detecta a la perfección un C:maj. Aunque lo hemos indicado como agudo, en realidad se trata de un registro medio-agudo, que puede que dé facilidades al sistema.

Teniendo en cuenta los resultados de la primera columna (sin pre-procesado), obtenemos el *promedio de la precisión del sistema:* *52.8%*

Colección de The Beatles

Parámetros de la estimación:

```
control.preProcess = 1; %harmonic/percussive separation
control.chromagramComputation = 0;
control.idealTemplates = 0;
control.PLPComputation = 0;
control.postProcess = 0;
alpha = 1.5;
```

Nº	Nombre de archivo	Precisión de la estimación (preProcess = 0)	Precisión de la estimación (preProcess = 1)
1	The_Beatles_A_Taste_of_Honey.wav	**37.1%**	**36.8%**
2	The_Beatles_Across_the_Universe.wav	**4.7%**	**5.3%**
3	The_Beatles_Act_Naturally.wav	**1%**	**1%**
4	The_Beatles_All_My_Loving.wav	**45.9%**	**44.36%**
5	The_Beatles_All_You_Need_Is_Love.wav	**1%**	**1%**

A la vista de los grandes problemas que tiene el sistema para funcionar igualmente bien para dos tipos de señales radicalmente diferentes, decidimos parar aquí la evaluación de los temas de The Beatles.

Nos remitimos de nuevo a nuestros objetivos iniciales, que eran el audio polifónico monotímbrico. El sistema ha tenido que tratar de adaptarse constantemente para contemplar el caso del pop-rock, y esa diversidad de desempeño va más allá del alcance de este trabajo.

5.9. Problemas encontrados y soluciones adoptadas

5.9.1. Umbral de detección de acorde

El umbral de detección para la colección pop-rock arrojaba unos resultados totalmente irreales, con grandes discontinuidades en la decisión debido a que todas las correlaciones resultan más débiles para el sistema final que en el caso del prototipo. En éste tan solo habíamos probado audios sintéticos y ejemplos aislados de piano. Con $\alpha = 0.9$, estos ejemplos segmentaban relativamente bien las situaciones monofónicas y de silencio, pero en las condiciones actuales, aparecen símbolos 'N' injustificadamente.

Vemos que, por ejemplo, *All my loving* (*The Beatles)* arroja un resultado medio de $\bar{d}_E = 1.6682$ para el cálculo de la distancia euclídea. Asumimos, pues, que este umbral es dependiente de gran cantidad de factores y en un principio optamos por incrementarlo para ser capaz de detectar la situación de no-acorde o, por lo menos, no entorpecer el desempeño del resto del sistema. El nuevo valor para el umbral de detección, para tratar con estas señales y bajo las condiciones del sistema final, lo fijamos a $\alpha = 1.5$ para la colección de The Beatles. Tras nuevas pruebas, observamos que este umbral se ajusta de manera adecuada, en el comienzo del tema produce unos pocos símbolos de no-acorde, como por otra parte es habitual en el comienzo de un tema, y a continuación la decisión no presenta discontinuidades.

No obstante, al trabajar sobre acordes de piano los resultados cambian drásticamente (de un 45% a un 80% de precisión para el audio `UMAPiano-DB-C3E3Gs3-NO-F.wav`) si dejamos el umbral de detección a $\alpha = 0.95$. Por ello, finalmente establecemos el umbral condicionado al tipo de señal de entrada. El parámetro puede ajustarse automáticamente en función de si se ha aplicado o no el pre-procesado de separación armónico/percusiva. Esto presumiblemente implica que la canción contiene batería. Con esta hipótesis, que creemos razonable, y una de las variables de control que pensamos introducir en el código para habilitar/deshabilitar componentes, podemos condicionar la asignación de un valor al umbral.

5.9.2. Patrones de acorde y segmentación de silencio

Relacionado con el umbral de detección, hemos tenido problemas a la hora de segmentar los momentos de silencio inicial y final de una señal de audio, que no obstante se mostraban muy evidentes en los primeros vectores de croma del archivo y los últimos. Las prominencias de las clases de altura eran cero en estos instantes.

El umbral de detección de acorde estaba orientado a diferenciar el carácter monofónico del polifónico y, asimismo, detectar el silencio, fijando una distancia euclídea mínima que se tuviera que cumplir. Sí cumplía bien la primera función en general pero, no obstante, en los momentos de silencio inicial

y final sucedía que ningún patrón de acorde correlacionaba lo suficientemente bien con los vectores de croma iniciales (como hemos dicho, son nulos normalmente, por lo que es natural que las distancias resultasen grandes).

Resolvimos este problema incluyendo un patrón de acorde con elementos nulos para representar el silencio. La correlación de los vectores de croma nulos con éste debía ser perfecta, y resultar una distancia mínima que sin duda pasaría la criba del umbral de detección.

Como resultado, conseguimos recuperar los símbolos `'N'` que debían aparecer (y no aparecían antes) en los instantes inicial y final del fichero, que siempre suelen ser instantes de silencio.

Esto ha funcionado muy bien para la base de datos de la UMA, pero ha sido un despropósito para los temas que hemos probado de The Beatles, pues la correlación más fuerte era a menudo la del no-acorde.

5.9.3. Curva de pulsos locales predominantes y filtro de mediana

Nos dimos cuenta de que el algoritmo de Grosche y Müller (2011a, 2011b) no funcionaba para los acordes de piano de la UMA, y en base de diferentes pruebas comprobamos que la duración del fichero era demasiado corta para extraer una curva de novedad (aproximadamente 2 s). Probamos, por ejemplo, a concatenar dos y tres acordes y ejecutar el sistema con ellos. En ambos casos se ejecutaba correctamente el módulo de PLP.

En el caso de la concatenación de tres acordes, el valor de $\delta = 27$ muestras resultaba muy coherente con la duración del archivo (88 muestras en el cromagrama y, recordamos, tres acordes), reflejando claramente el ritmo armónico. Asumiendo esta limitación inherente a este componente de nuestro sistema, fijamos una duración mínima que deben tener los audios para obtener su curva de novedad, tempograma y PLP.

Se obtenían en muchos casos resultados en la evaluación que creíamos mejorables, y fue entonces cuando decidimos incluir el filtro de mediana como método de post-procesado de la estimación, aprovechando para ello el valor obtenido del parámetro δ, expresado en muestras, acerca de la separación de los PLP. Éste es el tamaño de la ventana deslizante que va aplicando filtros de mediana a lo largo del vector de etiquetas de acorde obtenido tras la decisión. En el caso de los temas de The Beatles el resultado suele ser muy bajo, de 2 o 1 muestras, lo cual para el filtrado parece poco útil.

Tratamos de asumir la hipótesis de que los pulsos PLP para temas con batería se agrupaban siguiendo una métrica binaria al estilo de los compases de 4/4, y entonces utilizamos múltiplos potencia de dos de δ (2, 4, 8) de manera que, por ejemplo, considerábamos 4 pulsos como unidad compás (aunque esto depende de si estos se corresponden o no con el *beat*), empleando por tanto ventanas de $\delta' =$

$4 \cdot \delta$ muestras. El resultado de la estimación empeoraba sensiblemente conforme incrementábamos el múltiplo de δ, y en ningún caso mejoraba, por lo que asumimos que en el caso de que δ resultase 1 inhabilitaríamos el filtro de mediana (no iba a tener ningún efecto con tamaño de ventana igual a 1 muestra, la mediana dejaría de tener sentido).

Tras algunas pruebas para la colección de la UMA nos dimos cuenta de que el filtro, en los instantes inicial y final de la señal, entorpecía la segmentación del silencio, y que debía comenzar a actuar cuando terminasen los símbolos `'N'`. Decidimos detectar esta situación y que el proceso comenzara a partir de la primera muestra con un acorde estimado diferente de `'N'`, solucionando así el problema.

5.9.4. Rendimiento de los distintos componentes y uso de campos de control

Para ser capaces de comprobar el rendimiento de los diferentes componentes del sistema, teníamos que ser capaces de deshabilitar de manera rápida y sencilla cada uno de ellos y observar las diferencias en la tasa de acierto de la estimación.

Incluimos con este fin una estructura de escalares `control` que contiene diferentes campos de valor binario o entero (`control.preProcess`, `control.idealTemplates`, `control.PLPComputation`…), explicados en el código, con los que pueden deshabilitarse o controlarse parámetros y funciones del sistema de estimación de acordes. Por ejemplo, puede elegirse entre utilizar patrones ideales o no ideales, indicar si se ha utilizado pre-procesado para que el algoritmo cargue los archivos correctos, habilitar o deshabilitar el cálculo de la curva PLP y hacer lo propio con el filtro de mediana.

5.9.5. Sonic Annotator integrado en MATLAB

Por lo general, funciona correctamente, pero cuando estábamos probando un método de *batching* para procesar muchos cromagramas consecutivos desde MATLAB nos dimos cuenta de que periódicamente sale un mensaje de error de "Sonic Annotator ha dejado de funcionar" que, sin embargo, resulta innocuo en lo que respecta a la ejecución. Dándole a la opción Cancelar (la única que ofrece la ventana emergente) continúa el proceso y se genera el fichero con el cromagrama sin mayores desavenencias. En el sistema final no hemos incluido una opción de ejecución por lotes, pero comentamos el caso porque de vez en cuando ocurre. Escapa a nuestro control y, por otra parte, no sucede nada negativo.

5.9.6. Evaluación del sistema

En primer lugar, comentar que las anotaciones es el único elemento que no hemos podido automatizar; al menos no del todo, requiere escribir las etiquetas de acorde de referencia. En el Anexo B mostramos una pequeña rutina que puede, al menos, generar una base de tiempos en base a los

parámetros de resolución del sistema (recordamos, $\Delta t = 0.0464\,s$, debido a los parámetros de ventana y la frecuencia de muestreo) y tomando como referencia el cromagrama de cada archivo de audio, con lo cual la columna con los *frames* estaría lista y adecuada a las dimensiones del archivo.

También adjuntamos en este anexo otro procedimiento para adaptar la estructura de las anotaciones de Isophonics (tres columnas: $t_{inicial}$, t_{final} y nombre del acorde) a la que nosotros hemos empleado en el proyecto (dos columnas: t_{frame} y nombre de acorde). Antes de poder aplicar esta rutina se han de pasar las anotaciones a formato .csv. En la web de Isophonics se encuentran en .lab, y no hemos encontrado la forma de leer este formato desde MATLAB.

En este sentido, tuvimos varios inconvenientes a la hora de utilizar las anotaciones que proporciona la web de Isophonics para la colección de The Beatles, porque éstas además empleaban una sintaxis algo distinta a la que hemos adoptado en este proyecto. Curiosamente, ambos enfoques fueron propuestos por Christopher Harte (en 2005 y 2010, respectivamente), pero en el primero los acordes mayores, por ejemplo, se indican con `root:quality` (A:maj, E:maj…), mientras que en el segundo se indican únicamente como `root` (A, E…).

Además, como ya comentábamos en el descripción de la evaluación, tuvimos que decidir cómo adaptar y cómo evaluar nuestro vocabulario armónico en este contexto en que había muchos acordes "desconocidos", que sin embargo podían guardar cierta relación (tener la misma tónica, por ejemplo). Las numerosas inversiones y las tensiones tuvimos que obviarlas, teniendo en mente que, si el sistema era capaz de detectar una tónica pese a usarse en un acorde invertido, eso debía ser puntuado, pese a no estar "acertando" ese acorde complemente. No existe en nuestro vocabulario armónico ni estamos empleando herramientas lo suficientemente potentes, por lo que no podríamos acertarlo. Lo mismo ocurre con los acordes dominantes, indicados `root:7` o `root:9`, optamos por conservar la cualidad `:maj` que tiene de forma implícita y valorar la situación en la que algoritmo consigue estimar una triada mayor (pese a la presencia de séptima y novena).

Dedicamos las primeras líneas de la función que ejecuta el método de evaluación, por tanto, a la adaptación de todas estas características, para así poder obtener una valoración del desempeño del algoritmo que resultase, al menos, justa, pese a las diferencias de nomenclatura y asumiendo que las capacidades del método de platillas son ciertamente limitadas.

Capítulo 6

Conclusiones

La detección automática de acordes es una herramienta de enorme utilidad para la segmentación armónica de audio musical, y su ámbito de aplicación es enorme, desde la clasificación de música a numerosas y atractivas tareas con una comercialización potencial.

El uso del enfoque a partir de audio frente al tratamiento simbólico ha permitido que se realice la detección empleando una descripción del contenido musical mediante los perfiles de croma PCP, sin necesidad de disponer de una transcripción completa. La obtención de características polifónicas ha sido suficiente para determinar la identidad de los acordes de un archivo de audio de entrada en cada instante de tiempo. No perdemos de vista, no obstante, el límite natural de la técnica que ya han apuntado algunos autores, anunciando la necesidad de nuevos paradigmas para incrementar la precisión de la detección automática.

Con la propuesta actual se ha logrado alcanzar una modesta tasa de acierto en comparación con la de los sistemas punteros que actualmente definen el estado de la cuestión. El método de similitud utilizado, así como el paradigma de perfiles de croma patrón, tiene implícito un límite de eficacia, si bien mejorable con las técnicas de pre-procesado y refinamiento apropiadas. Sí hemos aportado un enfoque creativo, de carácter multimodal, a la hora de incorporar información de tempo y ritmo a la detección de acordes, empleando el recientemente acuñado tempograma para obtener una descripción adicional del contenido musical. Creemos que puede mejorarse, no obstante, el tratamiento y aprovechamiento de esta información.

También hemos explorado las opciones de secuenciación de los procesos que conforman el sistema y automatización o *batch processing* para extender este procesado a bases de datos completas alojadas en un disco local. Con un poco más de revisión de los recursos no costaría extender esto a bases de datos alojadas en servidores, pues Sonic Annotator está muy preparado para esas operaciones.

Como decíamos, la gran complejidad que presenta la extracción de información de la señal de audio hace necesario el uso de herramientas más sofisticadas, a menudo basadas en la inteligencia artificial y la teoría probabilística, para alcanzar resultados reseñables. Prevemos que ésta será la dirección en la que avance la técnica en el futuro próximo. La incipiente popularidad del aprendizaje profundo (*deep learning*), en el contexto del aprendizaje automático, no tardará en extenderse a muchas áreas de investigación conforme avancen las capacidades de procesamiento de *big data* y se optimicen las tareas de segmentación propias de la ciencia de la información. Varias arquitecturas de redes

neuronales profundas han sido ya aplicadas a tareas como la visión por computador, el reconocimiento del habla y el de audio y música (Boulanger-Lewandowski, et al., 2013; Bonvini, 2014), logrando resultados de vanguardia. El principio de análisis/re-síntesis aplicado a la imitación de las capacidades humanas ha dado y dará todavía grandes e increíbles soluciones a los problemas tecnológicos.

Entre los objetivos de este trabajo se encontraba el comprender el alcance actual de las tecnologías del sonido y de la música, por lo que se ha realizado un repaso exhaustivo del estado de la cuestión que, por otra parte, puede servir de punto de partida a futuros alumnos que deseen continuar desarrollando sistemas en esta dirección, en el contexto de las investigaciones del Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Alicante.

El principal objetivo de este trabajo era el diseño de un sistema de detección automática de acordes para apoyar la tarea de transcripción automática de música, desarrollada actualmente en el mencionado grupo de investigación. Hemos explorado esta dirección con el análisis de la base de datos de acordes de piano de la Universidad de Málaga, aunque queda mucho por hacer. El cumplimiento completo de ese principal objetivo deberá evaluarse en el transcurso de las investigaciones del departamento, en las que se tiene previsto adoptar un enfoque multimodal, combinando así diferentes fuentes de información para contribuir al conocimiento que el sistema posee del audio a transcribir. Esperemos que nuestra aportación sirva para dar soporte y enriquecer las capacidades de este transcriptor.

