

# Reconocimiento de Música

Por: Alfonso Alba Cadena  
Instituto de Investigación en Comunicación Óptica

# Indice

1. Descripción del proyecto .....	2
2. Requerimientos y restricciones .....	3
3. Fundamentos .....	5
4. Algoritmos .....	8
5. Uso del programa WATOMI .....	16
6. Referencias .....	18

# 1. Descripción

El proyecto tiene como objetivo analizar una grabación digital de una pieza musical para reconocer las notas que se presentan en dicha pieza, así como su tiempo de inicio y duración. De esta forma, es posible reconstruir la partitura de una composición musical a partir de una grabación.

Para lograr esto, se subdivide la señal digitalizada en bloques de duración pequeña (alrededor de 50 milisegundos) y se obtiene un mapa tiempo-frecuencia mediante la Transformada de Fourier con ventana. A partir de las frecuencias obtenidas, se determinan los tonos cromáticos que ocurren en ese bloque. Existen varios parámetros que se pueden considerar para obtener un equilibrio entre la precisión en frecuencia, la precisión en el tiempo y la velocidad de ejecución del algoritmo.

La señal de entrada será dada como un archivo WAV de Microsoft, con formato PCM sin compresión, una frecuencia de muestreo de 44100 Hz., y 8 o 16 bits de resolución. Prácticamente cualquier programa de edición de audio puede producir archivos con este formato. En particular, para grabar y generar los archivos de prueba se utilizó el programa Cool Edit Pro <sup>1</sup>. Al realizar el reconocimiento tiempo-frecuencia, la información obtenida se convierte en mensajes MIDI, los cuales no son mas que una representación de la partitura de una pieza.

La aplicación principal de un programa de este tipo está orientada claramente a los músicos que deseen transcribir alguna pieza musical de forma relativamente automática. Aunque esto no es posible hacerlo con una precisión perfecta, el resultado puede ser posteriormente editado en un secuenciador MIDI para hacer las correcciones necesarias. Es posible también hacer esto *sin corregir* posteriormente el resultado, obteniendo así una melodía parecida a la señal original pero con algunos cambios e imperfecciones.

---

<sup>1</sup>Se puede obtener una versión de prueba de Cool Edit Pro en [www.syntrillium.com](http://www.syntrillium.com)

## 2. Requerimientos y restricciones

El programa que ejemplifica el proyecto ha sido compilado con Microsoft Visual C++ 5.0, para usarse en Windows 95/98. Los requerimientos mínimos para ejecutar el programa son:

- Procesador Intel Pentium o equivalente
- 64 Mb de memoria RAM
- Video de 800x600 pixels a 256 colores
- Tarjeta de sonido con sintetizador MIDI o puerto MIDI externo
- DirectX 7.0

La velocidad de ejecución del programa depende directamente de la velocidad del procesador, por lo cual se recomienda un procesador de al menos 300 Mhz para obtener un tiempo de reconocimiento aceptable. El programa ha sido probado con las siguientes configuraciones:

- Laptop con procesador AMD K6-2 a 400 Mhz, 96 Mb RAM, tarjeta de sonido integrada.
- PC de escritorio con procesador Pentium III a 550 Mhz, 128 Mb RAM, tarjeta SoundBlaster Live! Platinum.
- PC de escritorio con procesador Pentium III a 600 Mhz, 128 Mb RAM, tarjeta Creative Ensoniq Audio PCI.

El programa acepta como entrada archivos WAV sin compresión a 44.1 KHz con 8 o 16 bits de resolución. La salida es enviada como información MIDI por el canal 1 del puerto MIDI seleccionado por el usuario. Esta versión del programa no incluye la opción para grabar en disco el archivo MIDI resultante.

El programa puede reconocer tanto señales monofónicas como polifónicas. Se asume que la señal de entrada es una grabación de algún instrumento musical con un timbre parecido al de una onda senoidal. Debido a esto, los sonidos de algunos instrumentos pueden generar “notas falsas” correspondientes a las armónicas producidas por dicho instrumento. Similarmente, una señal que presente ruido o ciertos instrumentos de percusión pueden ocasionar que el programa genere notas falsas.

