Comparación de Implementaciones en C y Matlab de Filtros Adaptativos para DSP

Bernardo Morcego
Dept. ESAII (UPC), Rbla. Sant Nebridi 10 (Terrassa, Barcelona), bernardo@esaii.upc.es

Miquel Àngel Cugueró
Dept. ESAII (UPC), Rbla. Sant Nebridi 10 (Terrassa, Barcelona)

Resumen

En este artículo se presenta un estudio comparativo de dos técnicas de implementación de filtros adaptativos con el equipo DSP DS1003 de dSPACE. Se realiza la implementación de diferentes algoritmos de adaptación mediante técnicas de generación de código automáticas y mediante transcripción manual en C. Se evalúa la eficiencia de los dos tipos de implementación en términos de los tamaños de los filtros que cada una permite realizar.

Palabras Clave: Filtros adaptativos, cancelación de ruido, algoritmo RLS.

1 INTRODUCCIÓN

Esta aportación se enmarca en un proyecto de mayor envergadura, Control Activo de Perturbaciones (CAP), cuya finalidad es el estudio y puesta en marcha de esquemas de control para eliminar, de forma activa, el efecto de las perturbaciones mecánicas.

1.1 CONTROL ACTIVO DE PERTURBACIONES MECÁNICAS EN UN CONDUCTO

Dentro del control activo de perturbaciones mecánicas, que es un tema amplio y con multitud de variantes y particularidades (ver [2]), se ha escogido el control activo de perturbaciones en un conducto. Este problema consiste en la eliminación (atenuación) de perturbaciones sonoras (ruido) que se propagan a lo largo de un conducto. Se trata de un problema frecuente, por ejemplo en grandes instalaciones de aire acondicionado.

Para resolver este problema se suelen utilizar captadores de ruido (micrófonos generalmente, pero también acelerómetros, etc.) y generadores de sonido (altavoces). El sonido original es transformado digitalmente, mediante una DSP, para generar un 'contrasonido' que se emite en el punto de cancelación (ver figura 1). El papel de la DSP es el de generar la señal sonora que sumada al ruido original producirá silencio en el punto de cancelación. Es decir, ha de modelar la propagación del ruido a través del conducto.

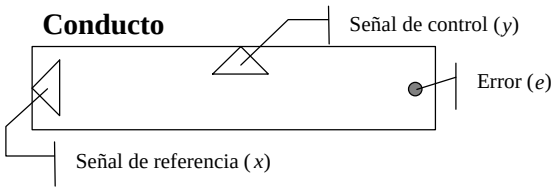

Figura 1: Esquema de un sistema de cancelación de ruido en un conducto

Este esquema de control presenta diversos problemas que han de tratarse con sumo cuidado si se quieren conseguir resultados positivos. En este artículo abordamos el problema de la implementación del algoritmo de cancelación, comparando la realización con un lenguaje de cuarta generación y la realización directa en C.

1.2 PLATAFORMA DE TRABAJO

La plataforma de trabajo que se ha utilizado en este proyecto consta de los siguientes elementos: el conducto de propagación del sonido, los transductores y acondicionadores, la DSP y un PC para programar el sistema.

El conducto de propagación es un paralelepípedo de conglomerado de madera de 3.54 m. de longitud y 0.30 m. de lado. Los transductores son dos altavoces (Beyma 5 MP60/N de 50W) y un micrófono (Behringer ECM8000). Ambos trabajan óptimamente en rangos bajos de frecuencias. Además de estos transductores se dispone de acondicionadores de señales: un preamplificador (Behringer Eurorack MX602A) y un amplificador (Ecler MPA 6-80).

La DSP es una DS1003 de dSPACE, uno de los modelos más potentes de este fabricante. Consta de un procesador TMS320C40 de Texas Instruments y de una tarjeta de comunicaciones con capacidad para 20 entradas y 10 salidas analógicas.

Para finalizar, el PC de programación no tiene una incidencia importante en el rendimiento del sistema de cancelación. Lo único importante es que tenga capacidad para ejecutar Matlab v5.1. De este programa se utilizan, como requisitos especiales, la *DSP Blockset* y la *Real Time Workshop*.