6.1. Líneas de trabajo futuro

Existen multitud de posibles direcciones de trabajo futuro que resultarían interesantes para continuar desarrollando este sistema, mejorar la detección del conjunto de acordes actual y ampliarlo para incluir inversiones y otros acordes comunes.

La similitud con acordes patrón, sin ir más lejos, podría enriquecerse empleando patrones basados en ejemplos reales. Para asemejar más los patrones a un caso real, es posible utilizar una serie de grabaciones, extraer sus cromagramas y, para cada acorde, obtener estadísticamente los patrones basados en los perfiles de croma analizados (por ejemplo, promediando los datos). De esta manera los patrones tendrían una base totalmente realista.

Partiendo de la propuesta anterior, puede adoptarse un enfoque orientado al reconocimiento de patrones. Se podrían almacenar las grabaciones y crear un sistema basado en el algoritmo de los *k*-vecinos más cercanos (*k-Nearest Neighbors*, kNN). En líneas generales, dado un patrón a clasificar (como en nuestro caso un instante de cromagrama o vector de croma), el algoritmo mide la distancia de este elemento a todos los patrones almacenados. Una vez hecho esto, busca los *k* elementos que

han dado como resultado la menor distancia y comprueba qué clase representan, es decir, qué acorde. El acorde asignado al nuevo patrón será el que la mayoría de los *k* elementos determinen. El núcleo del sistema no diferiría mucho del presentado en el actual proyecto, pero la función de similitud sería el algoritmo kNN. No obstante, considerar el problema desde este punto de vista da pie a probar algunas experiencias interesantes.

Una de ellas sería utilizar algoritmos de selección de prototipos. Como estos sistemas tienen una gran cantidad de ejemplos sobre los que trabajar, a menudo se confunden. Un algoritmo de selección de prototipos escogería qué patrones de acorde serían los más interesantes para realizar la estimación, de entre todos los posibles candidatos.

Otra experiencia a realizar sería, considerando que el problema de estimar un acorde es clasificar un patrón como perteneciente a una determinada clase, subdividir el problema en varias clases que no tienen realmente relación entre sí. Por ejemplo, si limitamos el vocabulario armónico a triadas mayores y menores, el sistema tiene que lidiar con 24 tipos de acorde (12 mayores y 12 menores). Desde una perspectiva de un único clasificador, cada elemento del cromagrama podría ser C:maj, C:min, D:maj, D:min, …, B:maj, B:min**.** Sin embargo, esto puede subdividirse en dos partes: la estimación de una raíz o tónica (que sería C, D, E, F, …, B; un total de 12 clases) y un modo (simplemente mayor o menor; lo que hace un total de 2 clases). Esta consideración se puede llevar a cabo o mediante diferentes configuraciones. Por ejemplo:

1. Una configuración en paralelo en la que las decisiones de cada clasificador no afecten al otro.
2. Una configuración en serie en la que la decisión tomada por el primero invalide ciertas opciones del segundo.

Todo este estudio, las diferentes configuraciones unidas a la selección de prototipos, podría ser una ampliación bastante interesante.

Por otra parte, la detección de las inversiones de los acordes, que en la literatura se entiende como un nivel superior de complejidad, podría abordarse en base a la propuesta de Goto (1999, 2000), obteniendo una caracterización adecuada de los elementos melódicos, armónicos y del bajo. Es común utilizar un cromagrama adicional exclusivamente para detectar la voz del bajo (*bass chromagram*) (McVicar, et al., 2014). La Universidad Queen Mary de Londres, con Matthias Mauch a la cabeza, ha desarrollado un paquete completo de Vamp plug-ins basado en el algoritmo NNLS que incluye un *bass chromagram* (Mauch y Dixon, 2010b).

Entendemos que la umbralización (*thresholding*) es un método interesante para diferenciar una situación polifónica de una monofónica, pero ésta debería ser dinámica y no estática. El umbral que

se establezca debería venir dado por alguna característica dependiente de la grabación analizada para garantizar cierta fiabilidad, como la energía de la señal, y no limitarse a fijar un valor estimado heurísticamente. Asimismo, la caracterización del silencio podría hacerse de manera independiente en base a la energía o bien empleando descriptores espectrales para la ruidosidad y la armonicidad.

Como técnica de pre-procesado alternativa para paliar el efecto de los sobretonos podría aplicarse sobre el espectro de la señal un filtro gaussiano centrado en la nota C_4, previamente a la obtención del cromagrama, tal y como señalan Cho y Bello (2014). De esta forma, las componentes de alta y baja frecuencia se verían reducidas, las primeras compuestas eminentemente por sobretonos (o, al menos, ésta es la hipótesis planteada) y las segundas provocadas por sonidos no deseados procedentes del bombo de la batería y del ruido debido a la baja resolución frecuencial.

Por último, podría adoptarse un enfoque probabilístico para estabilizar la decisión del acorde, que no se produzca basculación en el estado de la detección si no se cumplen ciertos criterios. Estimar un acorde puede entenderse como hallar la probabilidad de que en un instante de tiempo t se dé un patrón c_i dado:

$$P(t \mid c_i) = corr(\boldsymbol{x}(t),\ c_i)$$

El acorde estimado $\hat{c}(t)$ sería aquel que maximice el resultado de la ecuación anterior, esto es, la probabilidad de encontrar un patrón en el vector de croma asociado al instante t:

$$\hat{c}(t) = argmax_i\big(P(t \mid c_i)\big)$$

El problema de la inestabilidad habría que estudiarlo en instantes de tiempo posteriores como la probabilidad de que el acorde en $t + 1$ siga siendo $c_i(t)$:

$$\hat{c}(t+1) = argmax_i\big(P(t+1 \mid c_i)\big)$$

Deberá existir una función de inercia para indicar la baja probabilidad de que el acorde $\hat{c}(t+1)$ sea distinto del acorde $\hat{c}(t)$, en la que esta probabilidad vaya disminuyendo con el tiempo (en base al conocimiento previo que tenemos del ritmo armónico). Esta inercia tendría aspecto de exponencial decreciente $e^{-\alpha t}$ y estaría caracterizada por un coeficiente de amortiguamiento α, que deberá depender de la información de tempo (obtenida, por ejemplo, con un tempograma). La exponencial tomaría parte en el cálculo de la correlación entre los acordes patrón y el vector de croma, ponderando el resultado como $1 - e^{-\alpha t}$. Así, la probabilidad de que se detecte un acorde de la colección c_j distinto a c_i iría incrementándose con el tiempo, conforme se extinga el factor exponencial, hasta que finalmente el estado de la decisión bascule.

Bibliografía

Anderson, E. J., 1997. *Limitations of Short-Time Fourier Transforms in Polyphonic Pitch Recognition. Technical Report Ph.D.* Washington: Department of Computer Science and Engineering, University of Washington. http://www.cs.washington.edu/homes/eric/Publications.html.

Barbancho, A. M., Barbancho, I., Tardón, L. J. & Molina, E., 2013. *Database of Piano Chords. An Engineering View of Harmony.* Málaga: Springer. SpringerBriefs in Electrical and Computer Engineering.

Bello, J. P., n.d. *Chroma and tonality.* New York: MPATE-GE 2623 Music Information Retrieval. New York University.

Benetos, E. et al., 2012. Automatic Music Transcription: Breaking the Glass Ceiling. *ISMIR 13th International Society for Music Information Retrieval Conference.*

Benetos, E. y otros, 2013. Automatic Music Transcription: Challenges and Future Directions. *Journal of Intelligent Information Systems,* Vol. 41 (No. 3), pp. 407-434.

Blankertz, B., s.f. *The Constant Q Transform. Drafts on Digital Signal Processing, Universität Münster.* [En línea]
Available at: http://wwwmath.uni-muenster.de/logik/Personen/blankertz/constQ/constQ.html

Bonvini, A., 2014. *Automatic Chord Recognition Using Deep Learning Techniques.* Milán: Politecnico di Milano, Facoltà di Ingegneria de ll'Informazione.

Boulanger-Lewandowski, N., Bengio, Y. & Vicent, P., 2013. Audio Chord Recognition with Recurrent Neural Networks. *International Society for Music Information Retrieval.*

Bregman, J., 1998. Psychological Data and Computational Auditory Scene Analysis. In: D. Rosenthal & H. G. Okuno, eds. *Computational Auditory Scene Analysis.* s.l.:Lawrence Erlbaum Associates, Inc..

Brown, J. C., 1991. Calculation of a Constant-Q Spectral Transform. *Journal of the Acoustical Society of America,* Vol. 89(No. 1), pp. 425-434.

Brown, J. C. & Puckette, M. S., 1992. An Efficient Algorithm for the Calculation of a Constant Q Transform. *Journal of Acoustical Society of America,* Vol. 92 (No. 5), pp. 2698-2701.

Burgoyne, J. A., 2012. *Stochastic Processes and Database-Driven Musicology, Ph.D. diss..* Montréal, Québec, Canada: McGill University.

Burgoyne, J., Pugin, L., Kereliuk, C. & Fujinaga, I., 2007. A Cross-Validated Study of Modelling Strategies for Automatic Chord Recognition in Audio. *Proceedings 8th International Conference on Music Information Retrieval,* p. p.251.

Cano, P., 1998. Fundamental Frequency Estimation in the SMS Analysis. *COSTG6 Conference on Digital Audio Effects (DAFX).*

Catteau, B., Martens, J.-P. & Leman, M., 2007. A Probabilistic Framework for Audio-Based Tonal Key and Chord Recognition. *Proceedings of the 30th Annual Conference of the Gesellschaft für Klassifikation,* p. 637–644.

Cemgil, A. T., 2004. *Bayesian Music Transcription. Doctoral diss..* Nijmegen: Radboud University of Nijmegen.

Cho, T. & Bello, J. P., 2014. On the Relative Importance of Individual Components of Chord Recognition Systems. *IEEE/ACM Transactions on Audio, Speech, and Language Processing,* Vol. 22 (No. 2).

Chuan, C. & Chew, E., 2005. Polyphonic Audio Key Finding Using the Spiral Array Ceg Algorithm. *IEEE International Conference on Multimedia and Expo, Amsterdam.*

Davy, M. & Godsill, S., 2003. Bayesian Harmonic Models for Musical Signal Analysis. *Cambridge Music Processing Colloquium, Cambridge, UK.*

de Cheveigné, A., 2006. Chapter 1: Multiple f0 estimation. In: D. a. B. G. Wang, ed. *Computational Auditory Scene Analysis.* Wang, D. and Brown, G.J., editors ed. s.l.:John Wiley and sons.

de Cheveigné, A. & Kawahara, H., 2002. Yin, a Fundamental Frequency Estimator for Speech and Music. *Journal of the Acoustical Society of America,* Issue No. 111, pp. 1917-1930.

Doval, B. & Rodet, X., 1991. Fundamental Frequency Estimation Using a New Harmonic Matching Method. *International Computer Music Conference (ICMA, editor),* pp. 555-558.

Everest, F. A., 2001. *The Master Handbook of Acoustics.* 4th ed. s.l.:McGraw-Hill.

Fitzgerald, D., 2010. *Harmonic/Percussive Separation Using Median Filtering.* s.l.:Dublin Institute of Technology.

Fujishima, T., 1999. Realtime Chord Recognition of Musical Sound: A System Using Comon Lisp Music. *ICMA International Computer Music Conference,* pp. 464-467.

Fujishima, T., 2000. *Apparatus and Method for Recognizing Musical Chords.* U.S., Patent No. 6,057,502.

Godsmark, D. & Brown, G. J., 1999. A Blackboard Architecture for Computational Auditory Scene Analysis. *Speech Communication,* Issue No. 27, pp. 351-366.

Gold, B. & Rabiner, L., 1969. Parallel Processing Techniques For Stimating Pitch Periods of Speech in the Time Domain. *Journal of the Acoustical Society of America,* Issue No. 46, pp. 442-448.

Gómez, E., 2001. *Fundamental Frequency Study Report,* Barcelona: Music Technology Group, Universitat Pompeu Fabra.

Gómez, E., 2002. *Melodic Description of Audio Signals for Music Content Processing. Technical Report, PhD Research Work,* Universitat Pompeu Fabra: Music Technology Group.

Gómez, E., 2004. Tonal Description of Polyphonic Audio for Music Content Processing. *Audio Engineering Society Conference on Metadata.*

Gómez, E., 2006a. *Tonal Description of Music Audio Signals, Ph.D. diss..* Barcelona, España: Universitat Pompeu Fabra.

Gómez, E., 2006b. Tonal Description of Polyphonic Audio for Music Content Processing. *Journal on Computing, Special Cluster on Computation in Music,* Vol. 18(No. 3).

Gómez, E. & Herrera, P., 2004. Estimating the Tonality of Polyphonic Audio Files: Cognitive Versus Machine Learning Modelling Strategies. *International Conference on Music Information Retrieval.*

Gómez, E. et al., 2006. *A quantitative Comparison of Different Approaches for Melody Extraction from Polyphonic Audio Recordings. Technical report, MTG-TR-2006-01,* Barcelona: Music Technology Group, Universitat Pompeu Fabra.

Goto, M., 1999. A Real-Time Music Scene Description System: Detecting Melody and Bass Lines in Audio Signals. *IJCAI Workshop on Computational Auditory Scene Analysis,* pp. 31-40. http://www.etl.go.jp/ goto/PROJ/f0.html.

Goto, M., 2000. A Robust Predominant-f0 Estimation Method for Real-Time Detection of Melody and Bass Lines in CD Recordings. *IEEE International Conference on Acoustics Speech and Signal Processing,* pp. 757-760. http://www.etl.go.jp/ goto/PROJ/f0.html.

Goto, M. & Muraoka, Y., 1999. Real-Time Beat Tracking for Drumless Audio Signals: Chord Change Detection for Musical Decisions. *Speech Communication,* Issue No. 27, pp. 311-335.

Grosche, P. & Müller, M., 2011a. Extracting Predominant Local Pulse Information from Music Recordings. *IEEE Transactions on Audio, Speech and Language Processing,* Vol. 19(No. 6).

Grosche, P. & Müller, M., 2011b. Tempogram Toolbox: Matlab Implementation for Tempo and Pulse Analysis of Music Recordings. *International Society for Music Information Retrieval.*

Grosche, P., Müller, M. & Kurth, F., 2010. Cyclic Tempogram - A Mid-Level Tempo Representation for Music Signals. *IEEE ICASSP.*

Harte, C., 2010. *Towards Automatic Extraction of Harmony Information from Music Signals, PhD diss..* s.l.:Department of Electronic Engineering, Queen Mary, University of London.

Harte, C., Sandler, M., Abdallah, S. & Gómez, E., 2005. Symbolic Representation of Musical Chords: A Proposed Syntax for Text Annotations. *ISMIR Proceedings of the 6th International Society for Music Information Retrieval Conference,* pp. 66-71.

Helmholtz, H. L. F., 1954. *On the Sensations of Tone as a Physiological Basis for the Theory of Music.* 4th edition ed. New York, Dover: Trans., A.J. Ellis.

Herrera, E., 2015. *Teoría Musical y Armonía Moderna, Vol. I.* s.l.:Editado por Antoni Bosch.

Herrera, P., 2006. *Automatic Classification of Percussion Sounds: from Acoustics Features to Semantic Descriptions. Doctoral diss..* Barcelona: Universitat Pompeu Fabra.

Hess, W., 1983. *Pitch Determination of Speech Signals.* Springer-Verlag ed. Berlin, New York, Tokyo: Springer-Verlag, Springer Series in Information Sciences.

Iñesta Quereda, J. M., 2013. *Apuntes de síntesis digital de sonido.* Alicante: Departamento de Lenguajes y Sistemas Informáticos, Universidad de Alicante.

