CONVERSOR WAVE A MIDI EN TIEMPO REAL PARA GUITARRAS ELÉCTRICAS

Albert Molina Reverte

Proyecto Final de Carrera

Ingeniería técnica de Telecomunicaciones esp. Imagen y Sonido
Escola Universitaria d'Enginyeria Tècnica Industrial de Terrassa
Universitat Politècnica de Catalunya

8 de Junio de 2006

Tutor: Javier Ruiz Hidalgo
Departamento de Teoria de la Señal i Comunicaciones.

ÍNDICE

CONVERSOR WAV A MIDI EN TIEMPO REAL PARA GUITARRAS ELÉCTRICAS

Capítulo 1

INTRODUCCIÓN

La tecnología musical y su relación con la informática han crecido de forma espectacular durante los últimos 20 años, pasando de grandes estudios de grabación repletos de equipos *hardware* con un coste elevado a programas informáticos de coste más moderado. Con estos programas se pueden igualar o superar los resultados obtenidos con los sistemas *hardware* anteriores.

Estos nuevos sistemas *software*, se valen pese a todo, de algún tipo de controlador *hardware* con el cual podemos introducir notas, melodías o acordes que luego podremos tratar y reproducir en el sistema informático. Gracias al estándar MIDI (*Musical Instruments Digital Interface, www.midi.org)* se pudieron crear este tipo de controladores, bajo la forma de teclados, baterías e incluso guitarras de forma que el músico utilizara su instrumento favorito para introducir las notas al sistema musical, llamado secuenciador.

Las guitarras controladoras MIDI se basaban en un híbrido de guitarra y teclado en el cual las cuerdas se substituían por teclas a lo largo del diapasón, al pulsar la tecla el sistema reconocía directamente la nota tocada. Con la evolución del software musical de los últimos años, es posible utilizar una guitarra convencional como controlador MIDI. El sistema parte de conectar la guitarra vía entrada de línea de una tarjeta de sonido, procesar su sonido para reconocer las notas pulsadas y pasar los datos al secuenciador MIDI. Para esto se necesita de un algoritmo dedicado a reconocer las notas de una guitarra, y no solo una nota sino varias a la vez, hasta seis, que son las que suenan cuando se toca un acorde completo.

En el presente proyecto trataremos de crear un programa que realice la conversión de un archivo en formato WAVE (*WAVEform audio format*) que contenga una grabación de guitarra, en un fichero MIDI con las notas que se han tocado con la guitarra. Todo esto se realizará con un programa que sea capaz de trabajar en tiempo real, de forma que en un futuro se pueda sustituir el fichero WAVE por una guitarra conectada en la tarjeta de sonido y el fichero MIDI sea leído directamente por un secuenciador.

Para realizar el programa, primero se dividió el trabajo en los diferentes bloques con los que se formaría el algoritmo final, primero la lectura del fichero WAVE, segundo la detección de las notas existentes y el procesado de la señal, y tercero la escritura del archivo MIDI.

Una vez decididas las partes en que dividimos el algoritmo, se pasó a estudiar la forma más útil, precisa y optima de realizar cada uno de estos bloques. Primero se realizó un estudio sobre papel de las diferentes opciones de las cuales se disponía a la hora de realizar cada uno de los bloques. Seguidamente se realizaron las pruebas en el programa Matlab (*www.mathworks.com/),* creando en su entorno un programa totalmente funcional que nos serviría de base para el posterior desarrollo del algoritmo en lenguaje C con Microsoft Visual C++ 6.0.

Finalmente, una vez conseguido un programa totalmente funcional se pasó a evaluar los resultados obtenidos, comparando los resultados del entorno Matlab con los de C, y a la vez con los resultados esperados.

Capítulo 2

TEORIA DE LAS SEÑALES MUSICALES

Conocer en detalle como es una señal musical y en particular una señal de guitarra es esencial para la correcta realización del programa.

2.1 Propiedades de las notas musicales

Una señal musical se puede caracterizar por seis variables que son la frecuencia, intensidad, forma de onda, duración, crecimiento, decaimiento y vibración.

- La frecuencia es el número de ondas por segundo.
- La intensidad es la energía por unidad de tiempo.
- La forma de onda es una onda periódica formada por la frecuencia fundamental y sus armónicos.
- La duración es el tiempo que dura el tono.
- El crecimiento y el decaimiento describen la variación de la amplitud del tono, generalmente son substituidos por la nomenclatura de la función ADSR (Attack Decay Sustain Release).
- La vibración de la nota es una modulación en frecuencia que suele ir acompañada de una pequeña modulación de amplitud.

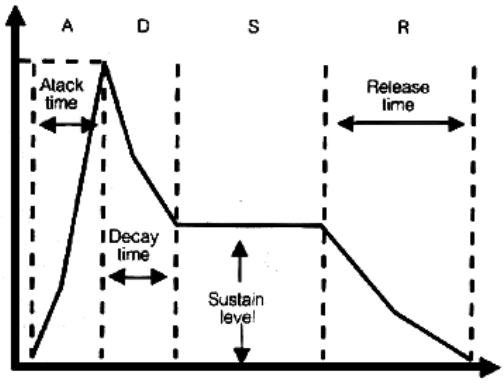

Figura 2.1.- Función ADSR

2.2 Caracterización frecuencial de las notas

Un parte muy importante para la comprensión del algoritmo de detección de pitch y sus problemas derivados es el estudio de los valores de frecuencia asociados a las notas y la relación distancia de frecuencia que se da entre notas.

Definimos el semitono como la distancia entre notas contiguas en el piano, incluyendo las negras. El tono equivale a dos semitonos y una octava esta formada por doce semitonos, los cuales se reparten siguiendo la siguiente regla:

$$F_{n+1}=F_n*2^{(1/12)}$$

Esta relación es obtenida sabiendo que al aumentar una octava, una nota queda multiplicada por dos. Las notas entre ellas dos se divide en doce partes de forma logarítmica, ya que la recepción auditiva del sonido en el cerebro humano sigue pautas logarítmicas.

Si queremos aumentar un semitono se multiplica es nota por dos elevado a 1/12, si por el contrario queremos bajarlo se dividirá por dos elevado a 1/12. El numerador del exponente se corresponde con el numero de semitonos de distancia que queremos calcular.

La organización de notas, con distancia de un semitono entre ellas, en una escala de Sol es la siguiente:

Do Do# Re Re# Mi Fa Fa# Sol Sol# La La# Si

Es de destacar que la distancia entre Mi/Fa y Si/Do es directamente de un semitono.

Como se observará más adelante en la programación, la notación utilizada para nombrar las notas no es la forma clásica, por comodidad y simplificación se utilizará la forma Americana. Esta llama a las notas con una sola letra, en este caso la escala de Sol quedaría de la siguiente forma:

C C# D D# E F F# G G# A A# B

La letra A queda asignada a la nota La, y se van asignando las letras del abecedario consecutivamente hasta Sol que es la G.

2.3 Las notas de guitarra

La guitarra es un instrumento de seis cuerdas formado por una caja de resonancia y un mástil. El mástil tiene en su parte frontal una placa de madera llamada diapasón. El diapasón esta dividido con unas tiras de metal llamadas trastes. Al pulsar una cuerda sin pisar ningún traste, suena lo que llamamos nota al aire. Ordenando las cuerdas de más grave a más aguda, las notas al aire en una afinación estándar son: MI, LA, RE, SOL, SI, MI.

Al pulsar una nota pisando la cuerda en el primer traste obtenemos la nota al aire más un semitono. En el caso de la cuerda más grave estaríamos tocando un FA, ya que de MI a FA existe un semitono. Pulsando la nota pisando en los trastes siguientes, vamos sumando a la nota anterior un semitono, siguiendo con el ejemplo de la cuerda grave en el segundo traste obtendríamos un FA Sostenido o FA#. Podemos llegar a tocar en la cuerda mas grave 5º traste la misma nota LA que suena en la cuerda siguiente, por lo que tocar mas lejos del 5º traste

repite notas que conseguiremos con las cuerdas siguientes.

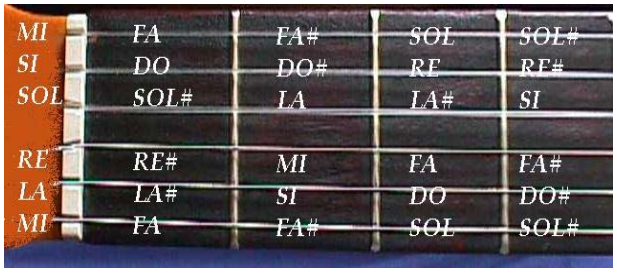

Figura 2.2.- Notas en la guitarra

La señal de una onda de guitarra esta formada por uno o más armónicos, hasta seis, uno por cuerda pulsada. Debido a que cada cuerda tiene unas características distintas: material, grosor y tensión, las variables ADSR dependen de la cuerda pulsada. Generalmente las cuerdas graves tienen un sustain y release muy largos, a medida que la cuerda es más aguda el ataque o crecimiento es mayor pero el sustain y release son mucho menores. Estos parámetros varían bastante teniendo en cuenta, también, la calidad de la guitarra, consiguiendo un sustain y release más largos en cuerdas agudas en las guitarras de buena calidad.

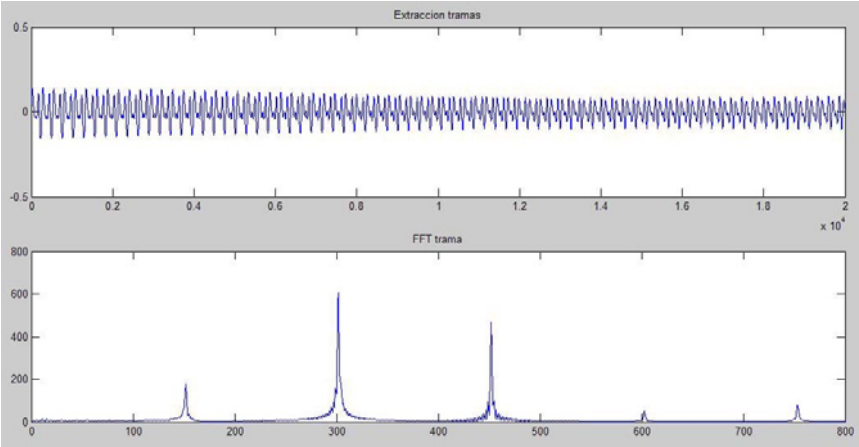

Figura 2.3.- Ejemplo de señal de guitarra, temporal y frecuencial

Como en una guitarra se puede tocar una misma nota en diferentes cuerdas, podemos encontrarnos con una nota que tiene el mismo tono pero a la vez su carácter es diferente, debido a la variación de su envolvente ADSR. Por este motivo, y debido a la sencillez de las notas MIDI, únicamente buscaremos la nota que suena independientemente de la cuerda que se toque, ya que el MIDI no nos permite emular estas pequeñas sutilezas de sonido.

La guitarra, al ser un instrumento de cuerda afinado con unas tuercas situadas en la parte superior del mástil, nunca tendrá la afinación exacta. Por este motivo el algoritmo de detección probablemente no encontrará el valor frecuencial exacto de la nota tocada, este valor se encontrará entre dos notas y tendremos que decidir a que nota está más cercana. Para esto los límites de detección de una nota se tendrán que encontrar en un intervalo de dos valores calculados previamente.

Durante el resto de esta memoria numeraremos las cuerdas de la guitarra asignándoles un número del 1 al 6, correspondiendo el 1 a la más aguda y el 6 a la cuerda más grave (al contrario de lo que podría parecer lógico). Se nombran de abajo hacia arriba.

2.4 Problemas asociados a la guitarra.

A la hora de aplicar un algoritmo de detección de pitch hemos de tener en cuenta los problemas que contrae tener una señal de guitarra.

El problema principal son los ruidos indeseados derivados de la interpretación del músico, como pueden ser golpes o deslizamientos por las cuerdas, sobretodo las graves que están confeccionadas por seda o acero con un hilo de cobre entorchado alrededor de ellas. Este tipo de ruidos pueden dar lugar a detecciones de notas no válidas.

Otro problema es la aparición de armónicos, que es habitual en cualquier sonido. Podemos confundir el segundo armónico derivado de tocar una nota con la pulsación de la cuerda que contiene el tono una octava superior. Pero este problema en el caso de la guitarra se puede agravar. Si la nota que tocamos no es al aire, es decir, pisando un traste, la transmisión de ondas de presión que se hace a la caja de resonancia vía mástil es mayor, aumentando aún más el valor de los armónicos indeseados.

Capítulo 3

ESTUDIO DEL ALGORITMO

En este capítulo analizaremos los diferentes bloques en que dividiremos nuestro programa, analizando las opciones de realización y comentando las ventajas de la opciones comentadas.

La división básica de nuestro algoritmo se realizará de la siguiente forma:

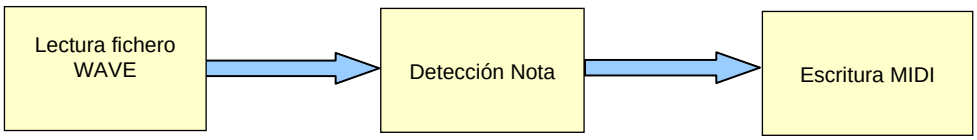

Figura 3.1.-Bloques del sistema WAVE a MIDI

3.1 Sistema de Ficheros WAVE.

Un fichero WAVE o WAV se trata de un contenedor de audio digital, propiedad de Microsoft e IBM utilizado para almacenar sonidos en el ordenador. Se trata de una variante del formato RIFF, método que almacena en paquetes y es muy similar al formato AIFF utilizado por Macintosh. Este formato tiene en cuenta particularidades de las CPU Intel utilizando ordenación little-endian, por lo que es el formato principal utilizado por Windows.

El contenedor WAVE puede soportar cualquier tipo de códec de audio, pero se utiliza fundamentalmente con el formato PCM, que lo hace un especialmente indicado para el tratamiento digital del sonido. El formato PCM no utiliza ningún tipo de compresión a las muestras de sonido, facilitando en gran medida los cálculos sobre la propia señal de audio. Hemos seleccionado la lectura desde un fichero WAVE debido a la similaridad que en un futuro tendrá con la señal proveniente de una entrada de línea de la tarjeta de sonido, evitando tener que modificar en exceso el procesado de señal que realizará nuestro algoritmo.