1.3 ESTRUCTURA DEL ARTÍCULO

Este artículo se divide en la presente introducción y 5 partes más. La sección 2 plantea los algoritmos de cancelación, que son filtros adaptativos. Las secciones 3 y 4 describen dos formas de implementar estos algoritmos. En la sección 5 se comparan ambas implementaciones, especialmente en cuanto a la cantidad máxima de parámetros que cada implementación puede realizar. Finalmente se dan las conclusiones resultantes en la sección 6.

2 FILTROS ADAPTATIVOS

El elemento principal de procesado, en esta aplicación, es un filtro digital adaptativo. Se trata de un filtro digital clásico, con la particularidad de que sus parámetros pueden variar a lo largo del tiempo. El encargado de modificar estos parámetros es un algoritmo de optimización que, con la finalidad de minimizar algún criterio, calcula constantemente los parámetros más adecuados. Seguidamente se exponen más detalladamente los tipos y funciones más habituales de filtros digitales y los algoritmos de optimización más utilizados.

2.1 TIPOS DE FILTROS

Un filtro digital realiza una transformación (en este caso lineal) de la señal de entrada, alterando su contenido espectral, la magnitud y la fase para acomodarla a unas especificaciones. En este artículo se describen los dos filtros más utilizados en la cancelación activa de perturbaciones, que son: el filtro FIR (*Finite Impulse Response*) y el filtro IIR (*Infinite Impulse Response*).

2.1.1 Filtros FIR

Los filtros FIR, de respuesta impulsional finita, deben su nombre al hecho de que su salida depende únicamente de la entrada, de forma que el efecto de un impulso en la entrada se extingue en tiempo finito.

La respuesta de un filtro FIR de de L coeficientes, w_i, para una entrada $x(k)$ es

$$y(k) = \sum_{l=0}^{L-1} w_l x(k-l) \qquad (1)$$

y su función de transferencia discreta es

$$\frac{Y(z)}{X(z)} = \sum_{l=0}^{L-1} w_l z^{-l}$$

Un filtro FIR siempre es estable, pues no añade polos al sistema. Por otro lado, no puede realizar funciones de transferencia racionales con un número finito de parámetros.

2.1.2 Filtros IIR

Los filtros IIR, de respuesta impulsional infinita, deben su nombre al hecho de que su salida puede depender tanto de la entrada como de la propia salida del filtro, de forma que el efecto de un impulso en la entrada puede no extinguirse en tiempo finito.

La respuesta de un filtro IIR de N y M coeficientes, a_n y b_m, para una entrada $x(k)$ es

$$y(k) = \sum_{n=0}^{N} a_n x(k-n) + \sum_{m=1}^{M} b_m y(k-m) \qquad (2)$$

y su función de transferencia discreta es

$$\frac{Y(z)}{X(z)} = \frac{\sum_{n=0}^{N} a_n z^{-n}}{1 - \sum_{m=1}^{M} b_m z^{-m}}$$

Los filtros IIR son sistemas genéricos de procesado de señales. Su principal ventaja es que pueden realizar cualquier transformación lineal y discreta con un número finito de parámetros, resultando más eficientes que los filtros FIR. Por otro lado, tienen inconvenientes importantes, como la falta de garantía en su estabilidad o los problemas derivados de la cuantización y redondeo de sus coeficientes, mucho mayores que en el caso de los filtros FIR.

2.2 ALGORITMOS DE ADAPTACIÓN

Los algoritmos de adaptación son algoritmos de optimización cuya finalidad es obtener los parámetros (de un filtro, en este caso) que minimicen algún criterio preestablecido. En esta aplicación el criterio a minimizar gira entorno a la señal capturada en el punto de cancelación, el error de cancelación.

Antes de describir los algoritmos que se utilizan en este artículo es conveniente introducir una expresión más general para las ecuaciones (1) y (2).

$$y(k) = r^T(k) \cdot p \qquad (3)$$

donde los vectores p y r corresponden a los parámetros del filtro (w_i, a_n y b_m en las expresiones anteriores) y a los regresores, que son valores conocidos e independientes del vector de parámetros.