Izmirli, O., 2005. Template Based Key Finding from Audio. *International Computer Music Conference.*

Jehan, T., 1997. *Musical Signal Parameter Estimation, PhD thesis.* s.l.:CNMAT; IFSIC. http://www.cnmat.berkeley.edu/ tristan/Report/Report.html.

Kabamba Mbikayi, H., 2013. Toward Evolution Strategies Application in Automatic Polyphonic Music Transcription using Electronic Synthesis. *IJACSA International Journal of Advanced Computer Science and Applications,* Vol. 4 (No. 3).

Kashino, K., Kinoshita, T. & Tanaka, H., 1995. Organization of Hierarchical Perceptual Sounds: Music Scene Analysis with Autonomous Processing Modules and a Quantitative Information Integration Mechanism. *International Joint Conference on Artificial Intelligence, Montreal.*

Klapuri, A., 2000a. Multiple Fundamental Frequency Estimation by Harmonicity and Spectral Smoothness. *IEEE Transactions in Speech and Audio Processing,* Vol. 11(No. 6), pp. 804-816.

Klapuri, A., 2000b. Qualitative and Quantitive Aspects in the Design of Periodicity Estimation Algorithms. *European Signal Processing Conference.*

Klapuri, A., 2004. *Signal Processing Methods for Music Transcription, Doctoral diss..* s.l.:Tampere University of Technology.

Klapuri, A., 2006. Multiple Fundamental Frequency Estimation by Summing Harmonic Amplitudes. *ISMIR Proceedings of the 7th International Conference on Music Information Retrieval,* pp. 216-221.

Klapuri, A., Virtanen, T., Eronen, A. & Seppänen, J., 2001. Automatic Transcription of Musical Recordings. *Consistent & Reliable Acoustic Cues Workshop.*

Kostek, A., 2004. Automatic Music Transcription As We Know it Today. *Journal of New Music Research,* Vol. 33 (No. 3), pp. 269-282.

Lahat, A., Niederjohn, R. J. & Krubsack, D. A., 1987. A Spectral Autocorrelation Method for Measurement of the Fundamental Frequency of Noise-Corrupted Speech. *IEEE Transactions on Acoustics, Speech and Signal Processing,* Vol. 35(No. 6), pp. 741-750.

Laroche, J., 1995. *Traitement des signaux audio-fréquences,* s.l.: Ecole National Supérieure de Télécommunications.

Lawson, C. L. & Hanson, R. J., 1974. Chapter 23: Linear Least Squares with Linear Inequality Constraints. In: *Solving Least Squares Problems.* s.l.:Prentice-Hall.

Leman, M., 1991. *Een model van toonsemantiek: naar een theorie en discipline van de muzikale verbeelding. Doctoral diss..* s.l.:University of Ghent.

Leman, M., 1995. Music and Schema Theory: Cognitive Foundations of Systematic Musicology. *Information Science. s.l.Springer-Verlag, Berlin-Heidelberg,* Issue No. 31.

Leman, M., 2000. An Auditory Model of the Role of Short Term Memory in Probe-Tone Ratings. *Music Perception, Special Issue in Tonality Induction,* Vol. 17 (No. 4), pp. 481-509.

Leman, M., Lesaffre, M., Baets, B. D. & Martens, J. P., 2004. Methodological Considerations Concerning Manual Annotation of Musical Audio in Function of Algorithm Development. *International Conference on Music Information Retrieval, Barcelona.*

Lester, J., 1989. *Enfoques analíticos de la música del siglo XX.* Ediciones Akal 2005 ed. Madrid: W.W. Norton and Company, Inc..

Levine, M., 2003. *The Jazz Piano Book.* s.l.:Sher Music Co..

Maher, R. C. & Beauchamp, J. W., 1993. Fundamental Frequency Estimation of Musical Signals Using a Two-Way Mismatch Procedure. *Journal of Acoustical Society of America,* Issue No. 95, pp. 2254-2263.

Martens, e. a., 2002. A Tonality-Oriented Symbolic Representation of Musical Audio Generated by Classification Trees. *EURO-FUSE Workshop on Information Systems,* pp. 49-54.

Martens, G. M. H. D., Baets, B. D. & Leman, M., 2004. Distance-Based Versus Tree-Based Key Recognition in Musical Audio. *Soft Computing (online).*

Martin, K. D., 1996. Automatic Transcription of Simple Polyphonic Music: Robust Front End Processing. *Third Joint Meeting of the Acoustical Societies of America and Japan.*

Mauch, M., 2010. *Automatic Chord Transcription from Audio Using Computational Models of Musical Context, Ph.D. diss..* London: Queen Mary University of London.

Mauch, M. & Dixon, S., 2008. A Discrete Mixture Model for Chord Labelling. *9th International Conference on Music Information Retrieval, Philadelphia, Pennsylvania,* pp. 45-50.

Mauch, M. & Dixon, S., 2010a. *Approximate Note Transcription for the Improved Identification of Difficult Chords.* London: Queen Mary University of London, Centre for Digital Music.

Mauch, M. & Dixon, S., 2010b. Simultaneous Estimation of Chords and Musical Context from Audio. *IEEE Transactions on ASSP,* Vol. 18(No. 6), pp. 1280-1289.

Mauch, M., Noland, K. C. & Dixon, S., 2009. Using Musical Structure to Enhance Automatic Chord Transcription. *ISMIR Proceedings of the 10th International Conference on Music Information Retrieval,* pp. 231-236.

McVicar, M., Santos-Rodríguez, R., Ni, Y. & De Bie, T., 2014. Automatic Chord Estimation from Audio: A Review of the State of the Art. *IEEE/ACM Transactions on Audio, Speech, and Language Processing,* Vol. 22 (No. 2).

Medan, J., Yair, E. & Chazan, D., 1991. Super Resolution Pitch Determination of Speech Signals. *IEEE Transactions on Signal Processing,* Vol. 39 (No. 1).

Meddis, R. & Hewitt, M. J., 1991. Virtual Pitch and Phase Sensitivity of a Computer Model of the Auditory Periphery: Pitch Identification. *JASA Journal of the Acoustical Society of America,* Vol. 89 (No. 6), pp. 2866-2882.

Nettles, B., 1987. *Harmony 1.* Boston: Berklee College of Music.

Noll, A. M., 1967. Cepstrum Pitch Determination. *Journal of the Acoustical Society of America,* Issue No. 41, pp. 293-309.

Ono, N. et al., 2008. *Separation of a Monaural Audio Signal into Harmonic/Percussive Components by Complementary Diffusion on Spectrogram.* s.l.:University of Tokyo.

Orio, N., 2006. Music Retrieval: A Tutorial and Review. *Foundations and Trends in Information Retrieval,* Vol. 1 (No. 1).

Pauws, S., 2004. Musical Key Extraction from Audio. *International Conference on Music Information Retrieval, Barcelona.*

Peeters, G., 2006. Chroma-Based Estimation of Musical Key from Audio-Signals Analysis. *ISMIR Proceedings of the 7th International Music Information Retrieval Conference.*

Piston, W., 1987. *Armonía.* SpanPress© Universitaria 1998 ed. Cooper City (Florida): W.W. Norton & Company, Inc., New York and London.

Piszczalski, M. & Galler, B. A., n.d. Predicting Musical Pitch from Component Frequency Ratios. *Journal of the Acoustical Society of America,* Issue No. 66, pp. 710-720.

Purwins, H., Blankertz, B. & Obermayer, K., 2000. A New Method for Tracking Modulations in Tonal Music in Audio Data Format. *Neural Networks - IJCNN, IEEE Computer Society,* Issue No. 6, pp. 270-275.

Rabiner, L. R., Sambur, M. R. & Schmidt, C. E., 1975. Applications of a Nonlinear Smoothing Algorithm to Speech Processing. *IEEE Transactions on ASSP,* Vol. 23 (No. 6).

Rabiner, L. R. & Schafer, R. W., 1978. *Digital Signal Processing of Speech Signals.* s.l.:Prentice-Hall.

Roads, C., 1996. Pitch and Rhythm Recognition in MIDI Systems. In: *The Computer Music Tutorial.* Massachusetts: The MIT Press, pp. 503-531.

Roads, C., 2001. *Microsound.* Cambridge, Massachusetts. London, England: MIT Press.

Romero, J. & Cerdá, S., 1997. Uso del análisis multirresolución para calcular el pitch de señales en presencia de ruido. *Revista de Acústica (SEA), 28. http://www.ia.csic.es/Sea/.*

Rossing, T. D., Moore, F. R. & Wheeler, P. A., 2002. *The Science of Sound.* 3rd edition ed. s.l.:Addison-Wesley (Pearson PLC).

Serra, X., 1996. Musical Sound Modeling with Sinusoids Plus Noise. In: G. D. Poli, A. Picialli, S. T. Pope & C. Roads, eds. *Musical Signal Processing.* s.l.:Swets & Zeitlinger.

Sheh, A. & Ellis, D., 2003. Chord Segmentation and Recognition Using em-Trained Hidden Markov Models. *Proceedings 4th International Society of Music Information Retrieval ,* pp. 183-189.

Shepard, R., 1964. Circularity in judments of relative pitch. *Journal of Acoustical Society of America,* Volumen Vol. 36, p. 2346.

Shepard, R., 2001. Chapter 13, Pitch Perception and Measurement; Chapter 15, Tonal Structure and Scales. In: *Music, Cognition and Computerized Sound.* s.l.:The MIT Press.

Talkin, D., 1995. Robust Algorithm for Pitch Tracking. In: W. B. Kleijn & K. K. Paliwal, eds. *Speech Coding and Synthesis.* Kleijin ed. s.l.:Elsevier Science B. V..

Terhardt, E., 1979. Calculating Virtual Pitch. *Hearing Research,* Issue No. 1, pp. 155-182.

Terhardt, E., Stoll, G. & Seewann, M., 1981. Algorithm for Extraction of Pitch and Pitch Salience From Complex Tonal Signals. *Journal of the Acoustical Society of America,* Issue No. 71, pp. 679-688.

Tzanetakis, G., 2002. Pitch Histograms in Audio and Symbolic Music Information Retrieval. *3rd international Symposium on Music Information Retrieval, Paris.*

Valero-Castells, A., 2012. *Apuntes de composición musical.* Valencia: Departamento de Composición, Conservatorio Superior de Música de Valencia.

Valero, J. J., 2012. *Harmonic/Percussive Separation Using Median Filtering.* s.l.:Cork Institute of Technology.

Wakefield, G., 1999. Mathematical Representation of Joint Time-Chroma Distributions. *Proc. Int. Symp. Opt. Sci. Eng. Instrum.,* Volume Vol. 99, pp. 18-23.

Walmsley, P. J., Godsill, S. J. & Rayner, P. J. W., 1999. Bayesian Graphical Models for Polyphonic Pitch Tracking. *Diderut Forum, Vienna.*

Yoshioka, T. et al., 2004. Automatic Chord Transcription with Concurrent Recognition of Chord Symbols and Boundaries. *Proceedings 5th Internacional Conference on Music Information Retrieval.*

Zhu, Y., Kankanhalli, M. S. & Gao, S., 2005. Music Key Detection for Musical Audio. *11th International Multimedia Modelling Conference (MMM'05).*

Anexo A. Código fuente del proyecto

main.m[37]

```
%=========================================================================
%
%% ESTIMACIÓN AUTOMÁTICA DE ACORDES PARA USO EN TRANSCRIPCIÓN MUSICAL A
%%
%%                            PARTIR DE AUDIO
%%
%                           Miguel Segura Sogorb
%
%                           Trabajo Fin de Grado
%
%=========================================================================
%
clear all


    filename = 'UMAPiano-DB-C3E3G3-NO-F.wav';

    %Control of individual components
    control = struct('preProcess',0,'chromagramComputation', 0,...
        'idealTemplates',0,'PLPComputation',0, 'postProcess',1);
    %-------------------------------------------------------------------
-
    %Source separation (Fitzgerald's harmonic/percussive separation
    %algorithm) previously done or not
    control.preProcess = 0; %0 == No pre-processing step
                    %1 == Usage of Harmonic component as a reference for
                    %computing PLP curve (already computed)
                    %2 == Usage of Percussive component as a reference
for
                    %computing PLP curve (already computed)
                    %4 == Files have not been pre-processed previously.
                    %The step will be performed now. Reference for PLP->
                    %Harmonic component
                    %5 == Files have not been pre-processed previously.
                    %The step will be performed now. Reference for PLP->
                    %Percussive component
    %-------------------------------------------------------------------
-
    %Chromagram computation previously done or not
    control.chromagramComputation = 0; %0 == Already done
                                       %1 == Not previously done;
retrieve
                                       %chroma features now
    %-------------------------------------------------------------------
-
    %Select whether chord templates are ideal or non ideal
    control.idealTemplates = 0; %0 == Patterns based on harmonic series
                                %1 == Ideal patterns
    %-------------------------------------------------------------------
-
```

[37] La pretensión era que todo el código estuviese comentado en inglés, pero el tiempo no nos ha permitido traducir todos los comentarios que ya estaban en castellano, pedimos disculpas por el bilingüismo forzado.

```
    control.PLPComputation = 0; %0 == Bypass PLP curve computation step
                                %1 == Compute PLP curve
    %-------------------------------------------------------------------
-
    control.postProcess = 0;    %0 == Bypass median filter
                                %1 == Apply median filter
    %-------------------------------------------------------------------
-

    if(control.preProcess >= 4)
        disp('Harmonic/Percussive separation...')
        filename = HPseparationFunction(filename);
        %Filename is updated in order to extract chroma features from
        %the harmonic component for improved accuracy
        control.chromagramComputation = 1;
    end

    if(control.chromagramComputation == 1)
        disp('Chromagram computing...')
        computeChromagram(filename);
    end

    %Carga y normalización del cromagrama
    disp('File load...');
    [filename] = checkFileName(filename,control);
    [X] = loadFiles(filename);

    %Generación de patrones (PCP)
    disp('Template generation...');
    [P] = getTemplates(control);

    %Similitud de los patrones con los vectores de croma analizados
    disp('Similarity...');
    [D] = patternMatching (X, P);

    %Obtención de la curva PLP con información acerca de los onsets
    if(control.PLPComputation == 1)
        try
            disp('PLP curve computing...');
            [PLPcurve] = function_compute_PLPcurve (filename, control);
        catch me
            disp('[Error: audio file is too short for computing PLP
curve]')
            PLPcurve = 0;
        end
    else
        PLPcurve = 0;
    end

    %Determinación de acordes más probables en cada frame y sus etiquetas
    disp('Decision...');
    [tags,delta] = decision(D, PLPcurve,control);

    %Suavizado de la estimación mediante un filtro de mediana
    if(delta > 1)
        disp('Post-processing...');
        tags = medianFilter(tags,delta);
    else
       disp('[Delta = 1; median filter ignored]')
    end
```

```
    %Volcado de los resultados a un fichero .csv
    disp('Results...');
    saveResults(X,tags,filename);
    GTChords = metrics(filename);
```

HPseparationFunction.m[38]

```
function [newfilename_H] = HPseparationFunction(filename)
%This function performs the harmonic/percussive separation of the audio
%file

%Check for sharp character because Sonic Annotator does not understand it
%(and after this step chroma features will be computed using SA)
filename = strrep(filename,'#','s');

    % Author: Derry FitzGerald
    %
    %-----------------------------------------------------------------------
------
    % This source code is provided without any warranties as published in
    % DAFX book 2nd edition, copyright Wiley & Sons 2011, available at
    % http://www.dafx.de. It may be used for educational purposes and not
    % for commercial applications without further permission.
    %-----------------------------------------------------------------------
------

    %----- user data -----
    WLen            =4096;
    hopsize         =2048;
    lh              =17; % length of the harmonic median filter
    lp              =17; % length of the percussive median filter
    p               =2;
    w1              =hanning(WLen,'periodic');
    w2              =w1;
    hlh             =floor(lh/2)+1;
    th              =2500;
    folder_name = ['Data_wav\' filename];
    [DAFx_in_st, FS]=wavread(folder_name);

    DAFx_in         =(DAFx_in_st(:,1)/2) + (DAFx_in_st(:,2)/2);%mono

    L               = length(DAFx_in);
    DAFx_in         = [zeros(WLen, 1); DAFx_in; ...
       zeros(WLen-mod(L,hopsize),1)] / max(abs(DAFx_in));
    DAFx_out1 = zeros(length(DAFx_in),1);
    DAFx_out2 = zeros(length(DAFx_in),1);

    %----- initialisations -----
    grain           = zeros(WLen,1);
    buffer          = zeros(WLen,lh);
    buffercomplex   = zeros(WLen,lh);
    oldperframe     = zeros(WLen,1);
    onall         = [];
    onperc        = [];
```