Los archivos de audio WAVE están formados por una cabecera de 44 bytes y las muestras de audio en formato PCM. Esta cabecera da información sobre los datos que contiene el fichero WAVE tales como su frecuencia de muestreo, número de canales, etc. La cabecera se separa en tres partes llamadas chunks. El primero contiene la palabra RIFF que indica que el fichero

es un archivo WAVE. El segundo contiene la palabra FMT y contiene la información de las muestras de audio, información de compresión, número de canales, Frecuencia de muestreo, Bytes por segundo, Número de bytes por muestra incluyendo los canales y Número de bits por muestra. El tercer chunk que contiene la palabra DATA indica la longitud del fichero WAVE y posteriormente vienen las muestras de audio en sí.

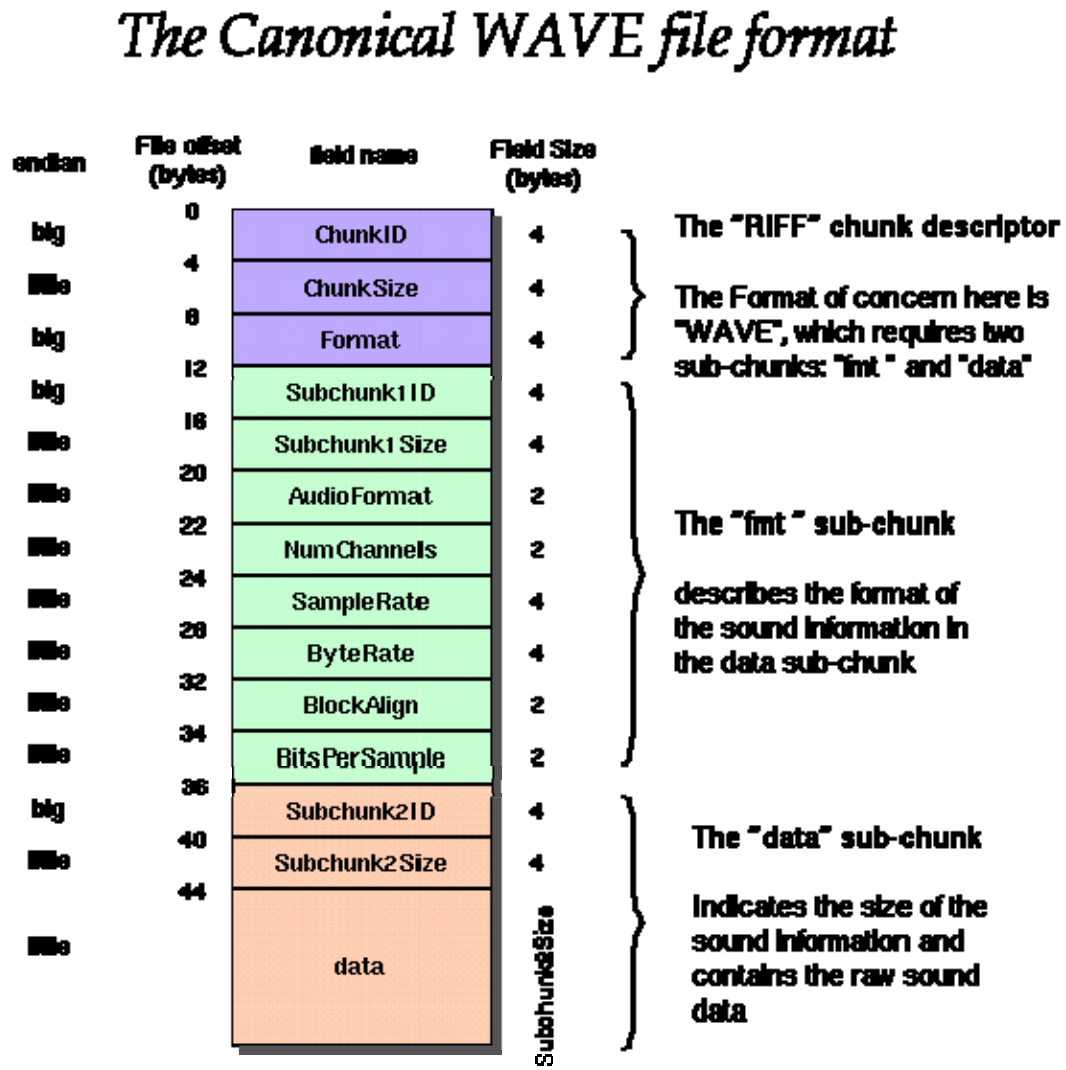

Figura 3.2.- Cabecera de un fichero WAVE.

La forma de almacenar los datos depende también del número de bits por muestra. Si se trata de una señal de muestras de 8 bits, los datos se almacenan como bytes sin signo con un margen dinámico de 0 a 255. En cambio si las muestras son de 16 bits se almacenan como dos bytes con signo con un margen dinámico de -32768 a 32767.

Para realizar el programa hemos definido un tipo de formato de archivo WAVE que se ha de utilizar por defecto, de forma que simplifiquemos en mayor medida el posterior procesado de la señal. Hemos definido el siguiente formato:

AudioFormat: PCM NumChannels: Mono SampleRate: 44100 Hz BitsperSample: 16 bits

Definimos formato PCM por el motivo comentado anteriormente, es el formato más indicado para el procesado de muestras de audio. Utilizaremos un sonido monofónico, la guitarra eléctrica tiene salida monofónica, por lo que es redundante utilizar una señal estéreo. Además, el formato MIDI solo podrá mostrar una única detección de nota.
La frecuencia de muestreo se ha fijado en 44100 Hz, aunque podremos ejecutar el programa con cualquier WAVE que tenga otra frecuencia de muestreo mayor. Por último hemos seleccionado 16 bits por muestra que nos dan una buena resolución de señal y simplifica el procesado de datos, ya que siempre serán dos bytes con signo tipo *short* con un margen dinámico de -32768 a 32767.

Como hemos podido comprobar, el tipo de fichero WAVE es idóneo para nuestros propósitos y debido a su uso extendido, se seleccionó directamente como formato de entrada de las muestras de sonido en nuestro algoritmo.

3.2 Determinación de nota musical.

En este apartado realizaremos un pequeño estudio de los algoritmos de detección de pitch, más comúnmente denominados PDA (*Pitch Detection Algorithm*) existentes evaluando sus ventajas e inconvenientes para su uso con señales de audio digital, y más concretamente de guitarra, para utilizarlo como algoritmo de detección de pitch de nuestro programa.

Primeramente distinguiremos entre procesos en tiempo real y procesos que no son en tiempo real. Una distinción para determinar si un algoritmo es a tiempo real es que este tarde menos en calcular el pitch de un segmento de sonido, que tiempo dure ese segmento. Por tanto nuestro programa ha de ser capaz de calcular el pitch de la señal en menos tiempo que dure el fichero WAVE de entrada.

A su vez nuestro proceso ha de ser *Online*, esto ocurre cuando el resultado se representa en un solo paso sin necesitar una acumulación considerable de datos futuros. En definitiva, el algoritmo PDA seleccionado debe permitir realizar la detección en tiempo real y en línea.

Nuestro algoritmo ha de trabajar en unas frecuencias comprendidas entre los 70 Hz y 700 Hz, que es el rango frecuencial que posee una guitarra acústica. Reduciendo a este intervalo, conseguimos que el algoritmo sea mucho más eficiente. La resolución de nuestro algoritmo ha de ser la mitad de la distancia entre dos semitonos, de esta forma cada tono tendrá un rango entre el cual decidiremos que la nota detectada es ese tono en concreto.

3.2.1 Dificultades en la detección de pitch

En general, cuando hablamos de señales que tienen una frecuencia fundamental, asumimos que hay armónicos presentes. Además, en el momento del ataque de la nota, no solo la fundamental está presente sino casi todo el espectro. Cuando el tono va disminuyendo, la fundamental y sus armónicos se hacen cada vez más claros.

La presencia de armónicos es la razón principal de los errores en la detección de pitch, y usualmente acaban con un error de detección en otra octava. A veces la fundamental puede llegar a ser inferior que el tercer armónico, por ejemplo.

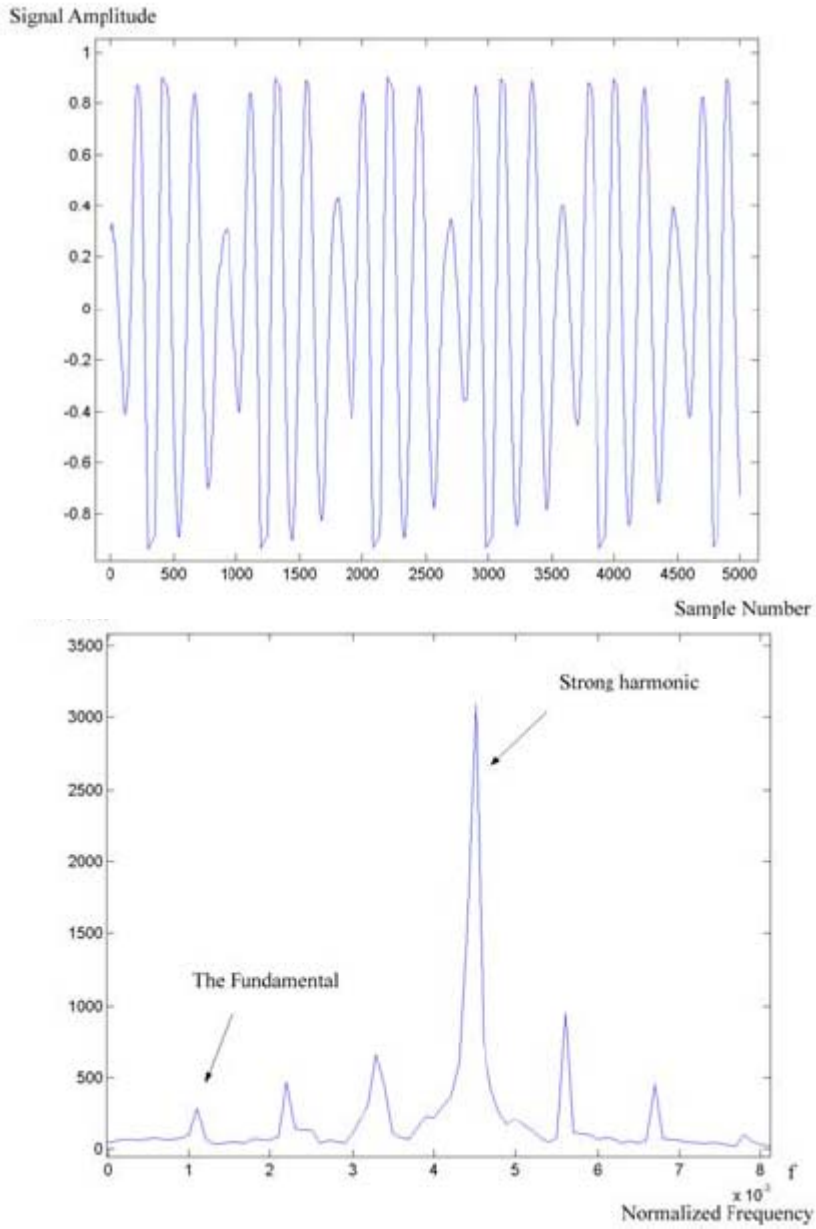

Figura 3.3.- Señal y T.F. Diferencia entre fundamental y armónico.

3.2.2 Evaluación de algoritmos PDA

Los algoritmos que describiremos a continuación se dividen en PDA's de dominio temporal, PDA's de dominio espectral y PDA's combinados.

3.2.2.1 PDA de dominio temporal

Este tipo de algoritmos son vistos como un método bastante antiguo una vez desarrolladas las técnicas de transformación a otros dominios. Las ventajas de este tipo de algoritmos es que no precisan de ningún tipo de transformación, y es muy útil si la carga de post-proceso es elevada.

Sus desventajas aparecen en los casos como los comentados anteriormente, cuando una armónico es elevado y su fundamental es baja. A la vez, si queremos determinar varios armónicos de una señal, el algoritmo se complica y no resultará optimo.
A continuación evaluaremos algunos de los algoritmos PDA en dominio temporal más relevantes:

3.2.2.1.1 Cruce por umbral

Se trata del más elemental extractor de periodicidad, estudia los cambios de polaridad de la señal. Generalmente se conoce como técnica de cruces por cero. Con señales simples como por ejemplo una sinusoide, extrae fácilmente el periodo, que es el tiempo existente entre 2 cruces positivos por cero, definiendo un cruce positivo el paso de polaridad negativa a positiva. Sin embargo, las señales musicales no pueden ser generalizadas con propiedades básicas y el algoritmo, sin duda, fallará.

Una forma de optimizar este algoritmo es calcular el punto de cruce diferente de cero. Calculando correctamente este punto la mejora es evidente. El problema es que las señales musicales son variables en el tiempo y por tanto es imposible predecir el punto óptimo de colocación de este nivel de cruce.

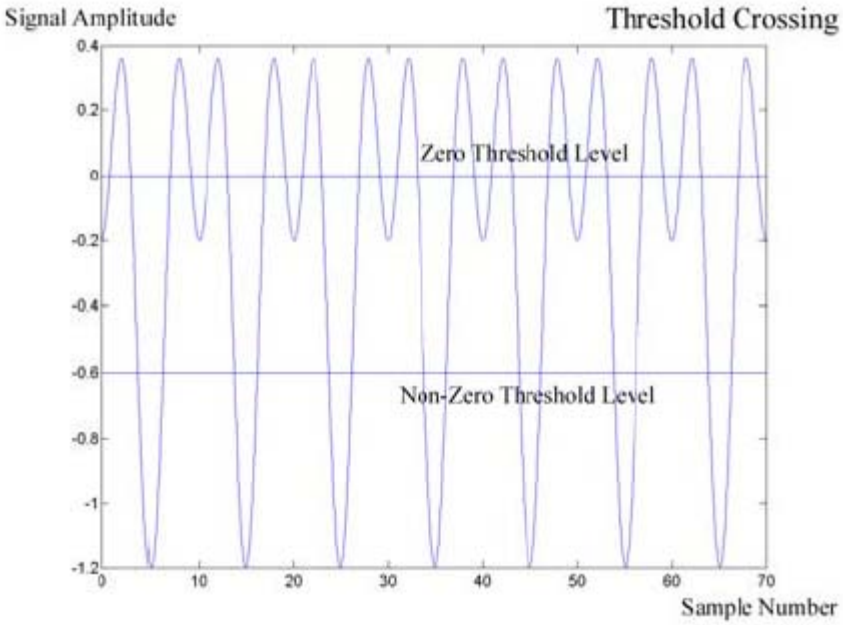

Figura 3.4.- Algoritmo de cruce por umbral

Otra forma de optimizar este algoritmo es con la inclusión de 2 umbrales de cruce, uno positivo y otro negativo. Ambos pueden ser cruzados un número infinito de veces pero solo cuando los dos son cruzados sucesivamente en una secuencia definida, se cuenta el inicio de un nuevo periodo.