Por otro lado, definimos el error de cancelación como

$$e(k, p) = d(k) - y(k) . \qquad (4)$$

La señal $d(k)$ es la señal que se desea cancelar, es decir, es el ruido que se propaga por el conducto hasta el punto de cancelación. Esta señal no está físicamente disponible (ver figura 1) pero sí que disonemos de la lectura del error. En la ecuación (4) se hace evidente que el error depende tanto de la señal de ruido como del conjunto de parámetros empleados en la cancelación.

Con todo ello, el tipo de algoritmos de optimización que se utiliza en esta aplicación calcula los parámetros de forma recursiva. Es decir, en cada paso de cálculo se actualizarán los parámetros según

$$p(k+1) = p(k) + g(k) \cdot e(k, p) , \qquad (5)$$

donde $g(k)$ es un vector que se obtiene a partir del criterio concreto a minimizar.

En este artículo se utilizan tres criterios de minimización que se explican seguidamente.

2.2.1 Criterio LMS

El criterio LMS (*Least Mean Squares*) es un método de cálculo de $g(k)$ ampliamente utilizado, especialmente en aplicaciones de procesado de señales en las que la velocidad de cálculo es alta.

En este algoritmo se parte de la función de coste cuadrática del error, pero se utiliza una solución aproximada en la que se evalúan los gradientes instantáneos de esta función. Como resultado, el cálculo de $g(k)$ se realiza según

$$g(k) = m \cdot r(k) \qquad (6)$$

donde m es una constante que determina la velocidad de convergencia, pero que necesita ser ajustada empíricamente en cada caso.

A pesar de su gran popularidad no prestaremos más atención al criterio LMS porque la experimentación nos ha permitido comprobar que este algoritmo cancela peor y con mayor número de parámetros que los que se explican en los siguientes apartados.

2.2.2 Criterio RLS

El criterio RLS (*Recursive Least Squares*) es la versión exacta del anterior algoritmo. Se utiliza el mismo criterio de optimización pero se calcula la solución exacta en cada iteración.

Como resultado se obtiene una expresión más compleja, que es

$$g(k) = \frac{\mathbf{P}(k-1)r(k)}{1 + r^T(k)\mathbf{P}(k-1)r(k)} \qquad (7)$$

en la que la matriz **P** (más concretamente su diagonal) caracteriza la incertidumbre de los parámetros estimados y también se calcula de forma recursiva según

$$\mathbf{P}(k) = \mathbf{P}(k-1) - g(k)r^T(k)\mathbf{P}(k-1) \qquad (8)$$

Como se observa en las ecuaciones (7) y (8) el algoritmo RLS tiene un producto de matrices y su coste computacional es cuadrático con respecto al número de parámetros a optimizar.

2.2.3 Criterio RLS simplificado

El algoritmo RLS no se utiliza, generalmente, en aplicaciones de procesado de señales porque exige velocidades de cálculo demasiado altas a los procesadores. Sin embargo, existen versiones simplificadas del algoritmo que, aunque convergen en un número mayor de iteraciones, comparten la optimalidad de RLS y son mucho menos costosas computacionalmente.

Aquí utilizaremos una simplificación atribuida a Kaczmarz y descrita en [1], normalmente referida como *projection algorithm*. El cálculo de $g(k)$ se realiza de la siguiente manera:

$$g(k) = c \cdot \frac{r(k-1)}{a + r^T(k-1)r(k-1)} \qquad (9)$$

donde a es una constante positiva y c debe cumplir que $0 \le c \le 2$. α no tiene más finalidad que la de evitar indeterminaciones cuando el vector de regresores se anula y c permite sintonizar la velocidad de convergencia del algoritmo.

2.3 ESTRUCTURA GENÉRICA DE LOS ALGORITMOS DE ADAPTACIÓN

Antes de entrar en los detalles de las implementaciones que se discuten posteriormente presentamos la estructura genérica de los algoritmos de adaptación, que se basa en las ecuaciones esbozadas hasta el momento.