[38] Función de pre-procesado creada por Derry Fitzgerald (2010).

```
    tic
    %UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
    pin  = 0;
    pout = 0;
    pend = length(DAFx_in)-WLen;

    while pin<pend
        grain = DAFx_in(pin+1:pin+WLen).* w1;
    %=========================================
        fc  = fft(fftshift(grain));
        fa  = abs(fc);
        % remove oldest frame from buffers and add
        % current frame to buffers
        buffercomplex(:,1:lh-1)=buffercomplex(:,2:end);
        buffercomplex(:,lh)=fc;
        buffer(:,1:lh-1)=buffer(:,2:end);
        buffer(:,lh)=fa;
        % do median filtering within frame to suppress harmonic
instruments
        Per = medfilt1(buffer(:,hlh),lp);
        % do median filtering on buffer to suppress percussion
instruments
        Har = median(buffer,2);
        % use these Percussion and Harmonic enhanced frames to generate
masks
        maskHar = (Har.^p)./(Har.^p + Per.^p);
        maskPer = (Per.^p)./(Har.^p + Per.^p);
        % apply masks to middle frame in buffer
        % Note: this is the "current" frame from the point of view of the
median
        % filtering
        curframe=buffercomplex(:,hlh);
        perframe=curframe.*maskPer;
        harframe=curframe.*maskHar;
        grain1 = fftshift(real(ifft(perframe))).*w2;
        grain2 = fftshift(real(ifft(harframe))).*w2;
        % onset detection functions
        % difference of frames
        dall=buffer(:,hlh)-buffer(:,hlh-1);
        dperc=abs(perframe)-oldperframe;
        oall=sum(dall>0);
        operc=sum(dperc>0);
        onall = [onall oall];
        onperc = [onperc operc];
        oldperframe=abs(perframe);
    %=========================================
        DAFx_out1(pout+1:pout+WLen) = ...
           DAFx_out1(pout+1:pout+WLen) + grain1;
       DAFx_out2(pout+1:pout+WLen) = ...
           DAFx_out2(pout+1:pout+WLen) + grain2;
        pin  = pin + hopsize;
        pout = pout + hopsize;
    end
    %UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
    toc
    % process onset detection function to get beats
    [or,oc]=size(onall);
    omin=min(onall);
    % get peaks
    v1 = (onall > [omin, onall(1:(oc-1))]);
```

```
    % allow for greater-than-or-equal
    v2 = (onall >= [onall(2:oc), omin]);
    % simple Beat tracking function
    omax = onall .* (onall > th).* v1 .* v2;
    % now do the same for the percussion onset detection function
    % process onset detection function to get beats
    [opr,opc]=size(onperc);
    opmin=min(onperc);
    % get peaks
    p1 = (onperc > [opmin, onperc(1:(opc-1))]);
    % allow for greater-than-or-equal
    p2 = (onperc >= [onperc(2:opc), opmin]);
    % simple Beat tracking function
    opmax = onperc .* (onperc > th).* p1 .* p2;
    %----- listening and saving the output -----
    DAFx_out1 = DAFx_out1((WLen + hopsize*(hlh-1)) ...
            :length(DAFx_out1))/max(abs(DAFx_out1));
    DAFx_out2 = DAFx_out2(WLen + (hopsize*(hlh-1)) ...
            :length(DAFx_out2))/max(abs(DAFx_out2));
    % soundsc(DAFx_out1, FS);
    % soundsc(DAFx_out2, FS);

    newfilename_H = strcat('Harmonic_',filename);
    newfilename_P = strcat('Percussive_',filename);
    folder_name_H = strcat('Data_wav\',newfilename_H);
    folder_name_P = strcat('Data_wav\',newfilename_P);
    wavwrite(DAFx_out1, FS, folder_name_P);
    wavwrite(DAFx_out2, FS, folder_name_H);


end
```

computeChromagram.m

```
function [] = computeChromagram(filename)

    %filename = strrep(filename,'.wav','');
    folder_name = ['Data_wav\' filename];
    filename
    listing = dir(folder_name);
    for i=1:length(listing)
        %command line orders
        order = ['C:\sonic-annotator-0.7-win32\sonic-annotator.exe -t
conf.n3 Data_wav\'...
            filename ' -w csv --csv-force --csv-basedir Chromagram'];
        system(order) %call cmd

    end
end
```

checkFileName.m

```
function [filename] = checkFileName(filename, control)
```

```
%CHECKFILENAME checks wheter the pre-processing step have been performed
    %(Fitzgerald's source separation algorithm)
%Additionally, shows a error message if a file does not exist

%Check for sharp character because Sonic Annotator does not understand it
%(wav files were renamed to prevent error, but new files may enter)
filename = strrep(filename,'#','s');

    if(control.preProcess == 1 || control.preProcess == 2)
            newHead = 'Harmonic_';
            filename = strcat(newHead,filename);
            %This new file name refers to the audio files pre-processed
            %with harmonic-percussion separation feature
    end
end
```

loadFiles.m

```
function [X] = loadFiles (filename)

    k = strfind(filename, '.wav');
    chromagramName = strcat('Chromagram/',filename(1:k-1), ...
        '_vamp_nnls-chroma_nnls-chroma_chroma.csv');

    X = load(chromagramName); % Carga del cromagrama (.csv)
    X (:,2:13) = X(:,2:13)/(max(max(X(:,2:13)))); % Normalizacion
    %de los vectores de croma

    %Recordemos la estructura del fichero:
    %   13 columnas (eje temporal + 12 clases cromaticas)
    %   M filas (numero de muestras temporales)

end
```

getTemplates.m

```
function [P] = getTemplates(control)

%PARÁMETROS DE ENTRADA: -

%PARÁMETROS DE SALIDA:
    %P:  matriz de 5x12, que contiene los PCP patrón (12 clases de
altura)
        %de 5 tipos de acorde

    P = zeros(6,12); %inicialización de la matriz de patrones

    if(control.idealTemplates == 1)
        %%%%%%%PATRONES DE ACORDE IDEALES%%%%%%%
```

```
        %TRIADA MAYOR
        P(1,:) = [1 0 0 0 1 0 0 1 0 0 0 0];
        %TRIADA MENOR
        P(2,:) = [1 0 0 1 0 0 0 1 0 0 0 0];
        %TRIADA DISMINUIDA
        P(3,:) = [1 0 0 1 0 0 1 0 0 0 0 0];
        %TRIADA AUMENTADA
        P(4,:) = [1 0 0 0 1 0 0 0 1 0 0 0];
        %TRIADA DE CUARTA SUSPENDIDA
        P(5,:) = [1 0 0 0 0 1 0 1 0 0 0 0];
        %SITUACIÓN DE SILENCIO (NO ACORDE)
        P(6,:) = [0 0 0 0 0 0 0 0 0 0 0 0];

    else
        %%%%%%%PATRONES DE ACORDE NO IDEALES%%%%%%%
        %Se han considerado 11 armónicos con amplitud decreciente para
        %cada nota, partiendo de la serie armónica y considerando
afinación
        %justa
        %TRIADA MAYOR
        P(1,:) = [0.76 0.04 0.34 0.00 0.88 0.06 0.08 1.00 0.12 0.05 ...
            0.10 0.36];
        %TRIADA MENOR
        P(2,:) = [0.68 0.08 0.25 0.68 0.11 0.09 0.00 1.00 0.00 0.07 ...
            0.26 0.11];
        %TRIADA DISMINUIDA
        P(3,:) = [1.00 0.37 0.06 0.95 0.23 0.06 0.95 0.23 0.06 0.05 ...
            0.52 0.00];
        %TRIADA AUMENTADA
        P(4,:) = [1.00 0.00 0.16 0.27 1.00 0.00 0.16 0.27 1.00 0.00 ...
            0.16 0.27];
        %TRIADA DE CUARTA SUSPENDIDA
        P(5,:) = [1.00 0.00 0.09 0.33 0.14 0.86 0.11 0.86 0.32 0.14 ...
            0.12 0.04];
    end

end
```

patternMatching.m

```
function [D] = patternMatching (X, P)

%PARÁMETROS DE ENTRADA
    %X:       contiene el cromagrama
    %P:       contiene los patrones de acorde triada en diferentes filas

%PARÁMETROS DE SALIDA
    %D:       matriz de distancias euclideas (norma L2) entre los
vectores
             %de croma X y los patrones de acorde P


    [M,N] = size(X);
    %Dimensiones del cromagrama:
    %- M: numero de muestras temporales
    %- N: 1 columna de eje temporal, 12 columnas con las clases de altura

    d_maj = zeros(M,12); %matriz resultado de la norma L2 con mayores
```

```
    d_min = zeros(M,12); %matriz resultado de la norma L2 con menores
    d_dim = zeros(M,12); %matriz resultado de la norma L2 con disminuidos
    d_aug = zeros(M,12); %matriz resultado de la norma L2 con aumentados
    d_sus = zeros(M,12); %matriz resultado de la norma L2 con suspendidos
    d_null = zeros(M,12); %matriz resultado de la norma L2 con silencio

    D = zeros(M,12,5);

    %%%%%%%COMPARACIÓN DE PATRONES%%%%%%%
    %El vector de croma será el que se desplace para realizar la
    %correlación con el patrón del acorde; es equivalente a usar
    %12 patrones de acorde distintos, uno por cada tónica

    for i=1:M

        h = X(i, 2:N); %vector de croma i-esimo

        for k=1:12  %Se calcularán 12 distancias euclideas por cada
modelo
                    %de acorde

            %%%CÁLCULO DE DISTANCIAS EUCLIDEAS%%%
            %patrón mayor
            d_maj(i,k) = norm(h - P(1,:)); %distancia euclidea

            %patrón menor
            d_min(i,k) = norm(h - P(2,:));

            %patrón disminuido
            d_dim(i,k) = norm(h - P(3,:));

            %patrón aumentado
            d_aug(i,k) = norm(h - P(4,:));

            %patrón suspendido
            d_sus(i,k) = norm(h - P(5,:));

            %patrón suspendido
            d_null(i,k) = norm(h - P(6,:));


            %%%PROTECCIÓN FRENTE A INDETERMINACIONES%%%
            if(isnan(d_maj(i,k)) == 1)
                d_maj(i,k) = 10000; %+inf
            end

            if(isnan(d_min(i,k)) == 1)
                d_min(i,k) = 10000; %+inf
            end

            if(isnan(d_dim(i,k)) == 1)
                d_dim(i,k) = 10000; %+inf
            end

            if(isnan(d_aug(i,k)) == 1)
                d_aug(i,k) = 10000; %+inf
            end
```

```
            if(isnan(d_sus(i,k)) == 1)
                d_sus(i,k) = 10000; %+inf
            end

            if(isnan(d_sus(i,k)) == 1)
                d_null(i,k) = 10000; %+inf
            end

            %%%ROTACIÓN DEL VECTOR DE CROMA%%%
            aux = h(1);
            h(1:11) = h(2:12);
            h(12) = aux;

            %Rotación a la izquierda (rota una posición por cada
iteración
            %del bucle interior)

        end
    end

    D(:,:,1) = d_maj;
    D(:,:,2) = d_min;
    D(:,:,3) = d_dim;
    D(:,:,4) = d_aug;
    D(:,:,5) = d_sus;
    D(:,:,6) = d_null;

End
```

function compute_PLPcurve.m[39]

```
function [PLP] = function_compute_PLPcurve(filename, preProcess)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%
    % Name: test_compute_PLPcurve.m
    % Date of Revision: 2011-10
    % Programmer: Peter Grosche
    % http://www.mpi-inf.mpg.de/resources/MIR/tempogramtoolbox/

    %
    % Description: Illustrates the following steps:
    %   1. loads a wav file
    %   2. computes a novelty curve
    %   3. computes a fourier-based tempogram
    %   4. derives a PLP curve
    %   5. visualizes novelty and PLP curves
    %   6. sonifies novelty and PLP peaks
    %   7. computes a PLP curve of a restricted tempo range
    %
    % Audio recordings are obtained from: Saarland Music Data (SMD)
    % http://www.mpi-inf.mpg.de/resources/SMD/
    %
    % License:
    %     This file is part of 'Tempogram Toolbox'.
    %
```

[39] A continuación comienzan una serie de funciones pertenecientes al Toolbox Tempogram (Grosche y Müller, 2011b).

```
    %     'Tempogram Toolbox' is free software: you can redistribute it
and/or modify
    %     it under the terms of the GNU General Public License as
published by
    %     the Free Software Foundation, either version 2 of the License,
or
    %     (at your option) any later version.
    %
    %     'Tempogram Toolbox' is distributed in the hope that it will be
    %     useful,
    %     but WITHOUT ANY WARRANTY; without even the implied warranty of
    %     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    %     GNU General Public License for more details.
    %
    %     You should have received a copy of the GNU General Public
License
    %     along with 'Tempogram Toolbox'. If not, see
    %     <http://www.gnu.org/licenses/>.
    %
    %
    % Reference:
    %   Peter Grosche and Meinard Müller
    %   Extracting Predominant Local Pulse Information from Music
Recordings
    %   IEEE Transactions on Audio, Speech, and Language Processing,
19(6), 1688-1701, 2011.
    %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%

    dirWav = 'Data_wav/';

    if(control.preProcess == 2 || control.preProcess == 5)
        filename = strrep(filename,'Harmonic_','Percussive_');
    end

    %% 1., load wav file, automatically converted to Fs = 22050 and mono

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%

    [audio,sideinfo] = wav_to_audio('',dirWav,filename);
    Fs = sideinfo.wav.fs;



    %% 2., compute novelty curve

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%

    parameterNovelty = [];

    [noveltyCurve,featureRate] = audio_to_noveltyCurve(audio, Fs,
parameterNovelty);


    %% 3., compute fourier-based tempogram
```

```
    % important parameters:
    %   .tempoWindow: trade-off between time- and tempo resolution
    %               try chosing a long (e.g., 12 sec) and short (e.g., 3
sec)
    %               window
    %   .BPM: tempo range and resolution

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%

    parameterTempogram = [];
    parameterTempogram.featureRate = featureRate;
    parameterTempogram.tempoWindow = 6;         % window length in sec
    parameterTempogram.BPM = 30:1:600;          % tempo values

    [tempogram, T, BPM] =
noveltyCurve_to_tempogram_via_DFT(noveltyCurve,parameterTempogram);
%     visualize_tempogram(tempogram,T,BPM)
%     title('Tempogram (Fourier)')


    %% 4., compute PLP curve

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%

    parameterPLP = [];
    parameterPLP.featureRate = featureRate;
    parameterPLP.tempoWindow = parameterTempogram.tempoWindow;

    [PLP,featureRate] = tempogram_to_PLPcurve(tempogram, T, BPM,
parameterPLP);
    PLP = PLP(1:length(noveltyCurve));  % PLP curve will be longer (zero
padding)



    %% 5, visualize

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%

    parameterVis = [];
    parameterVis.featureRate = featureRate;

%     visualize_noveltyCurve(noveltyCurve,parameterVis)
%     title('Novelty curve')

%     visualize_noveltyCurve(PLP,parameterVis)
%     title('PLP curve')


    %% 6, sonify

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%

%     parameterSoni = [];
```

```
%     parameterSoni.Fs = Fs;
%     parameterSoni.featureRate = featureRate;
%
%     sonification =
sonify_noveltyCurve(noveltyCurve,audio,parameterSoni);
%
%     wavwrite(sonification,Fs,'sonification_novelty.wav')
%
%
%     parameterSoni = [];
%     parameterSoni.Fs = Fs;
%     parameterSoni.featureRate = featureRate;
%
%     sonification = sonify_noveltyCurve(PLP,audio,parameterSoni);
%
%     wavwrite(sonification,Fs,'sonification_PLP.wav')


    %% 7, compute a PLP curve of a restricted tempo range
    % Here, we restrict the range of tempo parameters to [70,110] BPM
    % and compute a restricted PLP curve.
    %
    % Instead of computing a restricted tempogram, we could also set
    % parameterPLP.PLPrange = [70,110] and use the original tempogram
with the
    % tempo set [30,600]

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%

%
%     parameterTempogram.tempoWindow = 6;          % window length in sec
%     parameterTempogram.BPM = 70:1:110;           % tempo values
%
%     [tempogram, T, BPM] =
noveltyCurve_to_tempogram_via_DFT(noveltyCurve,parameterTempogram);
% %     visualize_tempogram(tempogram,T,BPM)
% %     title('Tempogram (Fourier) restricted')
%
%     parameterPLP.tempoWindow = parameterTempogram.tempoWindow;
%
%     [PLP_restricted] = tempogram_to_PLPcurve(tempogram, T, BPM,
parameterPLP);
%     PLP_restricted = PLP_restricted(1:length(noveltyCurve));  % PLP
curve will be longer (zero padding)
%
%     visualize_noveltyCurve(PLP_restricted,100) %parameterVis
%     title('PLP curve (restricted)')

    %sonification =
sonify_noveltyCurve(PLP_restricted,audio,parameterSoni);

    %wavwrite(sonification,Fs,'sonification_PLP-BPMrestricted.wav')

end
```