3.2.2.1.2 Seguidor de envolvente

Este algoritmo fue desarrollado en 1954 por Ladislav O. Dolanski. Muy utilizado en electrónica analógica, la técnica aún se utiliza el entornos digitales. El periodo del pitch puede ser derivado de la envolvente poniendo marcas cuando la señal excede la envolvente. Alternativamente un detector de picos encontrara el periodo.

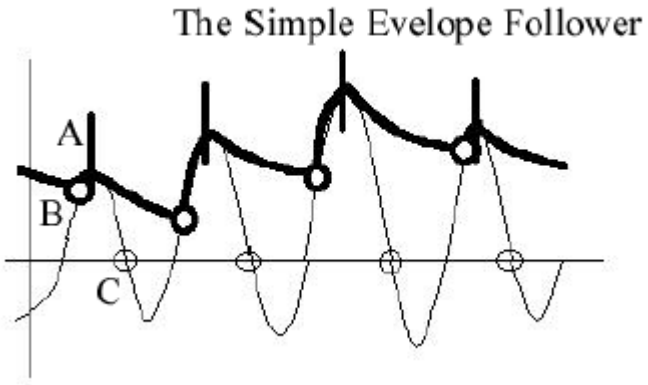

Fig 3.5: Seguidor de envolvente simple

Una mejora del seguidor de envolvente es añadiendo un seguidor detrás de otro, filtrando paso alto con un filtro de primer orden la señal obtenida de cada seguidor.

3.2.2.1.3 Detectores de pico

Existe también una serie de PDA de dominio temporal que dividen la señal en pequeños bloques y analizan los picos existentes en cada bloque. La decisión de la frecuencia se realiza a través de una serie de normas predefinidas que ha de cumplir cada pico. Ejemplos de este tipo de algoritmos son los algoritmos de Reddy, que tiene un rango de detección de 75 a 400 Hz o el algoritmo de Rabiner y Gold.

3.2.2.1.4 Auto correlación

Se trata de un tipo de detectores muy populares. Es un tipo de algoritmo muy utilizado ya que permiten ser aplicados de maneras muy diferentes. Es fácil de modificar para enfocarlo a la aplicación a la cual va destinado, pudiendo variar su rango de detección, resolución y tiempo de respuesta. Pese a ello es un sistema que puede ocasionar errores notorios como fallos de detección de octava.

Para poder utilizar un PDA de auto correlación necesitamos de un preprocesado sofisticado para reducir influencia de formantes y armónicos.

3.2.2.1.5 Otros tipos

Existen muchos otros tipos de algoritmos de algoritmos de detección en dominio temporal, pero la sofisticación de algunos de ellos, hacen más factible una realización en el dominio espectral, ya que sus costes de computación acaban siendo muy elevados.

3.2.2.2 PDA de dominio espectral

Los algoritmos de dominio espectral dependen de una transformación de dominio temporal a espectral, demandando unos altos costes de computación. La FFT (*Fast Fourier Transform*), sin embargo, puede ser aplicada efectivamente en los procesadores actuales. En este tipo de algoritmos podemos diezmar la señal sin perder fiabilidad de detección. Con ellos podemos tener mayor control sobre la situación de la energía de los armónicos, así como la relación entre ellos.

El problema principal de estos algoritmos es que un análisis simple del espectro resultante no es suficiente para determinar el periodo fundamental, por lo que será necesario un análisis posterior como la transformación a Cepstrum o el análisis de los armónicos más destacados.

3.2.2.2.1 Detectores de máximo de FFT

La transformación de los datos a dominio espectral se realiza en un análisis rápido, para esto se utiliza un algoritmo de FFT eficiente en la implementación de la DFT (*Discrete Fourier Transform*).

La forma más sencilla de análisis espectral se realiza simplemente encontrando el pico espectral más grande, suponiendo que este es el fundamental. Para señales que tienen un armónico fundamental elevado y armónicos secundarios pequeños, el resultado es excepcional.

Una forma simple de mejorar este algoritmo es el método de división, después de encontrar el pico en la posición F, se investigan los picos existentes en las posiciones F*n, siendo n un entero. Se considerará a partir de un umbral si estos picos corresponden a la frecuencia fundamental.

Cuando falta el armónico fundamental, se puede realizar un estudio de la distancia entre dos picos adyacentes.

3.2.2.2.2 Compresión espectral y suma de armónicos

Schroeder introdujo en 1968 un algoritmo que requería muchos cálculos, pero aún, uno de los mejores algoritmos disponibles. En esta técnica se estima la frecuencia fundamental a partir de los armónicos superiores, utilizando el hecho que la fundamental es máximo común divisor de los armónicos individuales.

Los picos presentes en el espectro se encuentran con un detector de pico, entonces se construye un histograma con los picos del espectro. Se dividen las frecuencias por 2 y se añaden al histograma, el mismo proceso se realiza con factores de división 3, 4, etc. Como resultado aumenta el coste de computación cuantos más valores utilicemos. Este proceso resultará en un máximo en el histograma en la frecuencia fundamental.

Otra mejora de este algoritmo es el PDA de producto o suma de armónicos. Se suman las magnitudes del espectro en frecuencias equidistantes. La frecuencia que sale magnificada de la suma es entonces la fundamental.

3.2.2.2.3 Otros tipos de algoritmos en dominio espectral.

Existen otros muchos tipos de algoritmos de detección de pitch en dominio espectral. Uno de ellos es el algoritmo transformación de Q constante, muy útil para análisis de aspectos más musicales de la señal como glissandos, vibratos o instrumentos afinados en escalas diferentes de la temperada, esto es debido a una mayor resolución en la zona de las altas frecuencias, resultante de este tipo de transformación.

Otros análisis como el Cepstrum o el Vocoder de Fase están más enfocados a señales de voz y sus formantes, utilizando técnicas que no serán útiles para nuestro algoritmo de detección.

3.2.2.3 PDA Híbrido.

Bajo esta denominación se agrupan los algoritmos que utilizan las características tanto del dominio temporal como del espectral. El más utilizado es el detector con banco de filtros.

3.2.2.2.1 Detector simple con banco de filtros

Son los que utilizan un banco de filtros paso banda espaciados geométricamente para dividir la señal en octavas. Cada salida del banco de filtros se une a un detector simple y el resultado se escoge del filtro que tenga más energía.

3.3 Sistema MIDI

Antes de la aparición del sistema MIDI a principios de los años ochenta, los músicos no podían conectar entre sí sintetizadores y controladores de diferentes marcas, por cada sonido nuevo que necesitaban, precisaban de un nuevo equipo. Los avances a pasos de gigante que iba realizando la electrónica y microinformática estaba convirtiendo a los sintetizadores en ordenadores especializados, substituyendo los antiguos controles por tensión por el control digital.

Estos hechos propiciaron que varios fabricantes, entre ellos Korg, Kawai, Roland, Requencial Circuits y Yamaha, se pusieran de acuerdo para conseguir los siguientes propósitos:

1. Hacer que sus máquinas intercambiaran información musical.
2. Ofrecer más expresividad a los músicos.

En el año 1983 la IMA (*International MIDI Association)* publica un conjunto de directivas que son el nacimiento de la norma MIDI.

El MIDI es un lenguaje que se manifiesta digitalmente sobre los cables que constituyen el soporte físico por el que viaja. Posteriormente, con la evolución del software y de los ordenadores personales como equipo MIDI, se desarrolló el estándar de archivo MIDI, que podía ser almacenado y reproducido por cualquier equipo conectado al ordenador. La unidad básica de la información son los bytes que se corresponden con diferentes señales que se envían en serie de un dispositivo a otro, y que transportan las acciones que se desea ejecutar. El dispositivo receptor, o Esclavo, se controlado mediante la transmisión de la información MIDI adecuada, por el dispositivo emisor de la información, o Maestro.

Los instrumentos MIDI pueden ser conectados en cascada, a través el conector MIDI THRU, que replica y regenera la señal de un dispositivo a otro. Cada equipo distingue la información dirigida a él con la existencia de canales MIDI. A cada equipo se le asigna un canal MIDI diferente. También se puede conectar los equipos en estrella con un HUB MIDI, aumentando así el número de equipos conectados al sistema.

La norma MIDI se refiere a los siguientes aspectos:

- Tipo de los mensajes intercambiados.
- Conectores que deben tener los equipos.
- Cables que los unen.
- Los canales MIDI.
- Manera de transmitir los mensajes.

Los parámetros más importantes que definen una nota según el estándar MIDI son los siguientes:

- Frecuencia: Cada nota tiene asignada una frecuencia fundamental, que las distingue entre ellas.
- Velocidad: Define como de fuerte se toca la nota MIDI, generalmente incide sobre la amplitud del sonido o a su volumen.
- Duración: Tiempo que permanece sonando la nota, esta finalización se puede producir por relajación o detención brusca.

Los aparatos MIDI se pueden clasificar en tres grandes categorías:

- Controladores: generan los mensajes MIDI (activación o desactivación de una nota, variaciones de tono, etc.). El controlador más típico existente tiene forma de teclado de piano, aunque pueden adoptar otras formas (generalmente, instrumentos musicales).

- Unidades generadoras de sonido: también conocidas como módulos de sonidos, reciben los mensajes MIDI y los transforman en señales sonoras (recordemos que MIDI no transmite audio, sino datos de eventos).

- Secuenciadores: no son más que aparatos destinados a grabar, reproducir o editar mensajes MIDI. Los puede haber en formato de hardware, como software de computadora, o incorporado en un sintetizador.

3.3.1 Mensajes MIDI

Los mensajes MIDI pueden transmitir información musical o de mando hasta los equipos MIDI. Los mensajes musicales contienen las notas en sí, con los parámetros anteriormente explicados, los mensajes de mando o control configuran los equipos conectados. Estos mensajes están formados por bits agrupados en bytes. Todos empiezan por un byte de estado llamado *status byte*, un grupo de ocho bits que contiene el número correspondiente a una acción musical, como puede ser tocar una nota. A continuación vienen más bytes de datos con números que dan información complementaria.

Para crear un sistema compatible con MIDI debemos conocer cual es la metodología de funcionamiento de estos sistemas. Al enviar un mensaje de nota MIDI sucede lo siguiente:

1.- Se aprieta la nota, genera mensaje MIDI de NOTE ON:

- Mensaje que indica que se ha tocado una nota, numero de canal MIDI.
- Información del pitch de la nota
- Información de su velocidad

2.- Se suelta la nota, se genera mensaje NOTE OFF:

- Mensaje que indica que una nota ha finalizado, número de canal MIDI
- Información de nota que finaliza.
- Información velocidad de relajación, rapidez con que se suelta la tecla.

3.3.2. Ficheros MIDI

Desde la aparición de programas informáticos, llamados secuenciadores, capaces de gestionar las conexiones MIDI y enviar los datos a los equipos correspondientes, se abrió la posibilidad de almacenar los datos MIDI en un fichero. De esta manera, los datos MIDI podían ser recuperados y reproducidos a través de cualquier equipo. Por tanto, para ejecutar el fichero MIDI necesitaremos un programa que haga de puente entre los datos MIDI y el sintetizador interno de nuestra tarjeta de sonido. En nuestro caso utilizaremos Cubasis VST v5.0, versión recortada del famoso software Cubase de Steinberg (*www.steinberg.net*).

Los parámetros necesarios para crear un fichero MIDI son los mismos que los comentados anteriormente, inicio de nota, nota, velocidad, duración y final de nota, pero ahora junto una cabecera que nos permita reconocer el formato MIDI y su estructura. La transmisión de estos parámetros se realiza en 8 bits y son muy similares al sistema de transmisión directa de datos por cables.

En un fichero MIDI el inicio de nota viene representado por el valor 0x90 y el final por el valor 0x80, representados en hexadecimal ASCII. Los valores de las notas están cuantificados en un rango de 0 a 127.

El parámetro de velocidad de la nota interpretada se cuantifica con valores de 0 a 127. Ordenados de menor a mayor intensidad. La duración se cuantifica con los mismos parámetros equivaliendo 0 a la mínima duración y 127 a la máxima duración que se permita.

Octava	C	C#	D	D#	E	F	F#	G	G#	A	A#	B
0	0	1	2	3	4	5	6	7	8	9	10	11
1	12	13	14	15	16	17	18	19	20	21	22	23
2	24	25	26	27	28	29	30	31	32	33	34	35
3	36	37	38	39	40	41	42	43	44	45	46	47
4	48	49	50	51	52	53	54	55	56	57	58	59
5	60	61	62	63	64	65	66	67	68	69	70	71
6	72	73	74	75	76	77	78	79	80	81	82	83
7	84	85	86	87	88	89	90	91	92	93	94	95
8	96	97	98	99	100	101	102	103	104	105	106	107
9	108	109	110	111	112	113	114	115	116	117	118	119
10	120	121	122	123	124	125	126	127				

Tabla 3.1.- correspondencia notas con número MIDI.

3.3.2.1 Organización de un fichero MIDI

Para poder crear un fichero MIDI debemos saber cuál es su estructura definida y los parámetros necesarios para transmitir toda la información. A continuación explicaremos la estructura de estos ficheros.

Los ficheros MIDI contienen una o más cadenas MIDI, con información para cada uno de los eventos. Secuencias de canciones, estructura de pistas, tempo, etc. son todos soportados. El fichero contiene los datos en formato binario de 8 bits y para la transmisión eficiente se convierten a formato hex ASCII.