Por otra parte, el programa funciona mejor con notas altas que bajas. Esto es debido a la escala exponencial que presentan las frecuencias de las

notas. La diferencia de frecuencias en notas bajas puede llegar a ser tan pequeña como para el programa confunda esas notas.

Todo esto significa que el programa tiene mayor probabilidad de reconocer exitosamente una señal si ésta se compone de notas altas tocadas con un instrumento con un timbre senoidal (como la flauta).

Finalmente, si se desea reconocer un archivo de gran tamaño (varios megabytes), se aconseja dividirlo previamente en varias secciones. De esta forma se pueden utilizar distintos parámetros del programa para reconocer cada sección.

### 3. Fundamentos

#### Transformada de Fourier Discreta

La señal de entrada  $\tilde{f}$  es una muestra digitalizada de una señal de audio en la que su valor en un tiempo específico es la amplitud de la señal y es por lo tanto real. A partir de ahora asumiremos lo siguiente:

- $\tilde{f}$  es una señal discreta finita de tamaño  $N$ ,
- $N$  es una potencia de dos,
- $\tilde{f}[n] \in R$  para  $n = 0, \dots, N - 1$ ,
- $|\tilde{f}[n]| \leq 1$  ( $\tilde{f}$  está normalizada).

En general, las señales de tamaño  $N$  pueden ser complejas, por lo tanto, el conjunto de señales de tamaño  $N$  es  $C^N$ , el cual es un campo. Podemos entonces definir el producto escalar de dos señales  $g$  y  $h$  como

$$\langle f, h \rangle = \sum_{n=0}^{N-1} f[n] \overline{g[n]}.$$

Para definir la Transformada de Fourier de nuestra señal, nos interesa que esta sea periódica, por lo cual definimos  $f$  como

$$f[n] = \tilde{f}[n \bmod N], \quad \text{para } n \in Z.$$

Entonces, la Transformada de Fourier Discreta  $\hat{f}$  de  $f$  se define como

$$\hat{f}[k] = \sum_{n=0}^{N-1} f[n] e^{\frac{-i2\pi kn}{N}}.$$

Se puede probar que la familia de funciones

$$\{e_k[n] = e^{\frac{i2\pi kn}{N}}\}$$

para  $0 \leq k < N$  es una base ortogonal del espacio de señales de periodo  $N$  (observar que  $\hat{f}[k] = \langle f, e_k \rangle$ ). Por lo tanto,  $\hat{f}[k]$  nos dice qué tanto oscila  $f$  a la frecuencia  $2\pi k/N$  (en radianes). De esta forma podemos saber cuales son las frecuencias que componen la señal.

## Análisis Tiempo-Frecuencia

Para el reconocimiento de música, no solamente es necesario reconocer las frecuencias que se producen, sino en qué momento se producen y la duración que tienen. Esto es algo que la Transformada de Fourier Discreta no puede lograr por sí sola. Para localizar las frecuencias utilizamos la Transformada de Fourier Discreta con Ventana. Este método analiza la señal en un intervalo alrededor de cada punto en el tiempo, obteniendo de esta forma una familia de señales que están bien localizadas en tiempo y frecuencia. Estas señales se conocen como átomos tiempo-frecuencia y pueden representarse como un mapa bidimensional (tiempo-frecuencia) donde la intensidad en cada un punto  $[m, l]$  del mapa representa la intensidad de la frecuencia  $l$  al tiempo  $m$ . La señal es multiplicada por una ventana  $g$ , la cual es una señal discreta real y simétrica respecto a  $N/2$ , para controlar la forma en que se concentra la energía alrededor de un punto.