wav_to_audio.m

```
function [f_audio,sideinfo] =
wav_to_audio(dirAbs,dirRel,wavfilename,parameter)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%
% Name: wav_to_audio
% Date of Revision: 2011-03
% Programmer: Meinard Mueller, Sebastian Ewert
%
% Description:
% Loads a Wav file and fills a sideinfo variable according to AGtoolbox
% specifications. Resampling and single channel conversion is default,
but
% optional.
%
% Input:
%        dirAbs
%        dirRel
%        wavfilename
%        parameter.useResampling = 1;
%        parameter.destSamplerate = 22050;
%        parameter.convertToMono = 1;
%        parameter.monoConvertMode = 'downmix';
%        parameter.message = 0;
%        parameter.vis = 0;
%        parameter.save = 0;
%        parameter.saveDir = [dirAbs,dirRel];
%        parameter.saveFilename = wavfilename;
%
% Output:
%        f_audio
%        sideinfo.wav.version
%        sideinfo.wav.filename
%        sideinfo.wav.dirRel
%        sideinfo.wav.size
%        sideinfo.wav.duration
%        sideinfo.wav.energy
%        sideinfo.wav.fs
%        sideinfo.wav.nbits
%        sideinfo.wav.channels
%        sideinfo.wav.resampled
%        sideinfo.wav.monoConverted
%        sideinfo.wav.monoConvertMode
%
%
% License:
%     This file is part of 'Chroma Toolbox'.
%
%     'Chroma Toolbox' is free software: you can redistribute it and/or
modify
%     it under the terms of the GNU General Public License as published
by
%     the Free Software Foundation, either version 2 of the License, or
%     (at your option) any later version.
%
%     'Chroma Toolbox' is distributed in the hope that it will be useful,
%     but WITHOUT ANY WARRANTY; without even the implied warranty of
%     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
%     GNU General Public License for more details.
%
%     You should have received a copy of the GNU General Public License
```

```
%     along with 'Chroma Toolbox'. If not, see
<http://www.gnu.org/licenses/>.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%
% Check parameters
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%

if nargin<4
    parameter=[];
end
if nargin<3
    error('Please specify at least the path and filename of the wav
file')
end

if isfield(parameter,'useResampling')==0
    parameter.useResampling = 1;
end
if isfield(parameter,'destSamplerate')==0
    parameter.destSamplerate = 44100; %antes, 22050
end
if isfield(parameter,'convertToMono')==0
    parameter.convertToMono = 1;
end
if isfield(parameter,'monoConvertMode')==0
    parameter.monoConvertMode = 'downmix';
end
if isfield(parameter,'message')==0
    parameter.message = 0;
end
if isfield(parameter,'vis')==0
    parameter.vis = 0;
end
if isfield(parameter,'save')==0
    parameter.save = 0;
end
if isfield(parameter,'saveDir')==0
    parameter.saveDir = [dirAbs,dirRel];
end
if isfield(parameter,'saveFilename')==0
    parameter.saveFilename = wavfilename;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%
% Main program
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%
if parameter.message == 1
    fprintf('wav_to_audio: processing %s, ',wavfilename);
end

[pathstr,name,ext] = fileparts(wavfilename);
if strcmp(ext,'.wav')
    [f_audio,fs,nbits] = wavread(strcat(dirAbs,dirRel,wavfilename));
```

```
else
    error(['Unknown file format ' ext]);
end


bConverted_to_mono = 0;
if parameter.convertToMono
    if size(f_audio,2)>1
        bConverted_to_mono = 1;
        if parameter.message == 1
            fprintf('converting to mono, ');
        end
        switch parameter.monoConvertMode
            case 'leftmost_channel'
                f_audio= f_audio(:,1);
            case 'rightmost_channel'
                f_audio= f_audio(:,size(f_audio,2));
            case 'downmix'
                % pay attention to energy loss due to differences in
phase
                % when using this method. This is often the case for bad
                % stereo mixes
                nChannels = size(f_audio,2);

                f_audio = sum(f_audio,2);
                f_audio = f_audio / nChannels;
            otherwise
                disp('wav_to_audio: monoConvertMode : Unknown method')
        end
    end
end

bResampled = 0;
if parameter.useResampling
    if (fs ~= parameter.destSamplerate)
        bResampled = 1;
        if parameter.message == 1
            fprintf('Resampling to %d, ', parameter.destSamplerate);
        end
        f_audio = resample (f_audio,parameter.destSamplerate,fs,100);
        fs = parameter.destSamplerate;
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%
% Update sideinfo
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%
sideinfo.wav.version    = 1;
sideinfo.wav.filename   = wavfilename;
sideinfo.wav.dirRel    = dirRel;
sideinfo.wav.size       = size(f_audio,1);
sideinfo.wav.duration   = (sideinfo.wav.size-1)/fs;
sideinfo.wav.energy     = sum(f_audio.^2);
sideinfo.wav.fs         = fs;
sideinfo.wav.nbits      = nbits;
sideinfo.wav.channels   = size(f_audio,2);
sideinfo.wav.resampled  = bResampled;
sideinfo.wav.monoConverted = bConverted_to_mono;
```

```
if bConverted_to_mono
    sideinfo.wav.monoConvertMode = parameter.monoConvertMode;
else
    sideinfo.wav.monoConvertMode = 'none';
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Saving data
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if parameter.save == 1
    if parameter.message == 1
        fprintf('Saving to file, ');
    end
    filename = strcat(parameter.saveFilename,'_audio');
    save(strcat(parameter.saveDir,filename),'f_audio','sideinfo');
end

if parameter.message == 1
    fprintf('Done\n');
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%
% Visualization
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%
if parameter.vis
    figure;
    for k=1:sideinfo.wav.channels
        if sideinfo.wav.channels > 1
            subplot(sideinfo.wav.channels,1,k);
        end
        plot( [0:sideinfo.wav.size-1] / sideinfo.wav.fs , f_audio(:,k));
        axis tight;
    end
end

end
```

wav_to_audio.m

```
function [noveltyCurve,featureRate] = audio_to_noveltyCurve(f_audio, fs,
parameter)
%AUDIO_TO_NOVELTYCURVE Computes a novelty curve.
%   noveltyCurve = AUDIO_TO_NOVELTYCURVE(f_audio,fs) returns the
%   novelty curve of the audio signal specified by
%   the waveform f_audio in noveltyCurve.
%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
% Name: audio_to_noveltyCurve
% Date of Revision: 2011-10
% Programmer: Peter Grosche
% http://www.mpi-inf.mpg.de/resources/MIR/tempogramtoolbox/
%
% Description:
```

```
% Computes a novelty curve (onset detection function)
% for the input audio signal. This implementation is a variant
% of the widely used spectral flux method with additional
% bandwise processing and a logarithmic intensity compression.
% This particularly addresses music with weak onset information
% (e.g., exhibiting string instruments.)
%
% Input:
%       f_audio : wavefrom of audio signal
%       fs : sampling rate of the audio (Hz)
%       parameter (optional): parameter struct with fields
%               .logCompression : enable/disable log compression
%               .compressionC : constant for log compression
%               .win_len : window length for STFT (in samples)
%               .stepsize : stepsize for the STFT
%               .resampleFeatureRate : feature rate of the resulting
%               novelty curve (resampled, independent of stepsize)
%
%
% Output:
%       noveltyCurve : the novelty curve
%       featureRate : feature rate of the novelty curve (Hz)
%
% License:
%     This file is part of 'Tempogram Toolbox'.
%
%     'Tempogram Toolbox' is free software: you can redistribute it
and/or modify
%     it under the terms of the GNU General Public License as published
by
%     the Free Software Foundation, either version 2 of the License, or
%     (at your option) any later version.
%
%     'Tempogram Toolbox' is distributed in the hope that it will be
useful,
%     but WITHOUT ANY WARRANTY; without even the implied warranty of
%     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
%     GNU General Public License for more details.
%
%     You should have received a copy of the GNU General Public License
%     along with 'Tempogram Toolbox'. If not, see
%     <http://www.gnu.org/licenses/>.
%
%
% Reference:
%   Peter Grosche and Meinard Müller
%   Extracting Predominant Local Pulse Information from Music Recordings
%   IEEE Transactions on Audio, Speech, and Language Processing, 19(6),
1688-1701, 2011.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
% Check parameters
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
```

```
if nargin < 2
    error('fs needed!');
end
if nargin < 3
    parameter = [];
end

parameter.fs = fs;
if isfield(parameter,'win_len')==0
    parameter.win_len = 4096*parameter.fs/44100;
end
if isfield(parameter,'stepsize')==0
    parameter.stepsize = 2048*parameter.fs/44100;
end
if isfield(parameter,'compressionC')==0
    parameter.compressionC = 1000;
end
if isfield(parameter,'logCompression')==0
    parameter.logCompression = true;
end
if isfield(parameter,'resampleFeatureRate')==0
    parameter.resampleFeatureRate = 200;
end


myhann = @(n)  0.5-0.5*cos(2*pi*((0:n-1)'/(n-1)));

%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Compute spectrogram
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%



parameter.returnMagSpec = 1;
parameter.StftWindow = myhann(parameter.win_len);
[specData,featureRate] =
audio_to_spectrogram_via_STFT(f_audio,parameter);
parameter.featureRate = featureRate;



% normalize and convert to dB
specData = specData./max(max(specData));
thresh = -74; % dB
thresh = 10^(thresh./20);
specData = (max(specData,thresh));


%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% bandwise processing
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%

bands = [0 500; 500 1250; 1250 3125; 3125 7812.5; 7812.5
floor(parameter.fs./2)]; %hz
```

```
compressionC = parameter.compressionC;

bandNoveltyCurves = zeros(size(bands,1),size(specData,2));

for band = 1:size(bands,1)

    bins = round(bands(band,:)./ (parameter.fs./parameter.win_len));
    bins = max(1,bins);
    bins = min(round(parameter.win_len./2)+1,bins);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % band novelty curve

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    bandData = specData(bins(1):bins(2),:);
    if parameter.logCompression && parameter.compressionC>0
        bandData = log(1 + bandData.*compressionC)/(log(1+compressionC));
    end

    % smoothed differentiator
    diff_len = 0.3;%sec
    diff_len = max(ceil(diff_len*parameter.fs/parameter.stepsize),5);
    diff_len = 2*round(diff_len./2)+1;
    diff_filter = myhann(diff_len).*[-1*ones(floor(diff_len/2),1);
0;ones(floor(diff_len/2),1) ];
    diff_filter = diff_filter(:)';
    bandDiff = filter2(diff_filter,
[repmat(bandData(:,1),1,floor(diff_len/2)),bandData,repmat(bandData(:,end
),1,floor(diff_len/2))]);
    bandDiff = bandDiff.*(bandDiff>0);
    bandDiff = bandDiff(:,floor(diff_len/2):end-floor(diff_len/2)-1);


    % normalize band
    norm_len = 5;%sec
    norm_len = max(ceil(norm_len*parameter.fs/parameter.stepsize),3);
    norm_filter = myhann(norm_len);
    norm_filter = norm_filter(:)';
    norm_curve = filter2(norm_filter./sum(norm_filter),sum(bandData));
    % boundary correction
    norm_filter_sum = (sum(norm_filter)-
cumsum(norm_filter))./sum(norm_filter);
    norm_curve(1:floor(norm_len/2)) = norm_curve(1:floor(norm_len/2))./
fliplr(norm_filter_sum(1:floor(norm_len/2)));
    norm_curve(end-floor(norm_len/2)+1:end) = norm_curve(end-
floor(norm_len/2)+1:end)./ norm_filter_sum(1:floor(norm_len/2));
    bandDiff = bsxfun(@rdivide,bandDiff,norm_curve);

    % compute novelty curve of band
    noveltyCurve = sum( bandDiff);
    bandNoveltyCurves(band,:) = noveltyCurve;

end

% figure;
% for band = 1:5
%     subplot(5,1,6-band);plot(bandNoveltyCurves(band,:));
```

```
%     ylim([0 1])
% end



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% summary novelty curve
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

noveltyCurve = mean(bandNoveltyCurves);


% resample curve
if (parameter.resampleFeatureRate >0 && parameter.resampleFeatureRate ~=
parameter.featureRate)
    [noveltyCurve,featureRate] =
resample_noveltyCurve(noveltyCurve,parameter);
    parameter.featureRate = featureRate;

end


% local average subtraction
[noveltyCurve] = novelty_smoothedSubtraction(noveltyCurve,parameter);



end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
% Local Functions
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%

function [noveltySub,local_average] =
novelty_smoothedSubtraction(noveltyCurve,parameter)

    myhann = @(n)  0.5-0.5*cos(2*pi*((0:n-1)'/(n-1)));
    smooth_len = 1.5;%sec
    smooth_len = max(ceil(smooth_len*parameter.fs/parameter.stepsize),3);
    smooth_filter = myhann(smooth_len);
    smooth_filter = smooth_filter(:)';
    local_average =
filter2(smooth_filter./sum(smooth_filter),noveltyCurve);

    noveltySub = (noveltyCurve-local_average);
    noveltySub = (noveltySub>0).*noveltySub;
end

function [noveltyCurve,featureRate] =
resample_noveltyCurve(noveltyCurve,parameter)

    p = round(1000*parameter.resampleFeatureRate./parameter.featureRate);
    noveltyCurve = resample(noveltyCurve,p,1000,10);
    featureRate = parameter.featureRate*p/1000;
end
```