Byte estado	Descripción
1000cccc	Desactivación de nota
1001cccc	Activación de nota
1010cccc	Postpulsación polifónica
1011cccc	Cambio de control
1100cccc	Cambio de programa
1101cccc	Postpulsación monofónica de canal
1110cccc	Pitch
11110000	Mensaje exclusivo del fabricante
11110001	Mensaje de trama temporal
11110010	Puntero posición de canción
11110011	Selección de canción
11110100	*Indefinido*
11110101	*Indefinido*
11110110	Requerimiento de entonación
11110111	Fin de mensaje exclusivo
11111000	Reloj de temporización
11111001	*Indefinido*
11111010	Inicio
11111011	Continuación
11111100	Parada
11111101	*Indefinido*
11111110	Espera activa
11111111	Reseteo del sistema

Mensajes MIDI en formato binario

Los ficheros MIDI se organizan en secuencias, cada secuencia se divide en paquetes.
Estos paquetes constan de 4 caracteres identificando el tipo de paquete, vienen seguidos de 4 bytes formando un mensaje de 32 bits que es el número de bytes en el paquete. A continuación vemos un ejemplo de estructura de cabecera de paquete:

```
struct CHUNK_HEADER
{
  char          ID[4];
  unsigned long  Length;
};
```

Tenemos dos tipos de paquetes:

- Header chunk
- Track chunk

El paquete *Header chunk* contiene los datos de la cabecera del fichero MIDI y el *Track chunk* una cadena secuencial de datos MIDI sobre alguno de los 16 canales MIDI disponibles. Un fichero MIDI siempre empieza con un paquete de cabecera y uno o varios paquetes de pista, organizándose de la siguiente forma:

MThd <length of header data>
<header data>
MTrk <length of track data>
<track data>
MTrk <length of track data>
<track data>
...

3.3.2.2 Header chunk

El header chunk es el paquete de inicio de un fichero MIDI, contiene simplemente los caracteres 'M' 'T' h' 'd' escritos en su valor hexadecimal y la longitud de los datos en esta cabecera, un ejemplo de cabecera es el siguiente:

Formato de los datos:

```
struct MTHD_CHUNK

{
  /* Cabecera de 8 bytes*/
  char          ID[4];  /*  Espacio para 'M','T','h','d' */
  unsigned long  Length; /* Será de 6 bytes, contados desde este punto*/

  /* 6 bytes */
  unsigned short Format;
  unsigned short NumTracks;
  unsigned short Division;
};
```

Valor de los datos:

4D 54 68 64 *MThd ID*

00 00 00 06 *Longitud de MThd es siempre 6.*
00 01 *Tipo de formato 1.*
00 02 *Hay 2 Mtrks.*
E7 28 *Cada incremento de delta-time es un milisegundo.*

Nótese que los primeros 4 bytes corresponden con los carácteres ASCII ‘M’ ’T’ ’h’ ’d’, y luego se indica una longitud de 6 bytes, después de los cuales podemos ir a buscar la siguiente cabecera.

3.3.2.3 Track chunk

El track chunk es donde se almacenan los datos de las melodías, se trata de una cadena de eventos MIDI precedida por unos valores denominados delta-time. La estructura de un Mtrk es la siguiente:

```
struct MTRK_CHUNK

{
  /* Here's the 8 byte header that all chunks must have */
  char          ID[4];   /* This will be 'M','T','r','k' */
  unsigned long  Length;  /* This will be the actual size of Data[] */

  /* Here are the data bytes */
  unsigned char  Data[];  /* Its actual size is Data[Length] */
};
```

<track data> = <MTrk event>+...

<MTrk event> = <delta-time> <event>

El delta time representa la cantidad de tiempo que transcurre antes de otro evento, por ejemplo, si suceden simultáneamente, el valor de delta time es cero. Un evento esta compuesto por los siguientes mensajes:

<event> = <MIDI event> | <sysex event> | <meta-event>

Donde *MIDI event* representa un evento MIDI, como la pulsación de una nota. *Sysex event* envía mensajes exclusivos del sistema MIDI, como un cambio de programa. *Meta event* especifica información no MIDI utilizada por los secuenciadores, como puede ser el nombre de la pista.

3.3.2.4 Escritura de notas MIDI

Para escribir un evento MIDI que represente una pulsación de nota escribiremos lo siguiente:

<delta-time> <0x90> <pitch> <velocidad>

Siendo 0x90 el identificador de MIDI NOTE ON.

Para un mensaje de fin de nota haremos lo siguiente:

<delta-time> <0x80> <pitch> <velocidad>

Siendo 0x80 el identificador de MIDI NOTE OFF.

3.3.2.4 Finalización de fichero MIDI

Para finalizar el fichero MIDI hemos de indicar que se ha terminado la información de pista y esta información se transmite con el parámetro *meta-event* de tipo *end of track*:

FF 2F 00

Una vez introducidos todos estos datos creamos un fichero MIDI, usualmente caracterizado con la extensión *.mid, que es posible reproducir en cualquier aplicación musical que soporte formato MIDI. Luego podremos asignar a las notas introducidas cualquier tipo de sonido existente en el sintetizador interno de nuestra tarjeta de sonido.

Como ejemplo de fichero MIDI completo veamos el siguiente:

4D 54 68 64	***MThd***
00 00 00 06	*Length of the MThd chunk is always 6.*
00 01	*Format 1.*
00 02	*2 MTrk chunks*
E7 28	*Each increment of delta-time represents a millisecond.*
4D 54 72 6B	***MTrk***
00 00 00 14	*chunk length (20)*
00	*delta time*
FF 58 04 04 02 18 08	*Key signature*
00	*delta time*
FF 51 03 07 A1 20	*tempo*
00	*delta time*
FF 2F 00	*end of track*
4D 54 72 6B	***MTrk***
00 00 00 XX	*chunk length (XX) se calcula al final*
00	*delta time*
90 24 40	*inicio nota midi C2*
86 7F	*delta time*
80 24 30	*fin nota midi C2*
00	*delta time*
FF 2F 00	*end of track*

Se trata de un fichero MIDI que contiene 2 *Mtrk*. El primero de ellos configura el tempo y *key signature* en el secuenciador MIDI. El segundo *Mtrk* escribe una nota MIDI de valor C2. La duración de la nota viene especificada por la diferencia entre los *delta-time* (escrito en formato *variable-length*). Los *delta time* que escribamos posteriormente vendrán referenciados al ultimo *delta-time*, es decir, el delta time del *End of Track* (de valor 00) sucede a distancia 00 del *delta time* de fin de nota (de valor 86 7F).

Capítulo 4

IMPLEMENTACIÓN DEL ALGORITMO

Una vez realizado el estudio de las partes en que se divide nuestro algoritmo, hemos de decidir cuál es la forma más óptima de realizar el programa, de forma que el resultado sea una buena detección de las notas de guitarra a tiempo real.

La estrategia de programación se dividió en dos etapas. La primera sería la realización de un programa de pruebas en entorno Matlab. La programación con Matlab es relativamente más sencilla que otros programas de más bajo nivel, nos ofrece la sencillez de visualización de gráficas de los datos que se van obteniendo, además el proceso de apertura de ficheros WAVE ya está implementada. De esta manera en el entorno Matlab únicamente nos preocupamos de diseñar el algoritmo PDA de manera eficiente y funcional, y crear un fichero MIDI que posteriormente nos permitiera comparar los datos obtenidos en Matlab con los obtenidos en C.

Una vez con un programa Matlab totalmente funcional, se pasará el programa a un lenguaje de programación como C a través del programa Microsoft Visual C++ v 6.0. El lenguaje C nos permitirá desarrollar la aplicación en tiempo real, optimizando en gran medida el código que genera automáticamente Matlab. Dada la mayor dificultad para poder visualizar datos y gráficos de C, tener un programa equivalente Matlab en el que mejorar el programa, simplificó mucho la programación del sistema conversor WAVE a MIDI. Cualquier modificación necesaria se diseñaba primero en Matlab y una vez funcionaba se pasaba al entorno C simplemente traduciendo el código escrito en Matlab.

El programa obtenido en C tendrá extensión .exe, que se ejecutará en entorno Windows a través de la consola de comandos de MS-DOS. El programa carece de interfaz gráfica ya que el proyecto se ha enfocado al diseño de un algoritmo eficiente y funcional capaz de cumplir con las características predefinidas. Este generará automáticamente y en el mismo directorio raíz donde se encuentra el ejecutable, un fichero MIDI (*out.mid*) con la detección de notas.

4.1 Programación en Matlab

Como ya hemos comentado anteriormente, realizar un programa previo en Matlab nos ayudará a desarrollar un algoritmo que funcione correctamente, en el cual podamos realizar las correcciones necesarias de forma rápida y sencilla. El algoritmo se ha dividido en las 3 partes comentadas en el capítulo anterior: lectura de fichero WAVE, aplicación de un algoritmo de detección de pitch y paso a un fichero MIDI.

4.1.1 Lectura de fichero wave

Matlab nos proporciona una lectura de ficheros WAVE muy sencilla, con la función *wavread:*

[x,fm,nbits]=wavread('../Sound/E2.wav');

En X tenemos un vector con las muestras del fichero WAVE, en *fm* almacena su frecuencia de muestreo y en *nbits* el valor de bits por muestra. En la función indicamos la ruta del fichero WAVE que queremos convertir a MIDI, que ha de ser un WAVE mono de 16 bits, como se comentó en el capítulo anterior.

Para analizar el sonido con el algoritmo de detección de pitch, tendremos que introducir los datos del WAVE en tramas, de forma que cada trama tenga una longitud idónea. Esta longitud ha de permitir tener una cantidad de señal, suficiente para realizar la FFT con una buena resolución, y a la vez que solo esté sonando una sola nota o acorde de guitarra. Si la longitud fuera demasiado larga, en una misma trama podrían solaparse dos notas consecutivas, con lo que en la detección MIDI escribiríamos dos notas al mismo tiempo.

Para realizar estos cálculos hemos seleccionado una longitud de trama de 20000 muestras, con un solapamiento de 10000, es decir, cada muestra se analiza 2 veces, obteniendo así mayor resolución de detección.

Pese a no haber fijado una frecuencia de muestro estricta, recomendaremos una frecuencia de 44100 Hz, dando como resultado que cada trama contiene 0.45 segundos de señal. Con 0,45 segundos tenemos un valor que permitirá reconocer notas que se toquen a un ritmo de negra con una velocidad de 120 BPM (*Beats per minute* o golpes por minuto). Estos valores de trama se pueden disminuir, o hacer seleccionables, de manera que obtengamos mayor resolución temporal de las notas detectadas.

4.1.2 Algoritmo de detección de pitch

Tras el análisis detallado de las varias opciones de algoritmos PDA, se decidió aplicar un algoritmo de detección espectral mediante un FFT.

Los algoritmos en dominio temporal tienen una desventaja que nos hizo excluirlos desde un principio, su dificultad de programación para la detección de hasta 6 notas simultáneas. Aumentar el número de notas que puede detectar a la vez, precisa de un algoritmo que se adapte a la señal de entrada en tiempo real. Calculando hasta 6 umbrales unidos al ruido que se puede generar en la guitarra, haría de la detección poco fiable y del algoritmo poco eficiente, siendo más rentable una detección en dominio frecuencial.

Los algoritmos PDA en dominio frecuencial permiten ver y detectar varias frecuencias fundamentales a la vez de manera más sencilla. Aplicados con uno de los algoritmos FFT más eficientes de la actualidad, y dada la capacidad de los procesadores actuales, podremos realizar estos cálculos en tiempo real sin grandes problemas.

La Fm del fichero WAVE ha de ser mayor que el número de puntos de la FFT seleccionada. Como fijamos la Fm en 44100 Hz, seleccionaremos un número de puntos de FFT de 40000.

El problema que nos presentará la FFT será la distinción entre fundamentales reales y armónicos derivados de las demás fundamentales, como puede ser este ejemplo:

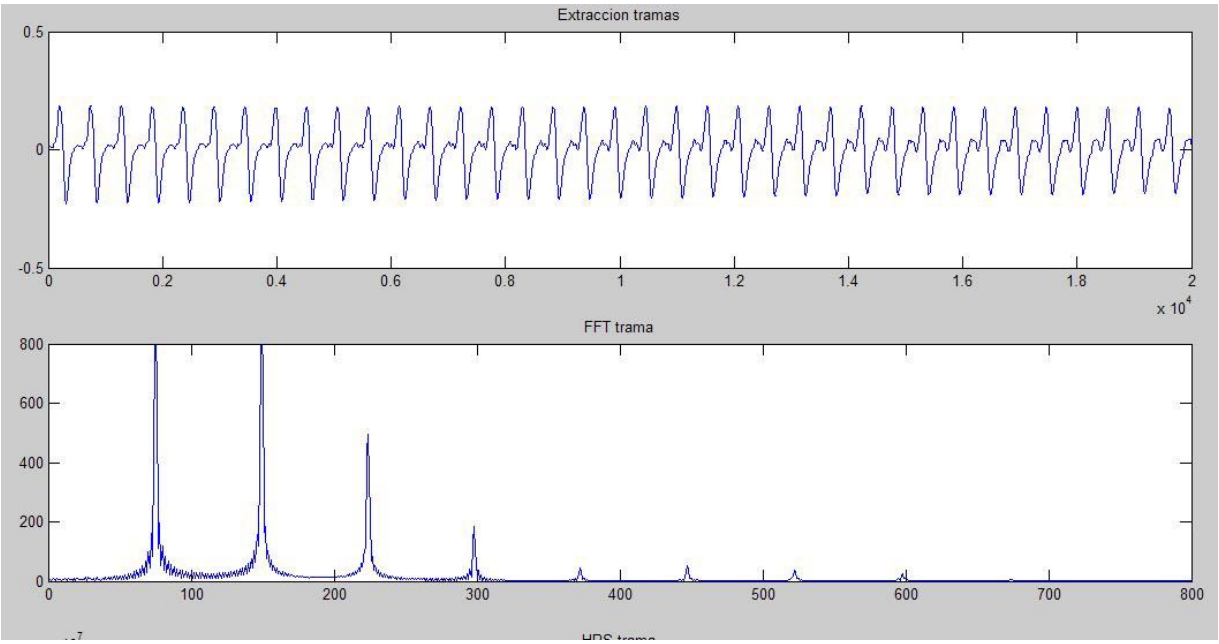

Figura 4.1.- Señal de fm=44100 Hz y módulo de su FFT de 40000 ptos.