Para el caso de la T.F.D. con ventana, los átomos se definen de la siguiente manera:

$$g_{m,l}[n] = g[n - m]e^{\frac{i2\pi ln}{N}}$$

donde  $g$  es la ventana mencionada en el párrafo anterior. El programa WATOMI permite elegir entre tres tipos de ventanas que son las siguientes:

Gausiana	$g[n] = e^{-18(\frac{i}{N-1} - \frac{1}{2})^2}$
Hanning	$g[n] = \cos^2(\pi(\frac{i}{N-1} - \frac{1}{2}))$
Blackman	$g[n] = 0.42 + 0.5 \cos(2\pi(\frac{i}{N-1} - \frac{1}{2})) + 0.08 \cos(4\pi(\frac{i}{N-1} - \frac{1}{2}))$

La Transformada de Fourier Discreta con Ventana de  $f$  se escribe entonces como

$$\begin{aligned} Sf[m, l] &= \langle f, g_{m,l} \rangle \\ Sf[m, l] &= \sum_{n=0}^{N-1} f[n]g[n - m]e^{\frac{-i2\pi ln}{N}} \end{aligned}$$

$Sf[m, l]$  es la función que describe el mapa tiempo-frecuencia que usaremos para el análisis. Sin embargo,  $Sf$  es una función compleja y nosotros requerimos únicamente un valor real de la intensidad de la frecuencia  $l$  al tiempo  $m$ , por lo cual definimos una densidad de energía (llamada *espectrograma*)  $P_s$  como

$$P_s f[m, l] = |Sf[m, l]|^2.$$

El espectrograma de  $f$  es una representación matemática de la partitura de la cual proviene la señal de audio  $f$ .



### 3. Algoritmos

#### Transformada Rápida de Fourier (FFT)

El cálculo de la T.F. de una señal requiere una cantidad de operaciones del orden de  $N^2$ . Existe un algoritmo que divide el cálculo de la T.F. de una señal de tamaño  $N$  al cálculo de dos T.F.'s de tamaño  $N/2$ . Aplicando este algoritmo recursivamente obtenemos un orden de complejidad  $O(N \log N)$  para el cálculo de la T.F. de tamaño  $N$ . El algoritmo recursivo es el siguiente:

##### Entradas:

$f$  es una señal de tamaño  $N$   
 $N$  es una potencia de dos

##### Algoritmo FFT( $f$ , $N$ )

- 1) Si  $N = 2$ 
  - a)  $f[0]$  por  $f[0] + f[1]$  y  $f[1]$  por  $f[0] - f[1]$
  - b) Termina el algoritmo
- 2) Definir  $g[n] = f[2n]$  y  $h[n] = f[2n + 1]$  para  $0 \leq n < N/2$
- 3) Calcular FFT( $g$ ,  $N/2$ ) y FFT( $h$ ,  $N/2$ )
- 4) Reemplazar  $f[n]$  por  $g[n] + e^{-i\frac{2\pi n}{N}} h[n]$  para  $0 \leq n < N$ .

Este es un algoritmo destructivo; es decir que reemplaza la señal  $f$  por su T.F.D., por lo cual hay que trabajar con una copia de la señal original o de lo contrario esta se perderá.

#### Generación del mapa tiempo-frecuencia

Para reconocer los tonos de una pieza musical a partir de una señal de audio  $f$ , necesitamos analizar la T.F.D. con ventana  $Sf$  para determinar qué frecuencias se escuchan y en qué momento. Para señales muy largas, sin embargo, los requisitos de almacenamiento y procesamiento aumentan considerablemente, haciendo necesario el uso de la Transformada Rápida de Fourier y la subdivisión de la señal original en señales de menor duración.

Partiendo de que  $Sf = \sum_{n=0}^{N-1} f[n]g[n-m]e^{i2\pi ln/N}$  y de que  $g$  es par, podemos observar que para un valor fijo de  $m$ ,  $Sf[m, l]$  es la T.F. de  $f[n]g[n-m]$  evaluada en  $l$ . Por lo tanto, podemos calcular  $Sf$  mediante  $N$  transformadas rápidas de Fourier, con una complejidad de orden  $O(N^2 \log N)$ .

Desde el punto de vista de almacenamiento,  $Sf$  requiere  $N^2$  “localidades” de memoria, las cuales en nuestro caso serán de 32 bits cada una. Esto significa que requerimos una capacidad de  $4N^2$  bytes tan solo para almacenar  $Sf$ . La solución que se propone a este problema consiste en calcular y analizar  $Sf$  en bloques de  $M$  muestras. Requerimos que  $M$  sea una potencia de dos para poder aplicar la F.F.T., y por otro lado,  $M$  debe ser lo suficientemente grande como para poder detectar frecuencias bajas y lo suficientemente pequeño para obtener una precisión aceptable en el tiempo.