Este algoritmo se ha de ejecutar con una frecuencia que varía en cada aplicación. En este caso, la frecuencia de muestreo es de 5000Hz, que corresponde a la frecuencia máxima de cancelación mutliplicada por 10.

El algoritmo de adaptación, en pseudolenguaje, queda de la siguiente manera:

1.- `Bucle de cancelación`
2.- `Adquisición:` $x(k)$ `y` $e(k)$
3.- `Formación del vector` $r(k)$
4.- `Cálculo de` $y(k) \rightarrow$ `ec. 3`
5.- `Escritura:` $y(k)$
6.- `Cálculo de` $g(k)\begin{cases}\text{RLS} \rightarrow \texttt{ec. 7 y 8} \\ \text{RLS simp.} \rightarrow \texttt{ec. 9}\end{cases}$
7.- `Cálculo de` $p(k+1) \rightarrow$ `ec. 5`
8.- `Fin del bucle`

3 CANCELACIÓN CON SIMULINK

En esta sección se describen los bloques de Simulink mediante los que se puede programar la DSP DS1003 de dSPACE para implementar el algoritmo de cancelación.

El proceso de generación de código nativo para el procesador TMS320C40 es muy sencillo cuando la aplicación que se desea ejecutar se crea en el entorno Simulink. Se parte de un modelo de Simulink que puede incluir cualquiera de los bloques usuales de este entorno y los bloques de la librería `rtilib` (r*eal time interface library*), que permiten realizar operaciones de lectura y escritura de señales con la DSP. Este modelo se transforma y transfiere a la DSP automáticamente mediante los menús del *Real Time Workshop*, que han de configurarse adecuadamente para cada procesador. A partir de este momento, la DSP ya está lista para ejecutar nuestra aplicación.

Para entender los algoritmos de cancelación programados a partir de modelos de Simulink bastará con comprender el funcionamiento de los bloques que implementan estos algoritmos.

3.2 Cálculo de la salida

Las líneas 3, 4 y 7 del algoritmo de cancelación, en la sección previa, llevan a cabo la lectura de datos y el cálculo de la salida correspondiente. Esta operación se realiza mediante el diagrama de bloques de la figura 2.

En concreto, la parte superior del diagrama prepara el vector $r(k)$ y la parte inferior mantiene y actualiza el vector de los parámetros, $p(k)$. De hecho, en esta figura no se hacen explícitas las operaciones de lectura y escritura para no complicarla, pero es evidente que habrá que incluir los bloques que realizan estas operaciones en el diagrama final.

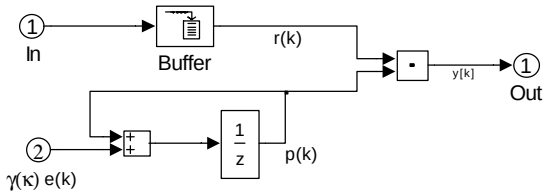

Figura 2: Diagrama de bloques de las ecuaciones (3) y (5)

Cabe destacar, también, que en la figura 2 se muestra un filtro FIR. En el caso de implementar un filtro IIR habría que mantener las secuencias de entradas y salidas en sendos bloques *buffer* y unirlos para obtener el vector $r(k)$.

3.3 Cálculo de $g(k)$

El cálculo de $g(k)$ es diferente para cada uno de los algoritmos de optimización tratados en la sección 2. Por este motivo los diagramas de bloques que implementan el cálculo son diferentes en cada caso.

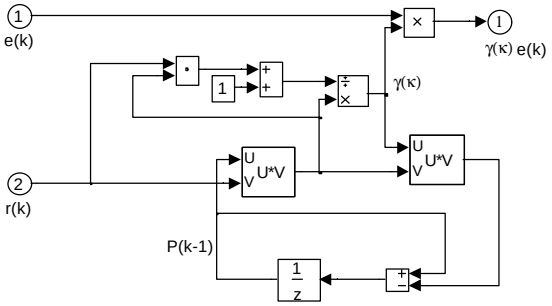

Figura 3: Cálculo de $g(k)\cdot e(k)$ en RLS

El algoritmo RLS contiene varios productos vectoriales, tal y como se aprecia en la figura 3. El bloque `U*V` de la izquierda calcula $\mathbf{P}(k\text{-}1)\cdot r(k)$ y el de la derecha realiza parte del cálculo de la ecuación (8).