noveltyCurve_to_tempogram_via_DFT.m

```
function [tempogram, T, BPM] =
noveltyCurve_to_tempogram_via_DFT(novelty,parameter)
%NOVELTYCURVE_TO_TEMPOGRAM_VIA_DFT Computes a tempogram using STFT.
%   [tempogram] = noveltyCurve_to_tempogram_via_DFT(novelty) returns the
%   complex valued fourier tempogram tempogram for a novelty curve
novelty.
%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
% Name: noveltyCurve_to_tempogram_via_DFT
% Date of Revision: 2011-10
% Programmer: Peter Grosche
% http://www.mpi-inf.mpg.de/resources/MIR/tempogramtoolbox/
%
% Description:
% Computes a complex valued fourier tempogram for a given novelty curve
% indicating note onset candidates in the form of peaks.
% This implementation provides parameters for chosing fourier
% coefficients in a frequency range corresponding to musically meaningful
% tempo values in bpm.
%
% To use the fast implementation as mex file,
compute_fourierCoefficients.c
% needs to be compiled by calling "mex compute_fourierCoefficients.c".
%
% Input:
%       novelty : a novelty curve indicating note onset positions
%       parameter (optional): parameter struct with fields
%               .featureRate : feature rate of the novelty curve (Hz).
%                       This needs to be set to allow for setting other
%                       parameters in seconds!
%               .tempoWindow: Analysis window length in seconds
%               .stepsize: window stepsize in frames (of novelty curve)
%               .BPM: vector containing BPM values to compute
%               .useImplementation: indicating the implementation to use
%                   1 : c/mex implementation (fast, needs to be compiled)
%                   2 : Matlab implementation (slow)
%                   3 : Goertzel algorithm via Matlab's spectrogram
%
%
% Output:
%       tempogram : the complex valued fourier tempogram
%       T : vector of time positions (in sec) for the frames of the
tempogram
%       BPM: vector of bpm values of the tempo axis of the tempogram
%
%
% License:
%     This file is part of 'Tempogram Toolbox'.
%
%     'Tempogram Toolbox' is free software: you can redistribute it
and/or modify
%     it under the terms of the GNU General Public License as published
by
%     the Free Software Foundation, either version 2 of the License, or
%     (at your option) any later version.
```

```
%
%     'Tempogram Toolbox' is distributed in the hope that it will be
useful,
%     but WITHOUT ANY WARRANTY; without even the implied warranty of
%     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
%     GNU General Public License for more details.
%
%     You should have received a copy of the GNU General Public License
%     along with 'Tempogram Toolbox'. If not, see
%     <http://www.gnu.org/licenses/>.
%
%
% Reference:
%   Peter Grosche and Meinard Müller
%   Extracting Predominant Local Pulse Information from Music Recordings
%   IEEE Transactions on Audio, Speech, and Language Processing, 19(6),
1688-1701, 2011.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
% Check parameters
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%


if nargin < 2
    parameter = [];
end
if isfield(parameter,'featureRate')==0
    warning('parameter.featureRate not set! Assuming 1!')
    parameter.featureRate = 1;
end
if isfield(parameter,'tempoWindow')==0
    parameter.tempoWindow = 6; % in sec
end
if isfield(parameter,'BPM')==0
    parameter.BPM = 30:1:600;
end

if isfield(parameter,'stepsize')==0
    parameter.stepsize = ceil(parameter.featureRate./5); % 5 Hz default
end
if isfield(parameter,'useImplementation')==0
    parameter.useImplementation = 2; % 1: c implementation, 2: MATLAB
implementation, 3: spectrogram via goertzel algorithm
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%
% Main
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%

win_len = round(parameter.tempoWindow.* parameter.featureRate);
win_len = win_len + mod(win_len,2) - 1;
parameter.tempoRate = parameter.featureRate./parameter.stepsize;
```

```
myhann = @(n)  0.5-0.5*cos(2*pi*((0:n-1)'/(n-1)));

windowTempogram = myhann(win_len);

novelty = [zeros(1,round(win_len/2)), novelty,
zeros(1,round(win_len/2))]';

switch parameter.useImplementation
    case 1, % c/mex implementation
        if exist('compute_fourierCoefficients','file')~=3
            error('noveltyCurve_to_tempogram_via_DFT: mex function
compute_fourierCoefficients not found. Compile (mex
compute_fourierCoefficients.c or COMPILE.m), or set
parameter.useImplementation to use the Matlab implementation (slow)!')
        end
        tic
        [tempogram, BPM, T] =
compute_fourierCoefficients(novelty,windowTempogram,win_len-
parameter.stepsize, (parameter.BPM./60)', parameter.featureRate);
        % normalize window (parsevals theorem)
        tempogram = tempogram./sqrt(win_len) / sum(windowTempogram) *
win_len;
        T = T(:)';
%         fprintf('C runtime: \t%f\n',toc);
    case 2, % matlab implementation
        tic
        [tempogram, BPM, T] =
compute_fourierCoefficients_matlab(novelty,windowTempogram,win_len-
parameter.stepsize, parameter.BPM./60, parameter.featureRate);
        tempogram = tempogram./sqrt(win_len) / sum(windowTempogram) *
win_len;
%         fprintf('Matlab runtime: \t%f\n',toc);
    case 3, % goertzel via matlabs spectrogram
        tic
        [tempogram, BPM, T] =
spectrogram(novelty,windowTempogram,win_len-parameter.stepsize,
parameter.BPM./60, parameter.featureRate );
        tempogram = tempogram./sqrt(win_len) / sum(windowTempogram) *
win_len;
%         fprintf('Goertzel runtime: \t%f\n',toc);
end

BPM = BPM.*60;
T = T - T(1);
```

tempogram_to_PLPcurve.m

```
function [PLP,featureRate] = tempogram_to_PLPcurve(tempogram, T, BPM,
parameter)
%TEMPOGRAM_TO_PLPCURVE Computes a PLP curve from a tempogram.
%   [PLP] = tempogram_to_PLPcurve(tempogram, T, BPM) returns the
%   PLP curve derived from the complex valued fourier tempogram tempogram
%   with time and tempo axis T and BPM,m respectively.
%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
% Name: noveltyCurve_to_tempogram_via_DFT
```

```
% Date of Revision: 2011-10
% Programmer: Peter Grosche
% http://www.mpi-inf.mpg.de/resources/MIR/tempogramtoolbox/
%
%
% Description:
% computes a PLP curve from a complex tempogram representation
% uses maximum values for each frame (predominant local periodicity)
% or a given tempocurve
% Input:
%        tempogram: (complex valued)
%        BPM: frequency axis of tempogram (in BPM)
%        T: time axis of tempogram (in sec)
%        parameter
%           .featureRate: feature rate of novelty curve, same is used for
%                         PLP curve
%           .tempoWindow: window size in sec (same as used for tempogram
computation)
%           .stepsize: stepsize used in the tempogram computation
%           .useTempocurve: if set to 1, tempocurve is used instead of
max values
%           .tempocurve: tempocurve (in BPM), one entry for each
tempogram frame,
%                used instead of predominant periodicity values if
useTempocurve == 1
%           .PLPrange: range of BPM values searched for predominant
%                      periodicities
%
%
% Output:
%        PLP: PLP curve
%        featureRate: feature rate of the PLP curve
%
%
% License:
%     This file is part of 'Tempogram Toolbox'.
%
%     'Tempogram Toolbox' is free software: you can redistribute it
and/or modify
%     it under the terms of the GNU General Public License as published
by
%     the Free Software Foundation, either version 2 of the License, or
%     (at your option) any later version.
%
%     'Tempogram Toolbox' is distributed in the hope that it will be
useful,
%     but WITHOUT ANY WARRANTY; without even the implied warranty of
%     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
%     GNU General Public License for more details.
%
%     You should have received a copy of the GNU General Public License
%     along with 'Tempogram Toolbox'. If not, see
%     <http://www.gnu.org/licenses/>.
%
%
% Reference:
%   Peter Grosche and Meinard Müller
%   Extracting Predominant Local Pulse Information from Music Recordings
%   IEEE Transactions on Audio, Speech, and Language Processing, 19(6),
1688-1701, 2011.
%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
% Check parameters
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%


if nargin < 4
    parameter = [];
end

if nargin < 3
    error('Not enough arguments!');
end

if isfield(parameter,'featureRate')==0
    warning('featureRate unset! Assuming 100.');
    parameter.featureRate = 100;
end
if isfield(parameter,'tempoWindow')==0
    warning('tempo window length unset! Assuming 6 sec.');
    parameter.tempoWindow = 6; % sec
end

if ~isfield(parameter,'useTempocurve')
    parameter.useTempocurve = 0;
end

if ~isfield(parameter,'tempocurve')
    parameter.tempocurve = 0;
end

if ~isfield(parameter,'PLPrange') | isempty(parameter.PLPrange) |
parameter.PLPrange == 0
    parameter.PLPrange = [BPM(1) BPM(end)];
end

if isfield(parameter,'stepsize')==0
    parameter.stepsize = ceil(parameter.featureRate./5); % 5 Hz default
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
% Variables
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%

myhann = @(n)  0.5-0.5*cos(2*pi*((0:n-1)'/(n-1)));

if isreal(tempogram)
   error('Complex valued fourier tempogram needed for computing PLP
curves!')
end
```

```
tempogramAbs = abs(tempogram);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
% determine BPM range
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%

[tmp, range(1)] = min(abs(BPM-parameter.PLPrange(1)));
[tmp, range(2)] = min(abs(BPM-parameter.PLPrange(2)));


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
% determine predominant local periodicity or use tempocurve
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%

local_max = zeros(size(tempogramAbs,2),1);
if ~parameter.useTempocurve
    for frame = 1:size(tempogramAbs,2)
        [tmp,local_max(frame)] =
max(tempogramAbs(range(1):range(2),frame));
        local_max(frame) = local_max(frame) + range(1)-1;
    end
else
    for frame = 1:size(tempogramAbs,2)
        [tmp,idx] = min(abs(BPM - parameter.tempocurve(frame)));
        local_max(frame) = idx;
    end
end



%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% cumulate periodiciy kernels
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%

win_len = round(parameter.tempoWindow.* parameter.featureRate);
win_len = win_len + mod(win_len,2) -1;

t = (T.*parameter.featureRate);
% if novelty curve is zero padded with half a window on both sides, PLP
is
% always larger than novelty
PLP = zeros(1,(size(tempogram,2)-1+1).*parameter.stepsize);

window = myhann(win_len)';
 % normalize window so sum(window) = length(window), like it is for box
 % window
window = window./(sum(window)./win_len);

% normalize window according to overlap, this guarantees, that
max(PLP)<=1
window = window./(win_len./parameter.stepsize);
```

```
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% overlap add
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%

for frame = 1:size(tempogram,2)

    t0 = ceil(t(frame) - win_len/2 );
    t1 = floor(t(frame) + win_len/2 );

    phase = angle(tempogram(local_max(frame), frame));
    Tperiod = parameter.featureRate.*60./BPM(local_max(frame)); % period
length
    len = (t1 - t0 + 1)/Tperiod;  % how many periods?

    cosine = window.*cos( (0:1/Tperiod:len-1/Tperiod)* 2* pi + phase ); %
create cosine

    if t0 < 1
        cosine = cosine(-t0+2:end);
        t0 = 1;
    end

    if t1 > size(PLP,2)
        cosine = cosine(1:end + size(PLP,2)-t1);
        t1 = size(PLP,2);
    end

    PLP(t0:t1) = PLP(t0:t1) + cosine;
end



%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% half wave rectification
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%

PLP = PLP.*(PLP>0);
featureRate = parameter.featureRate;
```

decision.m[40]

```
function [tagscell,delta] = decision(D, PLP,control)

%%%%%PARÁMETROS DE ENTRADA:
    %D:     matriz de distancias euclídeas de los patrones de acorde con
            %los vectores de croma estudiados.

            %Dimensión MxNxC, siendo:
```

[40] En este punto finalizan las funciones de Grosche y Müller y continúa el curso del programa.

```
                %M, número de frames de acuerdo con la resolución
temporal del
                %sistema;
                %N = 12, número de clases de altura;
                %C = 5, tipos de acorde

            %Por tanto, hay una matriz Mx12 por cada tipo de acorde (5 en
            %total). A saber:
                %C = 1 --> mayor
                %C = 2 --> menor
                %C = 3 --> disminuido
                %C = 4 --> aumentado
                %C = 5 --> suspendido
                %...C = 6 --> silencio (añadido)

%%%%%PARÁMETROS DE SALIDA:
    %tagscell:  vector columna con las etiquetas de los acordes más
                %probables en cada instante de tiempo discreto (tipo
                %de datos cell array)

%%%%%%%UMBRAL DE DETECCIÓN DE ACORDE%%%%%%%

    %Discriminación de situaciones polifónicas y monofónicas
    %o de silencio

    %alpha = 1.5; %umbral de detección de acorde; minimum Euclidean
    %distance

    if(control.preProcess == 1)
        alpha = 1.5; %We assume that, if HP Separation is needed (our
pre-
        %processing step), the signal will contain drums and a higher
value
        %for the threshold is also required
    else
        alpha = 0.95;
    end

%%%%%%%GENERACIÓN DE ETIQUETAS DE ACORDE%%%%%%%

    [M,~,~] = size(D);
    %M frames
    %N = 12 clases de altura
    %C = 5 tipos de acorde más el caso del silencio

    %Inicializacion del vector salida de etiquetas:
    tagscell = repmat({''},1,M);

    %Etiquetas de tónicas
    refTagRoot = {'A','Bb','B','C','C#','D','Eb','E','F','F#','G','Ab'};

    if(control.PLPComputation == 1)

        %CONDICIÓN DE ESTABILIDAD%
        %Definimos heurísticamente un tiempo mínimo transcurrido %para
que
        %pueda cambiar el estado de la decisión basándonos en la curva
PLP
        %(correlacionada con los onsets)
```

```
        %Encontrar los índices de todos los Predominant Local Pulses
(PCPs)
        l = 1;
        peak_index = zeros(1,length(PLP));
        for i=2:length(PLP)-1
           if (PLP(i-1)<PLP(i) && PLP(i)>PLP(i+1) && PLP(i)>=0.3)
               %condición de máximo valor en la curva
               peak_index (l) = i;
               l = l + 1;
           end
        end
        I = peak_index(1:l); %nuevo vector de índices de los PCPs en la
curva


        min_t_entre_picos = 99999;
        for i = 1:l-1
           if( (I(i+1)-I(i)) < min_t_entre_picos && I(i+1)~=0 )
                min_t_entre_picos = I(i+1)-I(i);
                %caso más restrictivo de distancia entre PLPs
           end
        end

        n = (0:length(PLP)-1)./200; %eje de la curva PLP en segundos
        delta_segundos = max(n)*min_t_entre_picos/length(PLP); %Time
interval
        %between each peak (seconds)
        delta = round(delta_segundos * (M/max(n))); %Time interval
between each
        %peak (samples)
        %delta = 2*delta %Assumption of binary patterns typical of 4/4
time
        %signature

        cont = 0; %contador de muestras (al llegar a delta muestras,
permite
        %decidir un nuevo acorde y se reinicia)
    else
        if(control.postProcess == 1)
            delta = 3;
        else
            delta = 1;
        end

        cont = 0;
    end


%%%%%%%BÚSQUEDA DEL ACORDE MÁS PROBABLE%%%%%%%

        for i = 1:M %M frames

            cont = cont + 1;

            menorDistMaj = 99999; %inicialización a infinito
            menorDistMin = 99999;
            menorDistDim = 99999;
            menorDistAug = 99999;
```

```
            menorDistSus = 99999;
            menorDistNull = 99999;

            posAcordeMaj = 0;
            posAcordeMin = 0;
            posAcordeDim = 0;
            posAcordeAug = 0;
            posAcordeSus = 0;

            if(i < delta || cont == delta)

                for j = 1:12 %12 elementos de croma

                    %%%ENCONTRAR LA DISTANCIA MÁS CORTA%%%
                    %Mayores
                    if( D(i,j,1) < menorDistMaj )
                         %disp('Mayores');
                         menorDistMaj = D(i,j,1);
                         posAcordeMaj = j; %Posición del acorde mayor más
probable
                    end

                    %Menores
                    if( D(i,j,2) < menorDistMin )
                         %disp('Menores');
                         menorDistMin = D(i,j,2);
                         posAcordeMin = j; %Posición del acorde menor más
probable
                    end

                     %Disminuidos
                    if( D(i,j,3) < menorDistDim )
                         %disp('Disminuidos');
                         menorDistDim = D(i,j,3);
                         posAcordeDim = j; %Posición del acorde menor más
probable
                    end

                     %Aumentados
                    if( D(i,j,4) < menorDistAug )
                         %disp('Aumentados');
                         menorDistAug = D(i,j,4);
                         posAcordeAug = j; %Posición del acorde menor más
probable
                    end

                     %Suspendidos
                    if( D(i,j,5) < menorDistSus )
                         %disp('Suspendidos');
                         menorDistSus = D(i,j,5);
                         posAcordeSus = j; %Posición del acorde menor más
probable
                    end

                    %No acorde
                    if( D(i,j,6) < menorDistNull )
                         %disp('Suspendidos');
                         menorDistNull = D(i,j,6);
                    end
```

```
                end

                %%%DECISIÓN%%%

                %Elección del acorde patron con menor distancia euclidea
al acorde
                %real que se ha leido del cromagrama

                %Sintaxis de acordes propuesta en Harte, et al., 2005)

                v = [menorDistMaj, menorDistMin, menorDistDim,
menorDistAug, menorDistSus,menorDistNull];
                [y,I] = min(v);

                if(y < alpha)
                    switch (I)
                        case 1
                            %disp('Major chord detected...')
                            tagscell(i) =
strcat(refTagRoot(posAcordeMaj),':maj');

                        case 2
                            %disp('Minor chord detected...')
                            tagscell(i) =
strcat(refTagRoot(posAcordeMin),':min');

                        case 3
                            %disp('Diminished chord detected...')
                            tagscell(i) =
strcat(refTagRoot(posAcordeDim),':dim');

                        case 4
                            %disp('Augmented chord detected...')
                            tagscell(i) =
strcat(refTagRoot(posAcordeAug),':aug');

                        case 5
                            %disp('Suspended chord detected...')
                            tagscell(i) =
strcat(refTagRoot(posAcordeSus),':sus');
                        case 6
                            tagscell(i) = {'N'};
                    end
                else
                    tagscell(i) = {'N'}; %no-detección de acorde (entrada
                    %monofónica o silencio)
                end

                cont = 0;
            else

                    tagscell(i) = tagscell(i-1); %no cambio


            end

        end
```

```
end
```