La frecuencia mas baja que nos interesa detectar es alrededor de 20 Hz. También podemos asumir que la frecuencia de muestreo de la señal original es de 44100 Hz (el estándar para los discos compactos). Esto significa que requerimos alrededor de  $44100/20 = 2205$  muestras para detectar al menos un ciclo de 20 Hz. Las potencias de dos mas cercanas a 2205 son 2048 y 4096. Debido a esto, se propone  $M = 2048$  como valor óptimo. El programa utiliza este valor por default, aunque permite usar cualquier potencia de dos entre 1024 y 8192, inclusive. Valores mayores para  $M$  permiten un mejor reconocimiento de frecuencias bajas y valores menores permiten mayor precisión en el tiempo y mayor eficiencia.

Finalmente, el algoritmo que se propone para realizar el análisis es el siguiente:

**Entradas:**

- El *tamaño de bloque*  $M = 2^p$ ,
- Una señal  $f \in l_2([0, N - 1])$ , donde  $N = 2^q \geq M$ ,
- Una ventana  $g \in l_2([0, M - 1])$ .

**Algoritmo:**

- 1) Calcular  $K = N/M$ .
- 2) Para  $j = 0$  hasta  $K - 1$ 
  - a) Calcular  $f_j[n] = f[Mj + n]$  para  $n = 0, \dots, M - 1$ .
  - b) Para  $m = 0$  hasta  $M - 1$  calcular
    - i)  $h_m[n] = f_j[n]g[n - m]$  para  $n = 0, \dots, M - 1$ .
    - ii) Calcular  $\hat{h}_m$  mediante la FFT.  $Sf_j[m, l] = \hat{h}_m[l]$ .
  - c) Analizar  $Sf_j$  para obtener las frecuencias que se producen en el bloque  $j$ .

La información obtenida en el paso c) se utiliza para reconstruir la “partitura” que dió origen a la señal  $f$ . Más adelante nos enfocaremos únicamente en este paso.

Se puede ver fácilmente que el algoritmo anterior tiene un orden de complejidad  $O(NM \log M)$ . Si se propone una  $M$  constante, entonces el algoritmo tiene complejidad lineal  $O(N)$ . Sin embargo, aunque el tiempo requerido para calcular una FFT de tamaño  $M$  es muy pequeño, debemos calcular  $N$  de estas transformadas, lo cual se convierte en un proceso muy lento. Esto significa que debemos modificar el algoritmo original para obtener una mayor velocidad.

La primera modificación consiste en asumir que las frecuencias que suceden en un bloque determinado no varían en gran medida. Es decir que podemos considerar que para un  $j$  fijo y  $m_1, m_2 \in \{0, \dots, M-1\}$  sucede que  $Sf_j[m_1, l] = Sf_j[m_2, l]$  para todo  $0 \leq l < M$ . Entonces es suficiente con calcular  $Sf_j[m, l]$  solamente para un valor de  $m$ . Esto divide el número de operaciones a realizar por  $M$ , lo cual nos deja con una complejidad de orden  $O(N \log m)$ .

El método anterior claramente realiza una cuantización muy severa del tiempo. El inicio o fin de cada nota se detectarían solamente al principio de cada bloque y las notas lo suficientemente cortas como para caber en un bloque podrían fácilmente ser pasadas por alto. Es decir que esta forma de optimización tiene sentido cuando las notas a reconocer tienen una duración relativamente larga. Para reconocer notas cortas necesitamos mayor resolución en el tiempo. La solución que se propone consiste en agregar un parámetro  $S_m$  que indique el número de valores distintos de  $m$  para los cuales se calcula  $Sf_j[m, l]$ . Dichos valores de  $m$  están regularmente espaciados por lo cual podemos decir que  $S_m$  indica el número de muestras de  $Sf_j[m, l]$  que se van a calcular para un  $j$  fijo. Es por eso que a  $S_m$  le llamamos *parámetro de submuestreo*. Si  $S_m = 1$ , obtenemos la optimización descrita en el párrafo anterior. Si existen notas muy cortas, se puede incrementar  $S_m$  a 2, 4 ú 8 (son preferibles las potencias de dos) para reconocerlas. El algoritmo modificado queda de la siguiente manera:

**Entradas:**

- El tamaño de bloque  $M = 2^p$ ,
- Una señal  $f \in l_2([0, N-1])$ , donde  $N = 2^q \geq M$ ,
- Una ventana  $g \in l_2([0, M-1])$ ,

- El parámetro de submuestreo  $S_m = 2^r$ .