En este caso tenemos el transitorio de tocar una sola nota E1. Tenemos la fundamental en la muestra 80 correspondiente a la frecuencia analógica de E1 y el segundo armónico correspondiente a una nota E2 virtual en la muestra 150. Resulta difícil en este caso distinguir si realmente hemos tocado solo la nota E1 o hemos tocado una combinación de la nota E1 y E2 a la vez con nuestra guitarra.

Para solventar este problema de la mejor forma posible utilizaremos un post-procesado de la señal transformada siguiendo las pautas del algoritmo PDA de compresión espectral y suma de armónicos. Llamaremos a este método HPS (*Harmonic Product Spectrum*).

4.1.2.1 HPS

EL algoritmo HPS es un algoritmo diseñado para minimizar la influencia de los armónicos en la detección de las frecuencias fundamentales.

Aprovecha la propiedad de que los armónicos de una fundamental F se encuentran a distancia F*n de ella, siendo n un numero entero cualquiera. Si diezmamos la señal espectral por un factor de 2 y la multiplicamos al espectro inicial, el valor de la fundamental se sumará con la de su primer armónico, reforzando así su amplitud. Todo nivel de señal que no sea armónico o fundamental quedará multiplicado por un valor pequeño, dejándola a un nivel imperceptible comparado con la fundamental. Si volvemos a diezmar, esta vez por un factor 3, los 3º armónicos se multiplicarán a la fundamental, aumentando otra vez su nivel.

Veamos el esquema de funcionamiento del algoritmo HPS:

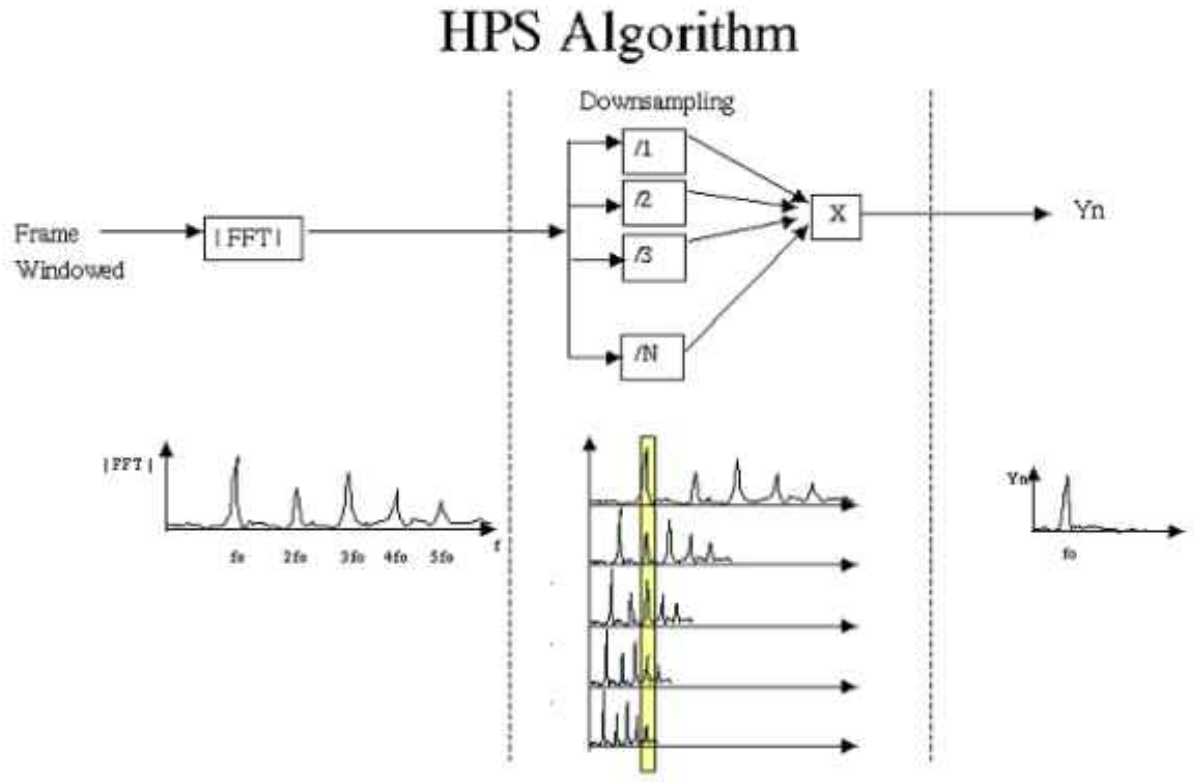

Figura 4.2.- esquema Harmonic Product Espectrum

Como podemos observar el resultado mejora bastante la detección de la fundamental. Hemos de tener en cuenta, sin embargo, que demasiados niveles de diezmado ralentizarán la velocidad de ejecución del algoritmo. Después de varias pruebas decidimos que con 2 niveles de diezmado de señal (por factores 2 y 3), nuestro algoritmo detectaba correctamente la mayoría de notas sin comprometer demasiado al resto. Solo en casos puntuales y sobretodo en el ataque (o transitorio) de una nota, el algoritmo detectaba un armónico como fundamental. Intentar solventar este problema ponía en compromiso las detecciones de notas en otro tipo de casos.

El algoritmo HPS desarrollado en Matlab es el siguiente:

```
function hps2 = hps(spectrum,NFFT)

spec = spectrum;
%----------1r NIVEL HPS----------
hps1(1)=0;
for i=2:NFFT/4
   hps1(i)= spec(i-1)*spec((2*i)-1) + spec(i)*spec(2*i) + spec(i+1)*spec((2*i)+1);
   %promediamos con las muestras más próximas
end
%----------2º NIVEL HPS----------
hps2(1)=0;
for i=2:NFFT/4
     hps2(i)= hps1(i-1)*spec((3*i)-1) + hps1(i)*spec(3*i) + hps1(i+1)*spec((3*i)+1);
end
```

Como podemos observar se trata de un algoritmo sencillo y muy rápido de ejecutar. Como parámetros de entrada tenemos el espectro y el número de puntos de la FFT (definido como 40000) del espectro de la FFT de entrada solo se utiliza la mitad, ya que el resto está duplicado. En el primer nivel multiplicamos cada nota por su equivalente en la posición doble quedando multiplicada la fundamental por su 2º armónico. Esta es la operación equivalente a diezmar la señal por un factor 2 y multiplicar por la original. Además promediamos con las muestras más próximas, ya que el armónico no siempre estará exactamente en el doble de la frecuencia fundamental.

En el segundo nivel realizamos la misma acción, esta vez multiplicando por el 3^r armónico de la señal y promediando con las muestras más próximas.

Aplicando este algoritmo conseguimos mejorar el resultado de las detecciones en gran medida, pongamos como ejemplo la gráfica obtenida para una nota MI en varias octavas:

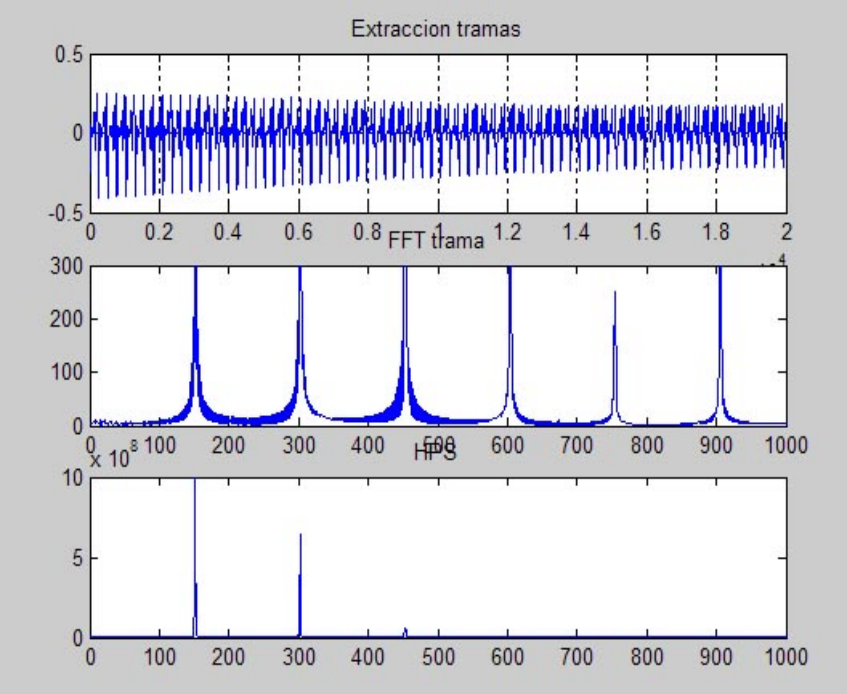

Figura 4.3. Señal, módulo de su FFT y procesado HPS

Ejemplo de E2 justo en el momento de ataque (zona transitoria) tocada en la 4º cuerda pisando en el segundo traste. Aparece claramente la fundamental a en la muestra 150, equivalente a una frecuencia analógica de 164 Hz, y el 2º y 3r armónico que podremos discriminar aplicando un umbral a partir del cual decidiremos que no es una nota válida. Estos umbrales se aplicarán estudiando los niveles de una nota real de la misma zona, que siempre tendrá más nivel. La mejora respecto a la FFT simple (gráfica del centro) es evidente.

En la gráfica inferior la señal esta en fase de *sustain*, la detección se hace mucho mejor que en el ejemplo anterior.

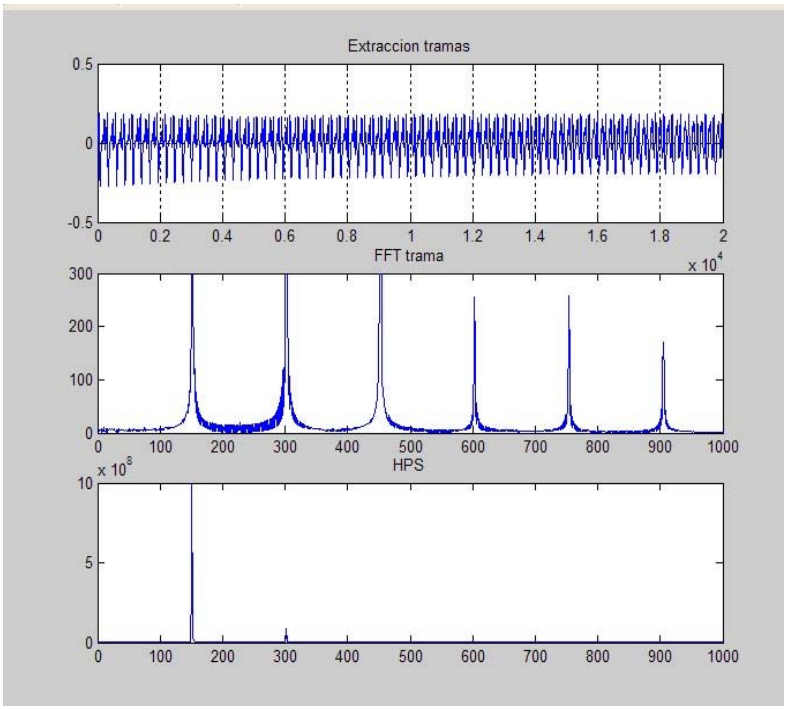

Figura 4.4.- Señal, módulo de su FFT y HPS

4.1.2.2 Decisión de la nota

Una vez realizada la FFT y el post-procesado con el algoritmo HPS, debemos programar otra función que encuentre los máximos del HPS y sepa aplicarle el nombre de la nota correspondiente.

Lo que haremos será dividir el espectro en los rangos de muestras de la FFT que abarca cada cuerda. El rango abarca hasta el 5º traste (recordemos que a partir de él la nota se puede tocar en la siguiente cuerda más aguda), excepto en la primera cuerda, la más aguda, que tendrá un rango mucho mayor, que marcaremos hasta la muestra 390 en la nota B3. Realmente solo importa la nota que se toca y no la cuerda que la toca.

```
S1 = M(60:98);
S2 = M(99:132);
S3 = M(133:176);
S4 = M(177:222);
S5 = M(223:297);
S6 = M(298:390);
```

Figura 4.3: división en 6 secciones.

Seguidamente buscamos el máximo de cada división de espectro, y si este supera un umbral preestablecido le asignamos la nota cuyo valor real sea el más próximo:

```
Mx1 = max(S1);  %buscamos el máximo de la zona de cuerda 1
    if (Mx1 <= th(1))
       %si no supera el umbral la trama no tiene notas
       Nts1 = X;
    else
       %hay notas, paso a decidir que nota es
       val_1a = find(S1==Mx1);
       val_1 = val_1a + offset(1);

       if (val_1 > 72.56 & val_1 < 77.09)
          Nts1 = E1;
          matx(i,4) = 40;
          matx(i,1) = j;
          i=i+1;
       elseif  (val_1 > 77.1 & val_1 < 81.63)
           Nts1 = F1;
           matx(i,4) = 41;
           matx(i,1) = j;
           i=i+1;
        elseif  (val_1 > 81.64 & val_1 < 86.16)
          Nts1 = FH1;
           matx(i,4) = 42;
           matx(i,1) = j;
           i=i+1;
       elseif  (val_1 > 86.17 & val_1 < 90.70)
          Nts1 = G1;
          matx(i,4) = 43;
          matx(i,1) = j;
          i=i+1;
       elseif  (val_1 > 90.71 & val_1 < 96.14)
          Nts1 = GH1;
          matx(i,4) = 44;
          matx(i,1) = j;
          i=i+1;
       else
          Nts1 = 3; %codigo de ERROR
      end
    end
```

Como vemos una vez encontrado el máximo, buscamos la nota a la cual está más cercano a partir del rango de los *else if* sucesivos. Si está en el rango, se le asigna el nombre de la nota (hemos substituido el signo de sostenido # por la letra H) y se rellena la matriz *matx* que posteriormente se convertirá en el fichero MIDI.