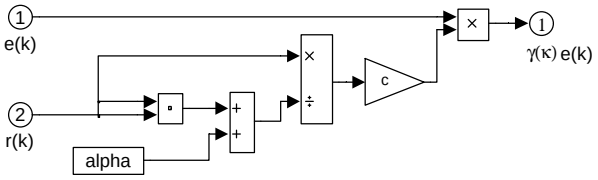

Figura 4: Cálculo de $g(k)\cdot e(k)$ en RLS simplificado

El diagrama de la figura 4 representa el algoritmo RLS simplificado. La diferencia entre este diagrama y el anterior es que los bloques que calculan y

actualizan **P**(*k*) han sido suprimidos. Además aparecen dos constantes, `c` y `alpha`, que corresponden a las constantes *c* y ***a*** descritas en el apartado 2.2.3.

Pues bien, el algoritmo de cancelación definitivo no es más que la correcta combinación de estos bloques con los que representan las operaciones de entrada y salida de la DSP. No ha sido necesario programar ni una sola línea de código.

4 CANCELACIÓN EN C

El algoritmo de cancelación en C es, esencialmente, el que se ha presentado en el apartado 2.3. Tiene dos puntos problemáticos que requieren un cierto cuidado a la hora de escribir el código. Se trata de las operaciones matemáticas y del tratamiento del tiempo en el programa.

4.1 Codificación de operaciones matemáticas

El primer paso que conviene dar, antes de codificar cualquier algoritmo de cálculo, es la simplificación de las expresiones matemáticas del mismo.

A menudo, es posible aplicar propiedades, por ejemplo sobre la regularidad de las matrices, que conducen a expresiones equivalentes menos costosas en tiempo de cálculo. Sin embargo, los algoritmos de la sección 2.2 no permiten realizar simplificaciones significativas.

Por este motivo, se ha puesto un énfasis especial en codificar el algoritmo de cancelación de la forma más eficiente posible. Para ello se han tomado las siguientes medidas: declaración estática de todas las variables, tratamiento circular de vectores y cálculo explícito de los índices de matrices y vectores.

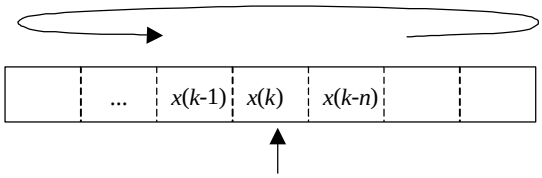

Figura 5: Un vector circular

De estas tres medidas, la más destacable es la del tratamiento circular de vectores. Esto permite mantener ordenada la información de un vector en todo momento sin tener que realizar ningún movimiento interno de datos. Para ello es necesario mantener un puntero al último dato introducido en el vector y de esta forma es posible recorrerlo en sentido temporal creciente o decreciente. En la figura 5 se muestra una representación de un vector circular de *n*+1 valores. Este vector se va rellenando de izquierda a derecha y se puede consultar en ambos sentidos, dependiendo de lo que nos convenga.

El cálculo explícito de los índices en matrices y vectores es una consecuencia de la utilización de vectores circulares. En las matrices se podía haber utilizado un esquema de direccionamiento indirecto, tal como se hace en librerías matemáticas como las *numerical recipes*, pero se ha optado por disponer la ocupación de memoria de forma lineal, para acceder a los valores de la forma más directa posible.