**Algoritmo:**

- 1) Calcular  $K = N/M$ .
- 2) Calcular  $\Delta = M/S_m$ .
- 3) Para  $j = 0$  hasta  $K - 1$ 
  - a) Calcular  $f_j[n] = f[Mj + n]$  para  $n = 0, \dots, M - 1$ .
  - b) Para  $m = 0$  hasta  $S_m - 1$  calcular
    - i)  $h_{m\Delta}[n] = f_j[n]g[n - m\Delta]$  para  $n = 0, \dots, M - 1$ .
    - ii) Calcular  $\hat{h}_{m\Delta}$  mediante la FFT.  $Sf_j[m\Delta, l] = \hat{h}_{m\Delta}[l]$ .
    - iii) Analizar  $Sf_j[m\Delta, l]$ ,  $l = 0, \dots, M-1$ , para obtener las frecuencias que se producen en el bloque  $j$ .

Este nuevo algoritmo tiene orden de complejidad  $O(NS_m \log M)$ . Sin embargo,  $S_m$  es mucho menor que  $M$ , por lo cual la eficiencia de este algoritmo es considerablemente mejor que la del algoritmo original, y además se puede obtener gran precisión en el tiempo. El programa permite valores de  $S_m = 1, 2, 4, 8, 16$ . Esto, combinado con el menor valor para  $M$  que es 1024, nos da una resolución de 1.5 milisegundos.

**Bajas frecuencias**

Supongamos que tenemos una señal continua  $h$  que consiste únicamente de un tono constante de frecuencia  $f_0$  en hertz; es decir que  $h(t) = e^{i2\pi f_0 t}$ . Si deseamos muestrear esta señal con un periodo de muestreo de  $T$  segundos, entonces tenemos que

$$h[n] = h(nT) = e^{i2\pi f_0 T n}.$$

Supongamos también que la señal muestreada es finita, de tamaño  $N$ . Podemos entonces obtener su T.F.D. de la siguiente manera:

$$\begin{aligned}\hat{h}[k] &= \sum_{n=0}^{N-1} h[n] e^{-i \frac{2\pi k n}{N}} \\ \hat{h}[k] &= \sum_{n=0}^{N-1} e^{i2\pi f_0 T n} e^{-i \frac{2\pi k n}{N}} \\ \hat{h}[k] &= \sum_{n=0}^{N-1} e^{i2\pi n (f_0 T - \frac{k}{N})}\end{aligned}$$

Es claro que  $|\hat{h}[k]| \leq N$  para  $0 \leq k < N$  y que  $\hat{h}[k] = N$  cuando  $f_0 T = k/N$ . Sabemos que en el caso continuo, la Transformada de Fourier de  $h$  debe ser igual a una delta de Dirac localizada en  $f_0$ . En el caso discreto, lo que obtenemos es un “pico” localizado en  $k_0$  tal que  $f_0 T - k_0/N$  sea mínimo (es posible que  $f_0 T = k/N$  no se cumpla para ningún  $k$ ). Ya que  $T$  y  $N$  son constantes, lo que obtenemos es una relación entre  $k$  y la frecuencia en hertz  $f_{Hz}$  que corresponde a dicha  $k$ :

$$f_{Hz} = \frac{k}{NT} = \frac{k f_m}{N},$$

donde  $f_m = 1/T$  es la frecuencia de muestreo en hertz. Es claro entonces que la T.F.D. solo puede distinguir frecuencias cuya diferencia es mayor que  $f_m/N$  hertz. Esto puede ocasionar que a dos frecuencias muy cercanas les corresponda la misma  $k$ . Debido a la escala exponencial que siguen las frecuencias de las notas, es posible que existan dos o mas notas de frecuencia tan baja que sean imposibles de distinguir.