4.1.3 Fichero MIDI

Para convertir los valores que hemos detectado en las diferentes secciones a MIDI nos hemos valido de una librería Matlab dedicada a gestión de datos MIDI. La librería se llama *MIDI TOOLBOX (Matlab tools for music research),* es de descarga gratuita y uso libre (*www.jyu.fi/musica/miditoolbox/).*

Esta librería nos proporciona una función llamada *writemidi* cuyo parámetro de entrada es una matriz con los datos MIDI obtenidos. Esta matriz ha de ser de 7 columnas, cada correspondiente a un valor MIDI, en las filas colocaremos los datos MIDI necesarios. Una vez hecha la matriz, con la llamada a la función *writemidi,* los datos pasan a formar parte de un Standard MIDI File con extensión .mid. Los valores de cada columna son los siguientes:

1.- ONSET (Beats)

2.- DURATION (Beats)

3.- MIDI channel

4.- PITCH

5- VELOCITY

6.- ONSET(sec)

7.- DURATION (sec)

El código escrito para llenar esta matriz es el siguiente:

```
matmid = zeros (Ntrames*6, 7);  %creo la matriz para pasar a MIDI file.
matmid(:,2) = 1.0; %establezco duracion de 1 en todas las notas que detecte.
matmid(:,3) = 1.0; %establezco canal midi 1 a todas las notas.
matmid(:,5) = 95;  %establezco velocidad de 95 a todas las notas.
```

Como podemos observar, creamos una matriz de Ntrames*6 filas, de forma que en el caso extremo de tener 6 notas en cada trama, la matriz tenga tamaño suficiente. Además establecemos unos parámetros fijos para todas las notas, como son la duración, canal y velocidad. Consideraremos que una nota detectada dura todo el intervalo de la trama.

Dentro de la detección llenamos la nota y su posición con el índice j, que aumenta en cada trama nueva.

```
matx(i,4) = 44;
matx(i,1) = j;
```

Veamos ahora un ejemplo de archivo MIDI generado por nuestro programa Matlab, abierto con el programa Cubasis VST.

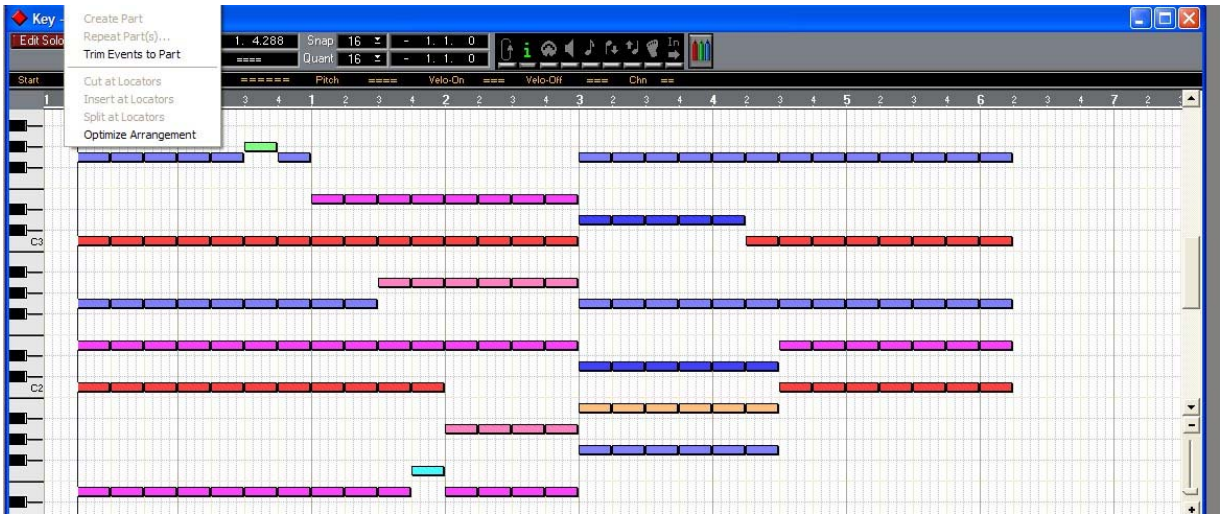

Figura 4.4.- Fichero MIDI obtenido con Matlab.

A la izquierda aparece un teclado de referencia, de forma que identifiquemos las notas que están escritas. Podemos ver una progresión de acordes Do- Fa- La menor – Sol muy similar a la tocada con la guitarra. Aparece algún error de detección, sobretodo en el cambio a Fa, debido a que se tocó una forma de acorde poco convencional.

4.1.4 Evaluación del programa Matlab

Con el entorno Matlab hemos podido conseguir un programa que realiza las funciones deseadas, habiendo comprobado la utilidad de nuestro algoritmo HPS y la exactitud de la detección de nota, excepto algún error puntual. Pese a esto el programa no cumple las expectativas de tiempo real, ya que tarda una media de 3.5 segundos de cálculo por 1 segundo de señal de audio.

Este problema, como comprobaremos más adelante, no se debe al algoritmo en si, sino a la implementación del propio programa Matlab.

4.2 Programación en Visual C++

Habiendo obtenido en el apartado 4.1 un programa conversor de WAVE a MIDI totalmente funcional, pasamos ahora a traducirlo a un lenguaje de más bajo nivel. Así podremos conseguir un programa ejecutable y que funcione a tiempo real, es decir, tarde menos de 1 segundo en calcular 1 segundo se señal de audio.

4.2.1 Consideraciones previas

Antes de pasar nuestro algoritmo a lenguaje C, hay que considerar las diferencias entre los dos lenguajes utilizados:

1. Debemos tener en cuenta que la apretura de un fichero WAVE en C no se puede realizar como en Matlab, ya que no existe, a priori, ninguna función para ello. Crearemos a partir de la función *fopen* un algoritmo de apertura de un fichero WAVE, dejando los datos en un vector que luego podamos recorrer sin ningún tipo de problemas.

2. En Matlab no nos hemos preocupado por el tipo de los datos utilizados. En cambio en C tendremos que utilizar el tipo de dato más idóneo para cada parte del programa, realizando el mínimo número de conversiones de tipo posibles.

3. Los vectores Matlab empiezan por la posición 1, en cambio la primera posición de C es la 0.

4. Hay que desarrollar o buscar una serie de librerías MIDI alternativas, dedicadas en exclusiva a C.

5. Hay que buscar un algoritmo de FFT más eficiente que el incluido en las librerías de C.

4.2.2 Lectura de fichero WAVE

La lectura del fichero WAVE la realizaremos mediante la función *fopen*. Abriremos el fichero que especifiquemos en el comando de ejecución del programa (argv[1]) en modo binario (rb).

```
fp = fopen(argv[1], "rb");
```

Fp es un puntero a un archivo, almacena la posición donde se encuentra nuestro archivo de sonido. A partir de aquí, y si *fp* no tiene valor nulo, es decir no existe el archivo, vamos leyendo sucesivamente la cabecera de datos del fichero para comprobar que cumple con las características fijadas anteriormente. La lectura se realiza con la función *fread:*

```
fread((void *)id, 4, 1, fp);
```

Tras la lectura, el puntero *fp* pasa directamente a la siguiente posición del fichero.

Con la función *strncmp*, comparamos el valor esperado de la cabecera con el obtenido en *fread*, de forma que comprobemos la integridad del fichero WAVE.

```
if ( strncmp( id, "RIFF", 4)!=0 )


        {
        fprintf(stderr, "ERROR: EL fichero WAV no es válido (no se encontró RIFF).\n");
        fclose(fp); // Cerrar el fichero WAV
        exit(-1); // Salir del programa
        }
```

Este proceso se realiza en cada campo. Además los datos que nos serán útiles en un futuro se almacenan en una serie de variables dedicadas a ello.

```
int Ncanales; // Número de canales (Mono=0, Estéreo=1, etc.)
int Fm; // Frecuencia de muestreo (SampleRate)
int Br; // Byte Rate
int Nbytes_bloque; // Bytes por bloque (muestras incluyendo todos los canales)
int Nbits_muestra; // Número de bits por muestra de sonido (16)
int i; // variable para guardar campos de 4 bytes
short s; //variable para guardar campos de 2 bytes
```

Una vez leída la cabecera del fichero WAVE, de acuerdo a lo expuesto en el capítulo 3, almacenamos el campo de datos en un vector llamado *vecwave*. Este contiene las muestras tipo short del fichero WAVE de 16 bits y un solo canal. Como su tamaño puede variar dependiendo de la duración del fichero, nos vemos obligados a utilizar memoria dinámica para su almacenamiento, con la variable *Tamany* obtenida de la cabecera. Esto se realiza de la siguiente forma:

```
short *vecwave; //creamos puntero a shorts que contienen las muestas del WAVE en un vector

vecwave = (short *) malloc (Tamany * sizeof (short));
```

Ya creado el vector, leemos sucesivamente las muestras del fichero WAVE, cerrando al final el puntero al archivo:

```
for (k=0; k<Nbloques; k++)
                {
                        // siempre usaremos 16 bits

                        fread((void*)&s, 2, 1, fp); // En s tenemos la muestra de sonido
                                                    [-32768,32767] signed short
                        vecwave[k]=(s); //almaceno el valor de la muestra en el vector
                }
fclose(fp);
```

4.2.3 Procesado del sonido

El procesado de sonido se realiza de forma muy similar a Matlab, teniendo en cuenta las diferencias entre ambos entornos y comentadas anteriormente. Extraemos una a una las tramas del vector de sonido, procesándolas de la siguiente forma:

```
for (index=0; index<Ntrames; index++)  //bucle ejecutado en cada trama para analizar la señal
{
                printf("Trama numero : %d\n",index);
                get_trama(vecwave,index,desp_trama,long_trama,data);
                absfft (data,mfft);
                division(mfft,mfft2);
                hps(mfft2,out_hps);
                deteccion (out_hps, Fm, vec_fin);
                printf("Resultado:%d%d%d%d%d%d\n\n",
                vec_fin[0],vec_fin[1],vec_fin[2],vec_fin[3],vec_fin[4],vec_fin[5]);
 }
```

La función get_trama extrae un vector *data* con la trama actual que estamos procesando, olvidándonos por ahora del resto de los datos. Seguidamente realizamos la FFT con la función *absfft*. Este algoritmo de FFT se ha escogido por ser de uso sencillo y cálculo rápido, ayudándonos a conseguir nuestro objetivo de tiempo real. El número de puntos utilizados para realizar esta FFT es de 32768, ya que el algoritmo requiere una potencia de 2 como valor.

Posteriormente se realiza una división de las amplitudes de la fft obtenida. Este paso se debe a que de Matlab asigna valores de amplitud al fichero WAVE de -1 a 1, siendo 1 la amplitud máxima. En cambio, como hemos visto, en C abrimos el fichero WAVE de 16 bits en un short con un rango dinámico de -32768,32767.

Al aplicar la FFT sus máximos son los mismos multiplicados por 32767, por lo que los umbrales de detección calculados en Matlab no serán válidos para este nuevo algoritmo en C. La manera más sencilla de solucionar esto, antes que recalcular los umbrales, es equiparar la FFT de C con la de Matlab mediante esta sencilla división.

Ahora, con la señal mfft2 podemos ya calcular el Harmonic Product Spectrum, y limpiar la señal eliminando los armónicos indeseados:

```
void hps (double *y, //mfft de entrada
          double *z) //hps de salida
{
        int i;
        double h[LEN_FFT/4]; //vector del 1r nivel hps
        //Primer nivel del hps
        h[0] = 0;
        for (i=1;i<(LEN_FFT/4);i++)
        {
                h[i] = (y[i-1]*y[2*(i-1)])+(y[i]*y[2*i])+(y[i+1]*y[2*(i+1)]);
        }

        //Segundo nivel del algoritmo hps
        z[0]=0;
        for (i=1;i<(LEN_FFT/4);i++)
        {
                z[i] = (h[i-1]*y[3*(i-1)])+(h[i]*y[3*i])+(h[i+1]*y[3*(i+1)]);
        }
}
```

Una vez realizado el algoritmo HPS, detectamos de manera análoga a Matlab los armónicos de la señal resultante. Gracias a la división los umbrales de detección se pueden mantener igual. Esta vez, sin embargo, hemos aplicado a la nota detectada el número de nota MIDI correspondiente. Así evitamos otra tabla de conversión de nombre de nota a número MIDI.

```
max_vector (s1,18, &mx1, &pos); //máximo de la sección 1
        if (mx1 <= th[0]) //no hay ningún máximo que supere el umbral, no hay nota
        {
        nts1 = 1;
        }
        else
        {
                val1 = pos + offset[0];

                if ((val1 > (79*a)) & (val1 < (85*a)))
                {
                        nts1 = 40; //Ponemos el valor de la nota midi p.e. 40 == E1
                }
```

La función *max_vector* se ha programado a mano, para que responda a las expectativas de detección esperadas, por contra de lo que sucede a veces con la función *max* de Matlab.

Una vez detectado el máximo de cada una de las 6 secciones, se almacenan los valores en un vector llamado *vec_fin*, que tiene 6 posiciones.

4.2.4 Escritura MIDI

Respecto a la incorporación de la escritura de ficheros MIDI, se ha tenido que cambiar por completo la metodología de trabajo respecto a Matlab, puesto que la librería utilizada en Matlab no tiene versión en C. La búsqueda de otras librerías MIDI dedicadas para C resultó infructuosa, pues la mayoría requerían un pago de derechos de autor por su uso. Finalmente se optó por escribir un fichero siguiendo la especificación SMF (Standard MIDI File). Tras varias búsquedas por la red, se consiguió recopilar un buen número de funciones en C útiles para escribir un archivo MIDI (*www.harmony-central.com/MIDI/files.html*). Finalmente la tarea fue unificar estas funciones (a nivel de nombres, comentarios, etc.) y modificarlas para adaptarlas a nuestras necesidades.