medianFilter.m

```
function [processed_tags] = medianFilter(tags,delta)


%---------------------------MEDIAN FILTERING-----------------------------
-%

    M = length(tags);
    processed_tags = repmat({''},1,M);

    %Reference tags (including 4 common cases of enharmony)
    refTags = {'C:maj';'C#:maj';'Db:maj';'D:maj';'D#:maj';'Eb:maj';...
        'E:maj';'F:maj';'F#:maj';'G:maj';'G#:maj';'Ab:maj';'A:maj';...
        'A#:maj';'Bb:maj';'B:maj';'C:min';'C#:min';'Db:min';'D:min';...
        'D#:min';'Eb:min';'E:min';'F:min';'F#:min';'G:min';'G#:min';...
        'Ab:min';'A:min';'A#:min';'Bb:min';'B:min';'C:dim';'C#:dim';...
        'Db:dim';'D:dim';'D#:dim';'Eb:dim';'E:dim';'F:dim';'F#:dim';...
        'G:dim';'G#:dim';'Ab:dim';'A:dim';'A#:dim';'Bb:dim';'B:dim';...
        'C:aug';'C#:aug';'Db:aug';'D:aug';'D#:aug';'Eb:aug';'E:aug';...
        'F:aug';'F#:aug';'G:aug';'G#:aug';'Ab:aug';'A:aug';'A#:aug';...
        'Bb:aug';'B:aug';'C:sus';'C#:sus';'Db:sus';'D:sus';'D#:sus';...
        'Eb:sus';'E:sus';'F:sus';'F#:sus';'G:sus';'G#:sus';'Ab:sus';...
        'A:sus';'A#:sus';'Bb:sus';'B:sus';'N'};

    %Check for the first chord
    cent = 0;
    av = 1;

    while(cent == 0 && isequal(tags(av),{'N'}) == 1)

        if(av>length(tags))
            cent = 1;
        end

        av = av + 1;
    end
    processed_tags(1:av) = {'N'};

    while(av < M)

        if((av+delta-1) > M) %Final segment
           delta = M - av + 1;
        end
        alias = zeros(delta,1);%Array which will contain the numbers
        %representing the different chord labels (their 'alias')
        k = 1;

        for j=av:(av+delta-1)
```

```
            %Assignment of values to chord labels in order to use the
            %median filter
            if(j<=M)
                alias(k) = getNumber(tags(j),refTags);
            end
            k = k + 1;
        end

         %Median calculation
            result = median(alias);
            %Retrieve the new label according to the value obtained
            if(result ~= 0)
                %This is the usual case
                processed_tags(av:av+delta-1) = ...
                    cellstr(refTags(round(result)));
            end

        av = av + delta;
    end

end
 function [index] = getNumber(currentTag,refTags)


    index = 0;
    currentTagCell = cellstr(currentTag);
    cent = 0;
    i = 1;
    while(cent == 0 && i<=length(refTags))

        if(isequal(refTags(i),currentTagCell) == 1)
            index = i;
            cent = 1;
        end

        i = i + 1;
    end

    %Enharmony is not considered completely

end
```

saveResults.m

```
function [] = saveResults(X,tags,filename)

%-------------------------ESCRITURA DEL FICHERO--------------------------
-%

    %Recordemos la estructura del fichero "X":
    %   13 columnas (eje temporal + 12 clases de alturas)
    %   M filas (numero de muestras temporales)
```

```
    %tags: vector con las etiquetas de los acordes


    [M,~] = size(X); %lectura del número de muestras
    time = zeros(M,1); %inicialización del eje temporal; 1 columna y
    %M filas (M muestras de tiempo discreto)

    %Composición del archivo de salida
    time(:,1) = X(:,1); % instantes de tiempo
    time(:,1) = round(time(:,1)*1000)/1000; %truncado a la milésima

    k = strfind(filename, '.wav');
    estimationFileName = strcat('Chord_Estimation/Chord_Estimation_',
filename(1:k-1),'.csv');
    fid = fopen(estimationFileName,'w');
    for i = 1:M
        b = num2str(time(i,1)); %conversión de numero a string
        %b = strrep(b, '.', ','); %sustitución de puntos por comas
        fprintf(fid,'%s,%s\n',b,char(tags(i)));
        %fprintf(fid,'%i,%s\n',time(i,1),char(tags(i)));
    end

    fclose(fid);


end
```

metrics.m

```
function [GTChords] = metrics(filename)

%============== METRICS FOR MEASURING THE PERFORMANCE OF
=================
%               THE AUTOMATIC CHORD ESTIMATION ALGORITHM
%

    k = strfind(filename,'.wav');
    GTFileName = strcat('GT/GT_',filename(1:k-1),'.csv');
    GTFileName = strrep(GTFileName,'Harmonic_','');
    %just in case we are dealing with pre-processed audio files
    estimationFileName = strcat('Chord_Estimation/Chord_Estimation_',
filename(1:k-1),'.csv')

    %Load of the two comma-separated text files, only retaining the
column
    %corresponding to chord labels
    EstimatedChords = LoadFile(estimationFileName);
    GTChords = LoadFile(GTFileName);

    M_estimated = length(EstimatedChords);
    M_gt = length(GTChords);
    M = min(M_estimated,M_gt);
```

```
    eval = 0;
    for u=1:M %Assumption of same lengths for both Estimated and GT
arrays

        %Otherwise, we will choose the  dimension and pad the smallest
        %array with zeros and 'N's.


%-------ADAPTATION OF ISOPHONICS' SYNTAX TO FIT HARTE'S SYNTAX*----------
-%
%                       AND OUR EVALUATION APPROACH
%                         (*)(Harte, et al., 2005)
        GT = char(GTChords(u)); %u-th chord (casting cell to char)

        %CHECK FOR INVERSIONS
        F = strfind(GT,'/');
        if(isempty(F) ~= 1)
            %This is an inversion:
            %As we do not consider inversion of chords, we will just
            %truncate this piece of information (all the content after
'/')
            %for being able to apply our metrics to chords regardless of
            %their inversions
            GT = GT(1:F-1);
        end

        %Henceforth, there are no notated inversions.

        %CHECK FOR MAJOR TRIADS NOTATED AS 'ROOT' AND NONE QUALITY
        K = strfind(GT,':');
        %If there are no ':' symbol and we are not in the 'no-chord'
        %case:
        if(isempty(K) == 1 && isequal(GT,'N') == 0)
            %Major chords should be labeled as 'X:maj'
             GT = strcat(GT(1:length(GT)),':maj');
        end

         %CHECK FOR LISTS OF PITCHES AND ADDED NOTES
        P = strfind(GT,'(');
        if(isempty(P) ~= 1)
           %We may just have a fifth chord, for instance, with only
(1,b3)
           %degrees. Our system will evaluate that case identifying only
           %the root.
           %We may also have a (#9) added. It will hinder the performance
           %in any case, but cannot be detected since our vocabulary is
           %modest
           GT = GT(1:P-1);

        end

        %CHECK FOR DOMINANTS
        I = strfind(GT,':7');
        J = strfind(GT,':9');
        if(isempty(I)~=1)
            %The chord is dominant. As we do not consider dominant
chords,
```

```
            %we only retain the 'maj' quality
            GT = strcat(GT(1:I),'maj');
        end
        if(isempty(J)~=1)
            %Same situation here: dominant seventh chord with ninth
added.
            %We only consider the 'maj' quality for evaluation purposes
            GT = strcat(GT(1:J),'maj');
        end

        %CHECK FOR TENSIONS {min6,min7,min9,maj6,maj7,maj9,...}
        n = GT(length(GT)); %last character of GT
        if(isstrprop(n,'alpha') == 0) %i.e. it is numeric
            %There is a tension or seventh. In our evaluation approach,
            %tensions and sevenths are negligible.
            GT = GT(1:length(GT)-1);
        end




%------------------------ROOT-QUALITY SEPARATION-------------------------
-%

        % Splitting chord root and chord quality(there may be inversions
or
        % other particularities in the quality field of GTarray)
        [EstimatedRoot,EstimatedType] =
CharSplit(char(EstimatedChords(u)),':');
        [GTRoot,GTType] = CharSplit(GT,':');

        %(*)GT is a processed version of GTChords(u) for it to match with
        %our chord vocabulary and syntax (already casted to char)
        GTChords(u) = cellstr(GT);


%-------------------------FRAME-BY-FRAME COMPARISON----------------------
-%
                    %Quest for the awarded situations

        %1) SAME CHORD and SAME ROOT CASES
        if(isequal(EstimatedRoot, GTRoot) == true)
           if(isequal(EstimatedType, GTType) == true)
               eval = eval + 1; %Same chord; true positive
           else
               eval = eval + 0.5; %Same root
           end
        else
           %2) RELATIVE MIN/MAJ CASE
           if(isequal(GTType,{'min'}) == true)
               %The ground-truth chord is minor, thus the RELATIVE MAJOR
root must be 3 half-steps upwards
               %It will be relevant if this relative chord matches the
estimated chord

               relRoot = RelativeRoot(GTRoot,{'maj'}); %Delivers the
relative major root
```

```
                if((isequal(relRoot,EstimatedRoot) == true) &&
(isequal({'maj'}, EstimatedType) == true))
                     eval = eval + 0.3; %Relative major (just one mistaken
degree);
                end
            else
                if(isequal(GTType,{'maj'}) == true)
                    %The ground-truth chord is major, thus the RELATIVE
MINOR root must be 3 half-steps downwards
                    %It will be relevant if this relative chord matches
the estimated chord

                    relRoot = RelativeRoot(GTRoot,{'min'}); %Delivers the
relative minor root

                    if((isequal(relRoot,EstimatedRoot) == true) &&
(isequal({'min'}, EstimatedType) == true))
                         eval = eval + 0.3; %Relative minor (just one
mistaken degree);
                    end
                end

            end

        end

    end


    eval = eval/M;
    FramesEstString = ['Estimated frames: ',num2str(M_estimated)];
    disp(FramesEstString)
    FramesGTString = ['Ground-truth frames: ',num2str(M_gt)];
    disp(FramesGTString)
    MString = ['Frames evaluated: ',num2str(M)];
    disp(MString)
    ResultString = ['Accuracy achieved : ',num2str(eval)];

    disp(ResultString);


end



function Out = LoadFile(PathToFile)
    %Pointer to file:
    fid = fopen(char(PathToFile),'r');
    %Loading the file (cell structure):
    C = textscan(fid, repmat('%s',1,2), 'delimiter',',',
'CollectOutput',true);
    %Retrieving chord labels:
    C = C{1};
    Out = C(:,2);
    %Closing file:
    fclose(fid);
end
```

```
function [a,b] = CharSplit(x,delimiter)
    %x: char type variable
    %delimiter: char type variable

    %a: cell-array type variable
    %b: cell-array type variable

    %Length of the file
    l = length(x);
    %Looking for the delimiter character

    i = 1;
    cent = 0;
    while(cent == 0 && i < l)

       if(x(i) == delimiter)
          p = i; %Position of the delimiter
          x1 = x(1:(p-1)); %Retrieving chord root (before ':')
          x2 = x((p+1):l); %Retrieving chord quality (after ':')
          a = cellstr(x1);
          b = cellstr(x2);
          cent = 1;
       end

       i = i + 1;
    end

    if(cent == 0)
        a = {''};
        b = {''};
    end

end

function [relRoot] = RelativeRoot(GTRoot,StrType)


    %Determining whether GTRoot is sharped or flattened (for dealing with
    %enharmony)
    if(isempty(strfind(GTRoot, 'b')) ~= true) %If it is not "true",
GTRoot is flattened
        PitchClasses = {'A' 'Bb' 'B' 'C' 'Db' 'D' 'Eb' 'E' 'F' 'Gb' 'G'
'Ab'}; %For flattened roots
    else
        PitchClasses = {'A' 'A#' 'B' 'C' 'C#' 'D' 'D#' 'E' 'F' 'F#' 'G'
'G#'}; %For both sharpened and natural roots
    end

    %Position of the GTRoot in the chromatic scale (indexing in A, i.e. A
=> 1)
    p = 0;
    i = 1;
    while ((i <= 12) && (p == 0))
       if(isequal(PitchClasses(i), cellstr(GTRoot)) == true)
           p = i;
       end
       i = i + 1;
    end
```

```
    %Finding the relative chord
    if(isequal(StrType,{'maj'}) == true) %It will get the relative major
chord

        RelativeChordPosition = p + 3;

        if(RelativeChordPosition > 12)
           RelativeChordPosition = RelativeChordPosition - 12;
        end

        relRoot = PitchClasses(RelativeChordPosition);

    else %It will get the relative minor chord

        %if(RelativeChordPosition ~= 0)
            RelativeChordPosition = p - 3;

            if(RelativeChordPosition <= 0)
               RelativeChordPosition = RelativeChordPosition + 12;
            end

            relRoot = PitchClasses(RelativeChordPosition);
        %end
    end


end
```