Hemos mencionado que la frecuencia de muestreo de las señales de entrada que deseamos manejar es de 44100 Hz, y también que dichas señales serán divididas en bloques de  $M$  muestras, donde  $M$  puede ser 1024, 2048, 4096 ó 8192. Para  $M = 2048$ , por ejemplo, tenemos que  $f_m/M = 44100/2048 = 21.53$ . Es decir que la T.F.D. puede distinguir dos notas solamente si la diferencia de sus frecuencias es mayor que 21 Hz. Sabiendo que la nota A4 (La nota “La” en la cuarta octava de un piano) tiene una frecuencia de 440 Hz y que al aumentar una octava se duplica la frecuencia de una nota, podemos determinar que la T.F.D. solamente distingue notas mayores que F3. Si aumentamos el tamaño de bloque  $M$  a 4096, de forma que  $f_m/M = 10.77$ , podremos reconocer frecuencias a partir de F2.

Por lo tanto, una forma en que podemos reconocer frecuencias bajas consiste en aumentar el tamaño de bloque  $M$ , lo cual a su vez reduce la resolución en el tiempo. Sin embargo, podemos incrementar también el parámetro de submuestreo  $S_m$  para recuperar la resolución en tiempo sin afectar el reconocimiento de frecuencias.

Ya que deseamos que  $f_m/M$  sea lo más pequeño posible, es posible disminuir  $f_m$  en vez de aumentar  $M$ . Esto equivale a aplicar un filtro sacamuestras a la señal original, lo cual puede afectar el reconocimiento de altas frecuencias. Podemos entonces realizar el reconocimiento en varias etapas que denominamos *pasadas*, en donde en la primera pasada se analiza la señal

original para reconocer frecuencias medias y altas. En la siguiente pasada se aplica un sacamuestras  $K_2$  a la señal y se realiza el reconocimiento de frecuencias una octava mas abajo que la mas baja frecuencia reconocida en la pasada anterior. En caso de ser necesario reconocer frecuencias aún mas bajas, se realizan nuevas pasadas aplicando un sacamuestras  $K_4$  o incluso  $K_8$ . Esto debe ser suficiente para reconocer todas las frecuencias audibles. Al aplicar un filtro sacamuestras a una señal, también se reduce la resolución en el tiempo, por lo que se debe incrementar el parámetro  $S_m$  para contrarrestar dicha reducción.

A continuación se presenta una nueva versión del algoritmo de reconocimiento para permitir pasar la señal por un sacamuestras:

**Entradas:**

- El tamaño de bloque  $M = 2^p$ ,
- Una señal  $f \in l_2(N)$ , donde  $N = 2^q \geq M$ ,
- Una ventana  $g \in l_2([0, M - 1])$ ,
- El parámetro de submuestreo  $S_m = 2^r$ .
- El parámetro de *downsampling*  $d$ . La señal será filtrada por  $K_d$ .

**Algoritmo:**

- 1) Calcular  $K = N/Md$ .
- 2) Calcular  $\Delta = M/S_m$ .
- 3) Para  $j = 0$  hasta  $K - 1$ 
  - a) Calcular  $f_j[n] = f[(Mj + n)d]$  para  $n = 0, \dots, M - 1$ .
  - b) Para  $m = 0$  hasta  $S_m - 1$  calcular
    - i)  $h_{m\Delta}[n] = f_j[n]g[n - m\Delta]$  para  $n = 0, \dots, M - 1$ .
    - ii) Calcular  $\hat{h}_{m\Delta}$  mediante la FFT.  $Sf_j[m\Delta, l] = \hat{h}_{m\Delta}[l]$ .
    - iii) Analizar  $Sf_j[m\Delta, l]$ ,  $l = 0, \dots, M-1$ , para obtener las frecuencias que se producen en el bloque  $j$ .

Hay que notar que cada bloque de una señal que es filtrada por  $K_d$  es  $d$  veces mas largo en tiempo que un bloque de la señal sin filtrar. Esto se debe tomar en cuenta a la hora de medir las duraciones de las notas reconocidas.