Dividiremos estas funciones en tres tipos:

1.- Genéricas, que nos permitirán escribir en hexadecimal cualquier dato que queramos introducir en el fichero MIDI. También convertir los datos a longitud variable, que es un tipo de dato que se utiliza en los ficheros MIDI, generalmente en el delta-time:

```
void WriteVarLen(ofstream &os, unsigned long value); %esribe valores en longitud  variable
void WriteUnsignedLong(ofstream &os, unsigned long ul); %escribe una variable unsigned long
void WriteUnsigned24BitValue(ofstream &os, unsigned long ul); %escribe valor de 24 bits
void WriteUnsignedShort(ofstream &os, unsigned short us); %escribe short sin signo
void WriteUnsignedChar(ofstream &os, unsigned char ub); %escribe caracteres
```

2.- Escritura directa de datos, que permiten crear cabeceras directamente, o introducir meta-datos como nombres de secuencia, Tempo, o el final de MTRK.

```
inline void WriteDeltaTime(ofstream &os, unsigned long value); %escribe delta-time en vector salida
void WriteChunkType(ofstream &os, char c1, char c2, char c3, char c4); %escribe tipo de chunk
void WriteHeaderChunk(ofstream &os); %Escribe cabecera
void WriteSequenceOrTrackName(ofstream &os, string n); %Escribe nombre de pista
void WriteTimeSignature(ofstream &os);%escribe la signatura
void WriteSetTempo(ofstream &os); %Ajusta Tempo
void WriteKeySignature(ofstream &os);
void WriteEndOfTrack(ofstream &os); %escribe el final de track
void WriteTempoTrackChunk(ofstream &os);
void WriteProgramChange(ofstream &os, int channel, int pc); %escribe un cambio de programa
```

3.- Escritura de notas, en estas funciones se basó todo el esfuerzo de modificación para permitir ajustarlas a la escritura de hasta 6 notas simultáneas, pudiendo variar el número de 0 notas a 6 a la vez. Las funciones son:

```
void WriteNoteOn(ofstream &os, int channel, int pitch, int velocity);
void WriteNoteOff(ofstream &os, int channel, int pitch);
void WriteNote(ofstream &os, int note, int note1,int note2,int note3,int note4,int note5); %escribe 6 notas a la vez
```

El algoritmo de escritura de una nota es el siguiente:

```
void WriteNote(ofstream &os, int note, int note1,int note2,int note3,int note4,int note5)
{
        //NOTE ON
        if ( note >= 10)
        {
        WriteDeltaTime(os, 0);
        WriteNoteOn(os, 2, note, 96);
        }
        else{}
        //NOTE OFF
        if (note >= 10)
        {
        WriteDeltaTime(os, 480);
        WriteNoteOff(os, 2, note);
        }
        else{}
}
```

Si la nota detectada tenia un valor MIDI mayor que 10 (de esta manera diferenciamos los valores 1, 2, y 3 que corresponden a errores de detección o inexistencia de nota) se escribe la nota, primero escribiendo un delta time a 0 y NOTE ON en el canal 2 y a una velocidad común de 96. En caso que haya error o ninguna nota el programa no escribe nada. Después se añade el mensaje NOTE OFF, muy similar al mensaje de NOTE ON.

Todos estos valores se van escribiendo secuencialmente en un vector *ofstream* llamado *ofs*. Se trata de un tipo C++. El funcionamiento es muy similar al procedimiento *fopen*, necesitando también cerrar el vector *ofs* al final de la escritura. Su inicialización y cierre se realiza de la siguiente forma:

```
ofstream ofs("out.mid");
WriteHeaderChunk(ofs);
WriteEndOfTrack(ofs);
ofs.close();
```

El resultado obtenido de la conversión de un fichero WAVE a MIDI con este procedimiento es el siguiente:

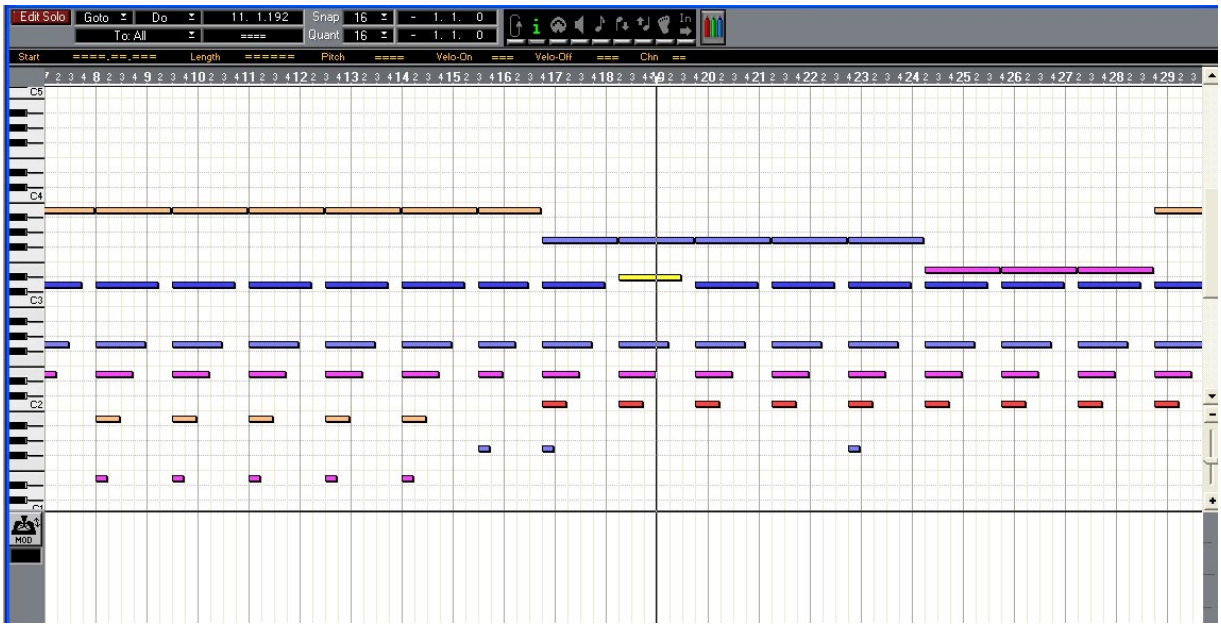

Figura 4.5.- Notas MIDI obtenidas en C.

Observamos una progresión de notas parecidas al ejemplo del archivo MIDI generado en Matlab, también podemos encontrar algún error de detección en las mismas condiciones que Matlab.

4.2.5 Evaluación del resultado del algoritmo en C

Con la programación en C hemos podido cumplir el objetivo propuesto, conseguir la misma funcionalidad que el programa en Matlab pero, esta vez si, en tiempo real. Según los cálculos realizados con la ejecución del fichero E2.wav de 38 segundos de duración, el programa tarda 1 segundo aproximadamente en realizar todos los cálculos y generar el archivo MIDI. Podemos decir que el programa dedica 35 ms de cálculo por cada segundo de señal de sonido, que siguiendo el criterio comentado en capítulos anteriores hacen del programa un algoritmo a tiempo real.

4.2.5.1 Ejemplos de detecciones

Para poner a prueba el programa hemos realizado varios sonidos que convertiremos a MIDI. Tres de estos sonido son las tres notas MI existentes en la guitarra: E1, E2, E3.

1.- E1.wav: Contiene la grabación de la cuerda más grave tocada al aire. En la detección observamos las notas E1 y 3 notas E2. En este caso el algoritmo confunde algún armónico con una fundamental, sucede sobretodo en la fase transitoria de la señal de guitarra.

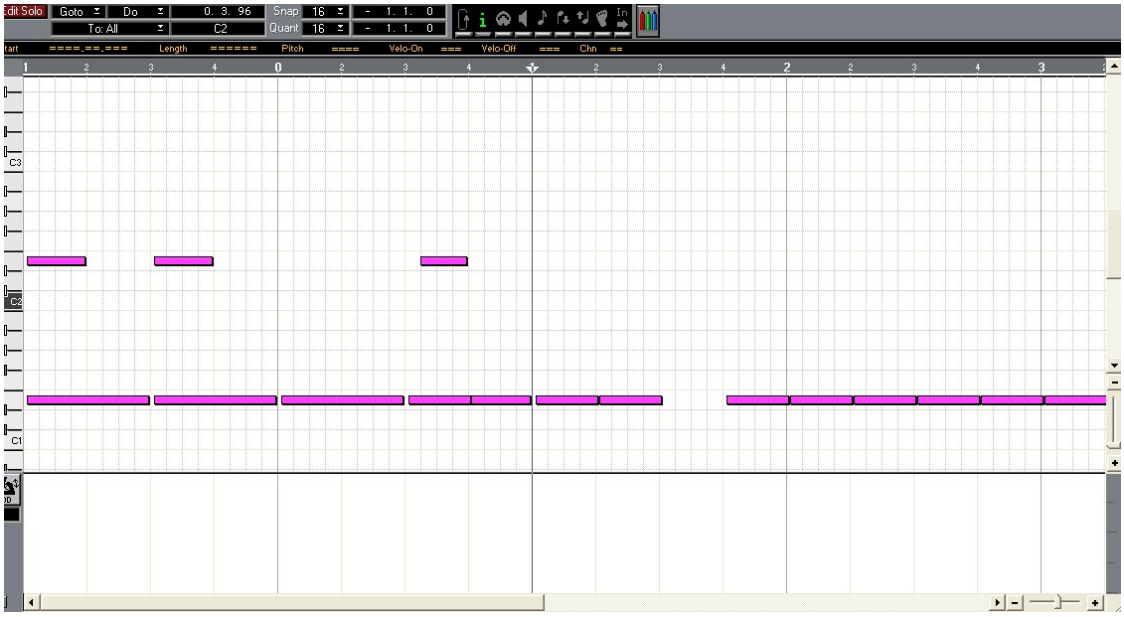

Figura 4.6.- MIDI de E1

2.- E2.wav: Contiene la grabación de la nota E2 tocada en la 4ª cuerda, 2º traste. El caso es muy similar al anterior, esta vez con la detección de una nota D3 ocasionada por la fricción de la cuerda con el traste (sonido metálico).

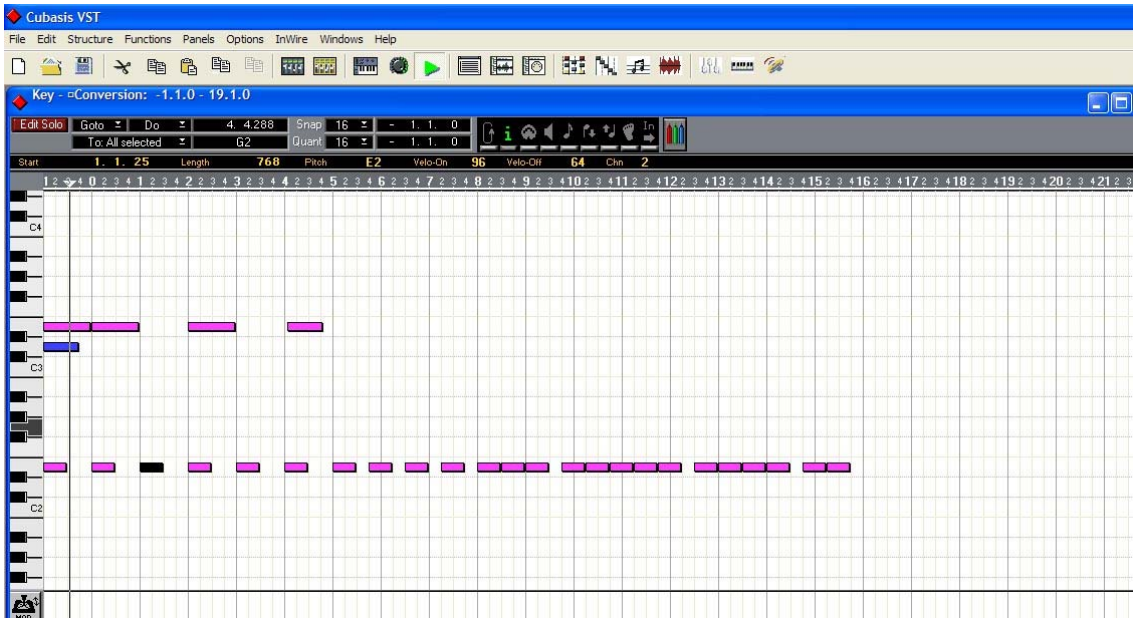

Figura 4.7.- MIDI de E2

2.- E3.wav: Contiene la grabación de la nota E3 tocada en la 1ª cuerda. Aquí solo observamos la cuerda tocada originalmente.

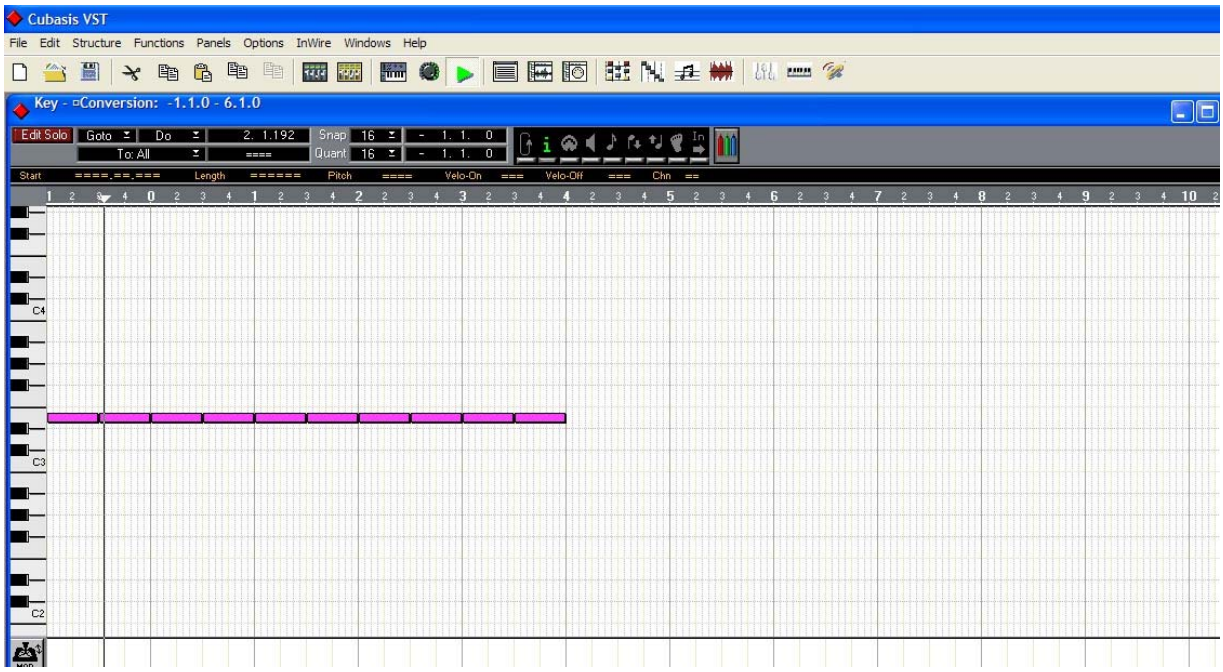

Figura 4.8.- MIDI de E3

El hecho de detectar armónicos es debido a la situación de los umbrales de detección. Durante el estudio de estos, nos encontramos con que el armónico a veces superaba, en 1 o 2 tramas de la fase transitoria, a una fundamental resultante de tocar solo la cuerda original. Pese a esto, si no existiera el algoritmo HPS, el resultado seria mucho peor.