Anexo B. Código fuente adicional

estimacion de tonalidad.m

```
%ESTIMACIÓN DE LA TONALIDAD MEDIANTE LA SUMA Y NORMALIZACIÓN DE LOS
%VECTORES DE CROMA EN TODOS LOS INSTANTES DE TIEMPO

X = load('2-iv_vamp_nnls-chroma_nnls-chroma_chroma.csv'); % Carga del
cromagrama (.csv)
X(:,2:13) = X(:,2:13)/(max(max(X(:,2:13)))); % Normalizacion
%de los vectores de croma

%Recordemos la estructura del fichero:
%   13 columnas (eje temporal + 12 clases cromaticas)
%   M filas (numero de muestras temporales)

[M,~] = size(X);

tonality = zeros(1,12);

for i=1:M
    tonality = tonality + X(i,2:13);
end

tonality = tonality/max(tonality); %normalización
tonality = round(tonality*1e4)/1e4; %truncado a cuatro decimales

%Representación
pitchClasses = {'A','A#','B','C','C#','D','D#','E','F','F#','G','G#'};

%Exportación de resultados para representación en Excel
fid = fopen('2-iv_nnls_tonalidad.csv','w');
    for i = 1:12
        b = num2str(tonality(i)); %conversión de numero a string
        b = strrep(b, '.', ','); %sustitución de puntos por comas
        fprintf(fid,'%s;%s\n',b,char(pitchClasses(i)));
    end

fclose(fid);
```

audio de prueba.csd[41]

```
<CsoundSynthesizer>
<CsOptions>
-W -o quintas.wav
</CsOptions>
<CsInstruments>

giTSeno ftgen 100, 0, 4096, 10, 1

instr 1
	iAmp = ampdbfs(p4) ; de dBFS a valor de amplitud
	iFrec = cpspch(p5) ; de octava.nota a Hercios (he bajado una octava)
	iTabla = p6
	iModo = p7 ; 0 = mayor, 1 = menor
	iFrIII = iFrec*2^(4/12) ; Frecuencia de la III segun relacion bien temperada
	iFriii = iFrec*2^(3/12) ; Frecuencia de la iii segun relacion bien temperada
	iFrV = iFrec*2^(7/12) ; Frecuencia de la V segun relacion bien temperada

	if iTabla==1 then
		a1 oscils iAmp/3, iFrec, 0
	if iModo==0 then
		a3 oscils iAmp/3, iFrIII, 0
	else
		a3 oscils iAmp/3, iFriii, 0
	endif
		a5 oscils iAmp/3, iFrV, 0
	elseif iTabla==2 then
		a1 buzz iAmp/3, iFrec, sr/(2*iFrec), giTSeno
	if iModo==0 then
		a3 buzz iAmp/3, iFrIII, sr/(2*iFrIII), giTSeno
	else
		a3 buzz iAmp/3, iFriii, sr/(2*iFriii), giTSeno
	endif
		a5 buzz iAmp/3, iFrV, sr/(2*iFrV), giTSeno
	elseif iTabla==3 then
		a1 gbuzz iAmp/6, iFrec, sr/(2*iFrec), 1, 0.67, giTSeno
	if iModo==0 then
		a3 gbuzz iAmp/6, iFrIII, sr/(2*iFrIII), 1, 0.67, giTSeno
	else
		a3 gbuzz iAmp/6, iFriii, sr/(2*iFriii), 1, 0.67, giTSeno
	endif
		a5 gbuzz iAmp/6, iFrV, sr/(2*iFrV), 1, 0.67, giTSeno
	else
		a1 pluck iAmp, iFrec, iFrec, 0, 1
	if iModo==0 then
		a3 pluck iAmp/2, iFrIII, iFrec, 0, 1
```

[41] Csound Document (.csd). Los instrumentos referidos en el texto como instr-1, instr-2 y instr-3 se corresponden con 1, 3 y 4 de este archivo, respectivamente.

```
        else
                a3 pluck iAmp/2, iFriii, iFrec, 0, 1
        endif
                a5 pluck iAmp/2, iFrV, iFrec, 0, 1
        endif
                asal = a1+a3+a5
                asal linen asal, 0.005, p3, 0.005
                out asal
endin

</CsInstruments>
<CsScore>

t 0 120

; i ini dur amp altura tabla modo --> si 0 = mayor, 1 = menor
;---------------------------------
i 1 0 4.0 -4 8.00 4 0 ; C ;
i 1 + . -4 8.05 . 0 ; F
i 1 + . -4 8.07 . 0 ; G
i 1 + . -4 7.09 . 1 ; Am
i 1 + . -4 8.04 . 1 ; Em
i 1 + . -4 8.07 . 0 ; G
i 1 + . -4 8.00 . 0 ; C
;i 1 + . -4 8.03 . 1 ;
e

</CsScore>
</CsoundSynthesizer>
<bsbPanel>
<label>Widgets</label>
<objectName/>
<x>0</x>
<y>0</y>
<width>0</width>
<height>0</height>
<visible>true</visible>
<uuid/>
<bgcolor mode="nobackground">
<r>0</r>
<g>0</g>
<b>0</b>
</bgcolor>
</bsbPanel>
<bsbPresets>
</bsbPresets>
```

batchChroma.m

```
%Method for batching a folder

listing = dir('Data_wav\UMA*.wav');
for i=1:length(listing)

    orden = ['C:\sonic-annotator-0.7-win32\sonic-annotator.exe -t conf.n3
Data_wav\'...
        listing(i).name ' -w csv --csv-force --csv-basedir Chromagram'];
    system(orden)

end
```

writePianoAnnotations.m

```
k = strfind(filename, '.wav');
chromagramName = strcat('Chromagram/',filename(1:k-1), ...
        '_vamp_nnls-chroma_nnls-chroma_chroma.csv');
X = load(chromagramName); % Carga del cromagrama (.csv)

[M,~] = size(X);

GTFileName = strcat('GT/GT_', filename(1:k-1),'.csv');
fid = fopen(GTFileName,'w');
for i = 1:M
    b = num2str(X(i,1)); %conversión de numero a string
    fprintf(fid,'%s,%s\n',b,char(tags(i)));
end
fclose(fid);
```

generarAnotaciónPlantilla.m

```
%SCRIPT: GENERAR ANOTACIÓN PLANTILLA
disp('Ground-truth template generation...'

%Frecuencia de muestreo fs = 44100 Hz
%Incremento de ventana de análisis (step_size) = 2048 muestras

deltat = 0.0464; %deltat = step_size/fs = 2048/44100 = 0.0464
t = 0:deltat:40; %de 0 a 40 segundos
N = length(t);
t=round(t*1000)/1000; %truncado del tiempo a la milésima

%Composición del fichero de texto de salida (.csv)
fid = fopen('anotacionPlantilla.csv','w');
for i = 1:N
    b = num2str(t(i)); %conversión de numero a string
    b = strrep(b, '.', ','); %sustitución de puntos por comas
    fprintf(fid,'%s;%s\n',b,'N');
end
```

```
fclose(fid);
```

GT on off to frame.m

```
%SCRIPT PARA ADAPTAR LAS ANOTACIONES DE 'ISOPHONICS' AL SISTEMA ACE
PROPUESTO

%REFERENCIAS:
%Mauch, M., 2010. "Automatic Chord Transcription from Audio Using
%     Computational Models of Musical Context." Ph.D. diss. Queen Mary
University of London.
%Harte, C., M. Sandler, S. Abdallah, and E. Gómez. 2005. "Symbolic
%     representation of musical chords: A proposed syntax for text
annotations."
%     In Proceedings of the 6th International Society for Music
Information Retrieval Conference
%     (ISMIR), 66-71.

FileList = dir('*.csv');
N = size(FileList,1);

for k = 1:N

   % get the file name:
   filename = FileList(k).name
   disp(filename);



    %%LECTURA DE ANOTACIONES
    data = importdata(filename);
    data = char(data);
    [A,B] = size(data); %A==número de acordes; B==todos los caracteres
incluyendo números, espacios y punto decimal

    GT_column_1 = zeros(A,1);        %correspondiente a t_on
    GT_column_2 = zeros(A,1);        %correspondiente a t_off
    GT_column_3 = repmat({''},A,1);  %correspondiente a nombre_acorde

    %GT_on_off = [GT_column_1 GT_column_2 GT_column_3];

    %%TRATAMIENTO DE LOS DATOS LEÍDOS
    for i=1:A
        cent = 0;
        j = 1;
        pos = 1;
        while(cent == 0) %mientras no se logre segmentar todo el
contenido
            if(isspace(data(i,j)) == 1)
                %detección de los espacios y comprobación de tipo
numérico/alfabético
                %del elemento actual
                if(pos == 1)
                    GT_column_1(i) = str2num(data(i,1:j-1));
                    pos = j + 1;
```

```
                else
                    GT_column_2(i) = str2num(data(i,pos:j-1));
                end
            else
                if(isstrprop(data(i,j),'alpha') == 1)
                    %si el carácter actual es alfabético, comienza el
nombre del
                    %acorde; copiándolo, hemos acabado de segmentar esta
fila
                    GT_column_3(i) = cellstr(data(i,j:B));
                    cent = 1;
                end
            end
            j = j + 1;
        end
    end

    %M filas (número de acordes),
    %N=3 columnas (t_on t_off acorde)

    %%DEFINICIÓN DE LOS NUEVOS VECTORES DE SALIDA (formato t_frame
acorde)
    fs = 44100;
    step_size = 2048;
    deltat = step_size/fs;
    frames = 0.000:deltat:GT_column_2(A); %eje de tiempo discreto
compatible
    frames = frames';
    %con los parámetros del sistema ACE
    M = length(frames);%número de frames que resulta con la resolución de
nuestro sistema
    %para la GT_annotation de entrada
    newChordArray = repmat({'N'},M,1); %inicialización

    %GT_frame -->   (:,1) == frames   (:,2) == newChordArray

    %adaptación de acordes de formato t_on/t_off a formato t_frame:

    pos = 1;
    for i=1:A %hay A acordes en la anotación original
        dif  = GT_column_2(i) - GT_column_1(i); %intervalo entre t_on y
t_off, esto es,
        %cuánto tiempo está sonando el acorde
        n_fr = round(dif/deltat); %número de frames que ocupa en el nuevo
.csv
        newChordArray(pos:(pos + n_fr)) = GT_column_3(i); %copia el
acorde i n_fr veces
        pos = pos + n_fr;
    end
    deltat = round(deltat*1000)/1000;%truncado a la milésima
    newChordArray = newChordArray(1:M); %compensación del desajuste entre
la longitud de
    %los vectores frames y newChordArray

    %%ESCRITURA DEL NUEVO ARCHIVO .CSV
    k = strfind(filename, '.csv');
    newfilename = strcat(filename(1:k-
1),'New',filename(k:length(filename)));
    %nombre_de_la_anotación_original + 'New' + '.csv'
    fid = fopen(newfilename,'w');
```

```
    for i = 1:M
        b = num2str(frames(i)); %conversión de numero a string
        b = strrep(b, '.', ','); %sustitución de puntos por comas
        fprintf(fid,'%s;%s\n',b,char(newChordArray(i)));
    end

    fclose(fid);

end
```

HPseparationBatch.m[42]

```
function HPseparationBatch
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here
FileList = dir('*.wav');
N = size(FileList,1);

for k = 1:N

   % get the file name:
   filename = FileList(k).name
   disp(filename);

       % Author: Derry FitzGerald
    %
    %------------------------------------------------------------------------
------
    % This source code is provided without any warranties as published in
    % DAFX book 2nd edition, copyright Wiley & Sons 2011, available at
    % http://www.dafx.de. It may be used for educational purposes and not
    % for commercial applications without further permission.
    %------------------------------------------------------------------------
------

    %----- user data -----
    WLen            =4096;
    hopsize         =2048;
    lh              =17; % length of the harmonic median filter
    lp              =17; % length of the percussive median filter
    p               =2;
    w1              =hanning(WLen,'periodic');
    w2              =w1;
    hlh             =floor(lh/2)+1;
    th              =2500;
    [DAFx_in_st, FS]=wavread(filename);

    DAFx_in         =(DAFx_in_st(:,1)/2) + (DAFx_in_st(:,2)/2);

    L               = length(DAFx_in);
    DAFx_in         = [zeros(WLen, 1); DAFx_in; ...
       zeros(WLen-mod(L,hopsize),1)] / max(abs(DAFx_in));
    DAFx_out1 = zeros(length(DAFx_in),1);
    DAFx_out2 = zeros(length(DAFx_in),1);
```

[42] Versión modificada del algoritmo de Derry Fitzgerald (2010) de cara al procesado por lotes.

```
    %----- initialisations -----
    grain           = zeros(WLen,1);
    buffer          = zeros(WLen,lh);
    buffercomplex   = zeros(WLen,lh);
    oldperframe     = zeros(WLen,1);
    onall         = [];
    onperc        = [];

    tic
    %UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
    pin  = 0;
    pout = 0;
    pend = length(DAFx_in)-WLen;

    while pin<pend
        grain = DAFx_in(pin+1:pin+WLen).* w1;
    %==========================================
        fc  = fft(fftshift(grain));
        fa  = abs(fc);
        % remove oldest frame from buffers and add
        % current frame to buffers
        buffercomplex(:,1:lh-1)=buffercomplex(:,2:end);
        buffercomplex(:,lh)=fc;
        buffer(:,1:lh-1)=buffer(:,2:end);
        buffer(:,lh)=fa;
        % do median filtering within frame to suppress harmonic
instruments
        Per = medfilt1(buffer(:,hlh),lp);
        % do median filtering on buffer to suppress percussion
instruments
        Har = median(buffer,2);
        % use these Percussion and Harmonic enhanced frames to generate
masks
        maskHar = (Har.^p)./(Har.^p + Per.^p);
        maskPer = (Per.^p)./(Har.^p + Per.^p);
        % apply masks to middle frame in buffer
        % Note: this is the "current" frame from the point of view of the
median
        % filtering
        curframe=buffercomplex(:,hlh);
        perframe=curframe.*maskPer;
        harframe=curframe.*maskHar;
        grain1 = fftshift(real(ifft(perframe))).*w2;
        grain2 = fftshift(real(ifft(harframe))).*w2;
        % onset detection functions
        % difference of frames
        dall=buffer(:,hlh)-buffer(:,hlh-1);
        dperc=abs(perframe)-oldperframe;
        oall=sum(dall>0);
        operc=sum(dperc>0);
        onall = [onall oall];
        onperc = [onperc operc];
        oldperframe=abs(perframe);
    %==========================================
        DAFx_out1(pout+1:pout+WLen) = ...
           DAFx_out1(pout+1:pout+WLen) + grain1;
       DAFx_out2(pout+1:pout+WLen) = ...
           DAFx_out2(pout+1:pout+WLen) + grain2;
        pin  = pin + hopsize;
        pout = pout + hopsize;
    end
```

```
    %UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
    toc
    % process onset detection function to get beats
    [or,oc]=size(onall);
    omin=min(onall);
    % get peaks
    v1 = (onall > [omin, onall(1:(oc-1))]);
    % allow for greater-than-or-equal
    v2 = (onall >= [onall(2:oc), omin]);
    % simple Beat tracking function
    omax = onall .* (onall > th).* v1 .* v2;
    % now do the same for the percussion onset detection function
    % process onset detection function to get beats
    [opr,opc]=size(onperc);
    opmin=min(onperc);
    % get peaks
    p1 = (onperc > [opmin, onperc(1:(opc-1))]);
    % allow for greater-than-or-equal
    p2 = (onperc >= [onperc(2:opc), opmin]);
    % simple Beat tracking function
    opmax = onperc .* (onperc > th).* p1 .* p2;
    %----- listening and saving the output -----
    DAFx_out1 = DAFx_out1((WLen + hopsize*(hlh-1)) ...
            :length(DAFx_out1))/max(abs(DAFx_out1));
    DAFx_out2 = DAFx_out2(WLen + (hopsize*(hlh-1)) ...
            :length(DAFx_out2))/max(abs(DAFx_out2));
    % soundsc(DAFx_out1, FS);
    % soundsc(DAFx_out2, FS);
    newfilename_H = strcat('Harmonic_',filename);
    newfilename_P = strcat('Percussive_',filename);
    wavwrite(DAFx_out1, FS, newfilename_P);
    %wavwrite(DAFx_out2, FS, newfilename_H);


end


end
```