**Reconocimiento de las notas**

Dado un bloque  $j$  fijo, una vez que hemos calculado  $Sf_j[m, l] = \hat{h}_m[l]$  para algún  $m$ , debemos determinar las frecuencias con mayor potencia en ese tiempo. El primer paso para realizar dicho análisis consiste en calcular y

normalizar la magnitud de  $\hat{h}_m[l]$  para  $0 \leq l < M$ . Ya que tanto la señal de entrada como la ventana  $g$  están normalizadas en magnitud a uno, entonces es fácil ver que  $|\hat{h}_m[l]|^2 \leq M$ ; por lo tanto, definimos la energía normalizada de la frecuencia  $l$  al tiempo  $m$  como  $P_m[l] = |\hat{h}_m[l]|^2/M$ ; todo esto sin olvidar que estamos enfocándonos localmente en el bloque  $j$ , ya que en realidad, esta energía corresponde al tiempo  $(Mj + m)T$  (en segundos), donde  $T$  es el periodo de muestreo de la señal.

$P_m[l]$  nos da una medida de la *presencia* de la frecuencia  $l$  al tiempo  $m$ . Ya hemos visto que la frecuencia  $l$  corresponde a  $f_m l/M$  en hertz, pero lo que necesitamos es una forma de determinar la nota producida a partir de  $l$ . El standard MIDI define un mapeo de los enteros del 0 al 127 a las notas de la escala cromática, en donde el 60 corresponde al Do central de un piano ( $C4 = 261.62$  Hz). Supongamos que tenemos una función discreta  $\text{Fnm} : \{0, \dots, 127\} \rightarrow \mathbb{R}$  donde  $\text{Fnm}[i]$  es igual a la frecuencia de la  $i$ -ésima nota MIDI (por ejemplo:  $\text{Fnm}[60] = 261.62$ ). Esta función puede ser calculada fácilmente conociendo la progresión exponencial de las frecuencias de las notas; sin embargo, el programa define esta función como un arreglo, en el cual los valores de frecuencias han sido especificados “a mano” debido a que la entonación común de las notas difiere un poco de la escala exponencial.

Para determinar la nota MIDI correspondiente a la frecuencia  $l$ , buscamos la nota tal que el logaritmo de su frecuencia sea el más cercano al logaritmo de la frecuencia correspondiente a  $l$ . Es decir, buscamos la  $u$  que minimice

$$|\log \frac{f_m l}{M} - \log \text{Fnm}[u]|.$$

Con esto podemos definir la función  $\eta$  de forma que  $\eta[l]$  sea igual a la nota MIDI correspondiente a la frecuencia  $l$ , donde  $0 \leq l < M$ . Esto nos permite calcular la energía  $E_m$  de cada nota al tiempo  $m$  (ya no de cada frecuencia) de la siguiente forma:

$$\begin{aligned} \mu[l, u] &= \begin{cases} 1 & \text{si } u = \eta[l] \\ 0 & \text{si } u \neq \eta[l] \end{cases} \\ E_m[u] &= \frac{\sum_{l=0}^{M-1} P_m[l] \mu[l, u]}{\sum_{l=0}^{M-1} \mu[l, u]} \end{aligned} \tag{1}$$

Es decir que  $E_m[u]$  es el promedio de las energías normalizadas de las frecuencias que corresponden a la nota  $u$ . Esto es finalmente lo que hemos

estado buscando: una función que nos de una medida normalizada de la presencia de cada nota al tiempo  $m$  (dentro del bloque  $j$ ).

Para determinar si una nota tiene la suficiente energía como para ser considerada parte de la partitura, definimos un *umbral*  $\mathcal{U} \in [0, 1]$ . Entonces, una nota  $u$  será incluida en la partitura al tiempo  $m$  si y solo si  $E_m[u] > \mathcal{U}$ .

Nuestra partitura es entonces un mapa discretizado tiempo-nota cuyos elementos son de tipo booleano e indican únicamente si una nota se escucha a un tiempo determinado o no. Recorriendo dicho mapa sobre el eje del tiempo, podemos fácilmente determinar cuando inicia y termina cada nota, y generar información MIDI que represente la partitura de una forma manejable. La descripción de este proceso de recorrido no es la “atracción principal” del proyecto y no se explica aquí; sin embargo, se puede revisar el código fuente para tener una mejor idea de cómo realizar este procedimiento. La función en la que se realiza todo el análisis descrito es `Analyzer::Analyze()` y se encuentra en el archivo `analyze.cpp`.