4.3 Comparación de resultados en Matlab y C.

Realizaremos ahora una comparación de los resultados obtenidos en los programas Matlab y C. Estos han de ser muy similares, ya que el algoritmo utilizado es el mismo. Las diferencias observadas son 2 principalmente:

1.- Fichero MIDI diferente. Se debe a una diferente implementación de las librerías MIDI y sobretodo de los delta-times. En Matlab la longitud de notas es igual para todas, en C las notas graves asignan menos duración que a las agudas. La rutina C trata de forma diferente los delta-times, pero las notas detectadas son las mismas.

2.-Pequeñas diferencias en la detección. Se deben a algunos cambios en rutinas traducidas de Matlab a C, como la búsqueda del máximo.

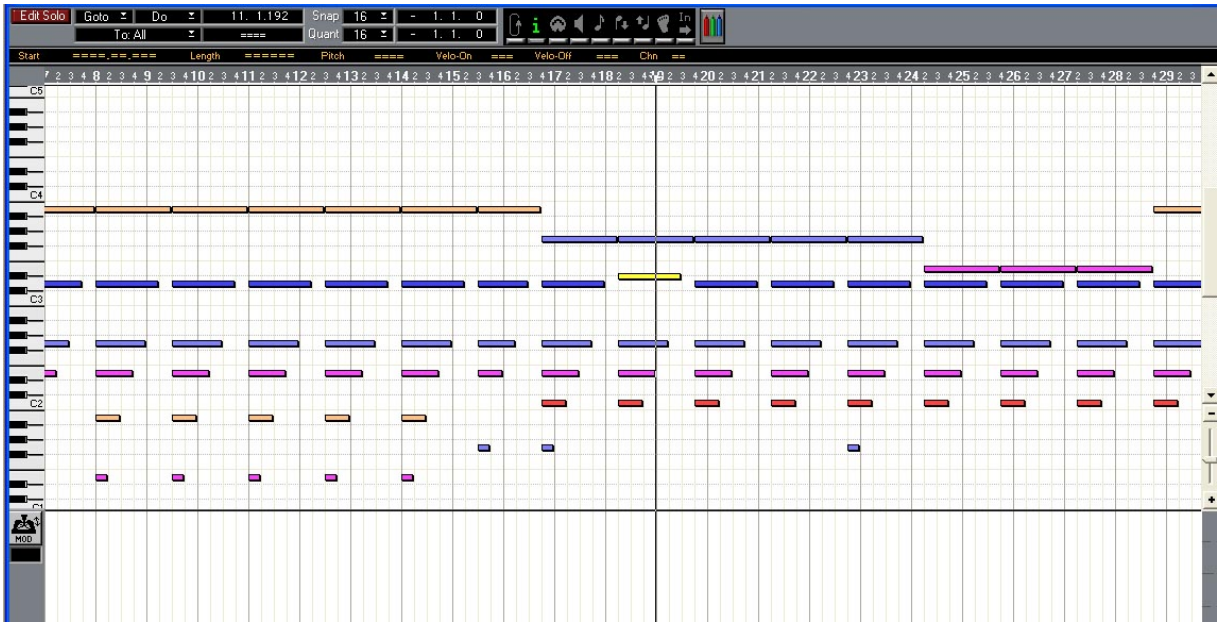

Figura 4.6.- Detección con C

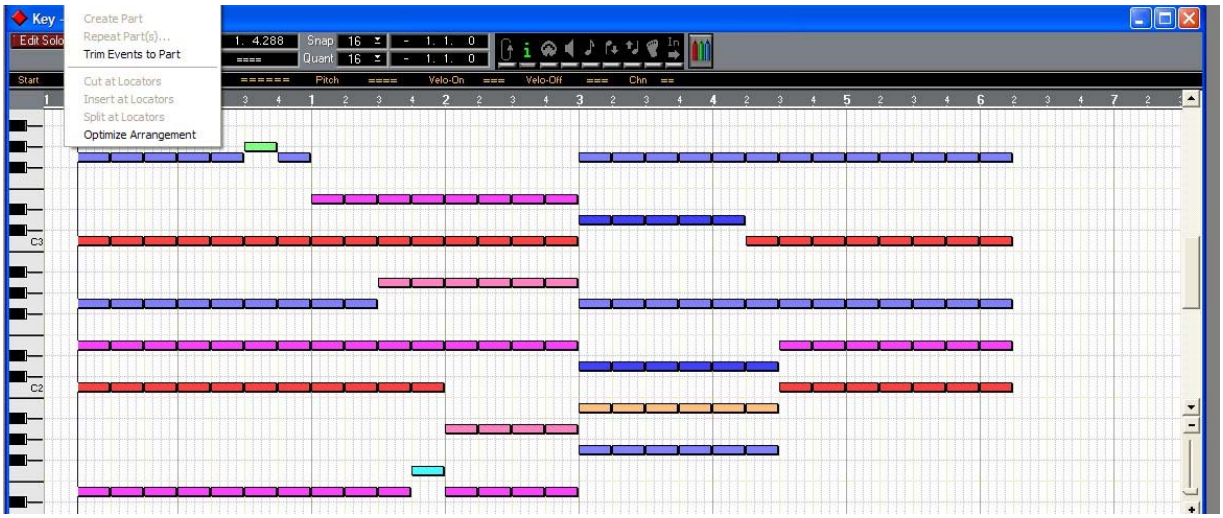

Figura 4.6.- Detección con Matlab

Capítulo 5

CONCLUSIONES

En el presente proyecto hemos realizado un conversor de ficheros WAVE a MIDI en tiempo real dedicado a las guitarras eléctricas. El resultado ha sido bastante satisfactorio, sobretodo en el enfoque principal del algoritmo desarrollado, la detección de las notas. Este tipo de programa con algunas mejoras puede ser una útil herramienta para músicos, en especial los guitarristas. A estos les abre la puerta a poder utilizar su guitarra de siempre como controlador MIDI, sintiéndose más cómodos en la interpretación musical.

Hemos podido demostrar la utilidad de un detector de pitch de dominio frecuencial, dada la potencia de proceso de los ordenadores actuales. Además hemos desarrollado un post-procesado de eliminación de armónicos muy efectivo y simple de utilizar, basado en el método de compresión espectral y suma de armónicos.

5.1 Mejoras Futuras

Este algoritmo deja la puerta abierta a la implementación de un sistema de captura de sonido directa desde una tarjeta de audio vía DirectX y la escritura MIDI directa, de forma que la interpretación de guitarra pase directamente como notas MIDI a un secuenciador MIDI.

Otra de las mejoras posibles se encamina hacia la detección de la velocidad MIDI y no únicamente del pitch, posiblemente calculando la energía de los armónicos detectados. Una mejora mucho más compleja sería la introducción del ritmo de guitarra en la ordenación de las notas MIDI, distinguiendo el proceso ADSR (Attack Decay Sustain Release) de la amplitud de la nota. Así escribiríamos la nota desde que empieza hasta que acaba independientemente del resto de notas que suene simultáneamente.

La posibilidad de incorporar una interfaz gráfica es muy viable y podría conseguir un programa mucho más atractivo de cara al uso profesional.

BIBLIOGRAFÍA

- Master of Science Thesis Project "IMPLEMENTATION AND ANALYSIS OF PITCH TRACKING ALGORITHMS"
  Realizado por Stefan Upgard del departamento de Señales, Sensores y Sistemas de KTH, Estocolmo, Suecia.

- Proyecto fin de carrera "DISEÑO DE UNA INTERFAZ MIDI PARA SEÑAL MONOFÓNICA MEDIANTE DSP"
  Realizado por Victor Sánchez Rebull en el departamento de TSC de la Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona (E.T.S.E.T.B).

Páginas web:

- www.midi.org Portal de la unión de fabricantes creadora del estándar MIDI.

- www.borg.com/~jglatt/tech/**midifile**.htm Especificación de un archivo MIDI estándar.

- www.jyu.fi/musica/**miditoolbox** Libreria MIDI para MATLAB.

- www.harmony-central.com/MIDI/files.html Recopilación de librerias MIDI para C.

- www.mathworks.com Portal del desarrollador de Matlab.

- www.steinberg.net Portal del desarrollador de Cubasis VST.

ANEXO A

Comparación de Resultado entre Matlab y C

RESULTADO EN MATLAB e2.wav

Ntrames =

35

ans =

-0.2488 *Valor en posicion 2001*

ans =

-0.0010 *Valor en posicion 201*

40 2 52 2 59 64

40 2 52 2 59 64

40 2 52 2 59 64

40 2 52 2 59 64

2 2 52 2 59 64

40 2 52 2 59 64

2 2 52 2 2 64

2 2 52 2 2 64

2 2 2 2 2 64

2 2 52 2 2 2

2 2 2 2 2 2

2 2 52 2 2 2

2 2 52 2 2 2

2 2 52 2 2 2

2 2 2 2 2 2

2 2 2 2 2 2

2 2 2 2 2 2

Hasta el final...

RESULTADO EN VISUAL C++

C:\>wavetomidi E2.wav

WAVE
2
MIDI

PFC by Albert Molina Reverte

Informacion fichero wav:

Fichero: E2.wav
N canales = 1
Fm = 44100
ByteRate = 88200
Bytes/Bloque = 2
Bits/Sample = 16
Tama±o Buffer = 731644
N bloques = 365822

Trama numero : 0
Resultado: 40 1 52 1 59 64

Trama numero : 1
Resultado: 40 1 52 1 59 64

Trama numero : 2
Resultado: 1 1 52 1 59 64

Trama numero : 3
Resultado: 1 1 52 1 59 64

Trama numero : 4
Resultado: 1 1 52 1 59 64

Trama numero : 5
Resultado: 1 1 52 1 1 64

Trama numero : 6
Resultado: 1 1 52 1 1 64

Trama numero : 7
Resultado: 1 1 52 1 1 64

Trama numero : 8
Resultado: 1 1 1 1 1 1

Trama numero : 9
Resultado: 1 1 1 1 1 1

Trama numero : 10
Resultado: 1 1 1 1 1 1

Trama numero : 11
Resultado: 1 1 1 1 1 1

Trama numero : 12
Resultado: 1 1 52 1 1 1

Trama numero : 13
Resultado: 1 1 1 1 1 1

Hasta el final…

ANEXO B

Tabla de frecuencias y notas MIDI

MIDI NOTE TO FREQUENCY

MIDI	Note	Freq
0	C -2	8.18
1	C# -2	8.66
2	D -2	9.18
3	D# -2	9.72
4	E -2	10.30
5	F -2	10.91
6	F# -2	11.56
7	G -2	12.25
8	G# -2	12.98
9	A -2	13.75
10	A# -2	14.57
11	B -2	15.43
12	C -1	16.35
13	C# -1	17.32
14	D -1	18.35
15	D# -1	19.45
16	E -1	20.60
17	F -1	21.83
18	F# -1	23.12
19	G -1	24.50
20	G# -1	25.96
21	A -1	27.50
22	A# -1	29.14
23	B -1	30.87
24	C 0	32.70
25	C# 0	34.65
26	D 0	36.71
27	D# 0	38.89
28	E 0	41.20
29	F 0	43.65
30	F# 0	46.25
31	G 0	49.00
32	G# 0	51.91
33	A 0	55.00
34	A# 0	58.27
35	B 0	61.74

MIDI	Note	Freq
36	C 1	65.41
37	C# 1	69.30
38	D 1	73.42
39	D# 1	77.78
40	E 1	82.41
41	F 1	87.31
42	F# 1	92.50
43	G 1	98.00
44	G# 1	103.83
45	A 1	110.00
46	A# 1	116.54
47	B 1	123.47
48	C 2	130.81
49	C# 2	138.59
50	D 2	146.83
51	D# 2	155.56
52	E 2	164.81
53	F 2	174.61
54	F# 2	185.00
55	G 2	196.00
56	G# 2	207.65
57	A 2	220.00
58	A# 2	233.08
59	B 2	246.94
60	C 3	261.63
61	C# 3	277.18
62	D 3	293.66
63	D# 3	311.13
64	E 3	329.63
65	F 3	349.23
66	F# 3	369.99
67	G 3	392.00
68	G# 3	415.30
69	A 3	440.00
70	A# 3	466.16
71	B 3	493.88

MIDI	Note	Freq
72	C 4	523.25
73	C# 4	554.37
74	D 4	587.33
75	D# 4	622.25
76	E 4	659.26
77	F 4	698.46
78	F# 4	739.99
79	G 4	783.99
80	G# 4	830.61
81	A 4	880.00
82	A# 4	932.33
83	B 4	987.77
84	C 5	1046.50
85	C# 5	1108.73
86	D 5	1174.66
87	D# 5	1244.51
88	E 5	1318.51
89	F 5	1396.91
90	F# 5	1479.98
91	G 5	1567.98
92	G# 5	1661.22
93	A 5	1760.00
94	A# 5	1864.66
95	B 5	1975.53
96	C 6	2093.00
97	C# 6	2217.46
98	D 6	2349.32
99	D# 6	2489.02
100	E 6	2637.02
101	F 6	2793.83
102	F# 6	2959.96
103	G 6	3135.96
104	G# 6	3322.44
105	A 6	3520.00
106	A# 6	3729.31
107	B 6	3951.07

MIDI	Note	Freq
108	C 7	4186.01
109	C# 7	4434.92
110	D 7	4698.64
111	D# 7	4978.03
112	E 7	5274.04
113	F 7	5587.65
114	F# 7	5919.91
115	G 7	6271.93
116	G# 7	6644.88
117	A 7	7040.00
118	A# 7	7458.62
119	B 7	7902.13
120	C 8	8372.02
121	C# 8	8869.84
122	D 8	9397.27
123	D# 8	9956.06
124	E 8	10548.08
125	F 8	11175.30
126	F# 8	11839.82
127	G 8	12543.85