## Posibles mejoras

Existen varias cuestiones e ideas que no han sido consideradas dentro del proyecto:

- a) Detectar el volumen de cada nota. Por ahora, todas las notas se reproducen con el mismo volumen.
- b) Realizar un escalamiento automático de los átomos tiempo-frecuencia para evitar la necesidad de varias pasadas.
- c) Aplicar propiedades conocidas de los instrumentos musicales para eliminar armónicas no deseadas.
- d) Detectar cambios continuos de frecuencia (como el vibrato o un slide de guitarra).



## 5. Uso del programa WATOMI

El programa que se presenta implementa el algoritmo de reconocimiento descrito en la sección anterior y permite modificar los distintos parámetros que se utilizan. Al ejecutar el programa aparecen dos ventanas, una de ellas consiste en una ventana de diálogo en donde se especifican los parámetros del algoritmo, y en la otra se muestra una representación de la señal de entrada y del mapa tiempo-frecuencia obtenido.

Antes que nada, hay que cargar un archivo WAV que contenga la señal a reconocer. Para ello se oprime el botón [Cargar WAV] con lo cual aparece una ventana de diálogo para elegir el archivo. Una vez que se ha cargado el archivo, se puede utilizar el botón [Reproducir] para escuchar la señal, o bien, el botón [Analizar] para comenzar el reconocimiento. Durante el análisis, una barra representa el progreso de cada pasada de reconocimiento. Al terminar el proceso se activa el botón [Repr. MIDI] con el cual se puede reproducir el resultado del análisis a través de un puerto MIDI, el cual puede ser seleccionado al oprimir el botón [Conf. MIDI]. Las notas reconocidas se envían por el canal 1 del puerto seleccionado.

Los parámetros que se utilizan, y que ya han sido descritos anteriormente (con excepción de la polifonía) se resumen en:

**Tamaño de bloque ( $M$ ).**- Es el número de muestras por bloque. Los posibles valores son 1024, 2048, 4096 y 8192. Se puede incrementar este valor para mejorar el reconocimiento de bajas frecuencias, sin embargo, esto implica una menor resolución en el tiempo y menor eficiencia del algoritmo.

**Polifonía.**- Es el número máximo de notas que puede tener la partitura en un tiempo determinado. Este parámetro se incluye para descartar armónicas no deseadas aún cuando su energía supere el umbral. Los valores permitidos están entre 1 y 16, inclusive.

**Ventana ( $g$ ).**- Define la ventana que se utiliza para calcular  $h_m$ . Las posibles ventanas son Gausiana, Hanning y Blackman. Estas ventanas producen resultados ligeramente distintos.

**Submuestreo ( $S_m$ ).**- Este parámetro se puede usar para incrementar la resolución en el tiempo, especialmente si se ha incrementado el tamaño de bloque o si se aplican varias pasadas para reconocer bajas frecuencias. Los valores permitidos para este parámetro son 1, 2, 4, 8 y 16.

**Número de pasadas.-** Indica el número de veces que la señal será filtrada por un sacamuestras y reanalizada para reconocer bajas frecuencias. Este método es más eficiente que aumentar el tamaño de bloque pero también tiene menor precisión. El programa permite realizar hasta cuatro pasadas.

**Umbral ( $\mathcal{U}$ ).**- El umbral se especifica por medio de un control deslizador en el cual va de cero a uno de abajo hacia arriba. El valor por default es 0.8.

## 6.- Referencias

A Wavelet Tour of Signal Processing  
Stéphane Mallat  
Academic Press, 1999. Second Edition

La Transformada Rápida de Fourier  
Revista Solo Programadores  
Año 3, Número 17.

MIDI Technical Fanatic's Brainwashing Center  
<http://www.borg.com/~jglatt/>

OpenPTC Graphics Library  
<http://www.gaffer.org/ptc>

FMOD Sound And Music API for Windows 95/98/NT  
Firelight Multimedia  
<http://www.fmod.org>