El siguiente código muestra la implementación de una de las rutinas del algoritmo de cancelación. Se trata del producto de una matriz y un vector. La variable `indice_r` mantiene el puntero al origen del vector apuntado por `v`, el resultado se escribe en la dirección apuntada por `v_res` y `nParams` es una constante que indica el número de valores del vector.

```
matrizXvector (float *m,*v,*v_res)
{
  int aux=indice_r;
  for(f=0;f<nParams;f++)
  {
    v_res[f]=0;
    for(c=0;c<nParams;c++)
    {
      aux=nParams*f+c;
      v_res[f] += m[aux]*v[indice_r];
      indice_r--;
      if (indice_r < 0)
            indice_r=nParams-1;
    }
  }
}
```

4.2 Tratamiento del tiempo

Todas las aplicaciones que se ejecutan en tiempo real o con la necesidad de cumplir unas restricciones temporales, han de llevar un control, ya sea implícito o explícito, del transcurso del tiempo. Éste es el caso de los algoritmos de control digital.

Para ello existen diversos mecanismos, aunque el más fiable y extensamente utilizado es el de servirse de un hardware externo que interrumpe al procesador a intervalos regulares. Estas interrupciones pueden ser capturadas por el sistema operativo, que informa a las aplicaciones que lo requieran del tiempo transcurrido. También pueden ser capturadas directamente por la propia aplicación, siendo entonces responsabilidad de ésta el control de los sucesos que tengan darse en cada instante.

La DSP DS1003 de dSPACE no sigue ninguno de estos dos esquemas, sino que utiliza uno intermedio.

Por un lado, es posible programar interrupciones para que se ejecuten a intervalos regulares dando, de esta manera, el control del tiempo y sus consecuencias a la aplicación. Pero también dispone de funciones para comprobar la temporización correcta de las interrupciones programadas.

El algoritmo esquematizado en el apartado 2.3 debe ejecutarse cada 0.2 milisegundos (es decir, a una frecuencia de 5000Hz) y no hay otras operaciones que se tengan que realizar, ni a velocidades menores ni como procesos de fondo (*background*). Así pues, la implementación en C del algoritmo de cancelación se ha realizado íntegramente dentro de la rutina de interrupción.

La aplicación se compone de dos partes: el programa principal, cuya tarea es inicializar todas las variables y activar las interrupciones; y la interrupción de control, que realiza la adquisición y entrega de datos y todo el cálculo necesario en cada iteración. En el caso de que el algoritmo de cancelación no pudiera ejecutarse en menos de 0.2 milisegundos el sistema se pararía dando un "error de sistema".

5 COMPARACIÓN DE LAS REALIZACIONES EN MATLAB Y EN C

En esta sección se comparan las implementaciones de los algoritmos vistos hasta el momento. Estamos interesados especialmente en evaluar la velocidad de ejecución de cada implementación. Esto es equivalente a evaluar el tamaño de los filtros puesto que, como ya se ha comentado, todo el algoritmo se ejecuta dentro de la rutina de interrupción.

El resultado que hemos obtenido ha sido sorprendente. El tamaño máximo de los filtros que cada implementación permite albergar se detalla en la tabla 1. En ella se observa que no es posible conseguir un incremento significativo con la programación en C, como se podía esperar a priori. Es decir que, aparentemente, no tiene especial relevancia implementar los filtros de cancelación activa en C o en Matlab.

	C	**Matlab**
RLS	12	7
RLS simplificado	51	55

Tabla 1: Tamaños máximos de los filtros

De hecho, si se considera el tiempo de desarrollo de esta aplicación en ambos lenguajes, queda patente la potencia de utilizar un lenguaje de cuarta generación. Este tipo de programación, alejado de las particularidades de una sintaxis rígida y de consideraciones que van más allá de los propios algoritmos (como la codificación de subrutinas muy básicas, las interrupciones, etc.) se está popularizando actualmente, pero al mismo tiempo tiene poca credibilidad entre la comunidad informática. Pues bien, aquí tenemos un ejemplo de las altas prestaciones que se puede lograr con esta metodología.

5.1 Resultados de cancelación

Aunque la finalidad de este artículo no sea la de evaluar la capacidad de cancelación de los algoritmos descritos, seguidamente se presentan unos resultados, sobretodo para poder comprobar que ambos realizan su tarea de forma equivalente.

En este ejemplo se utiliza como ruido una señal generada a partir de la superposición de 5 tonos puros, cuya representación frecuencial se da en la figura 6.

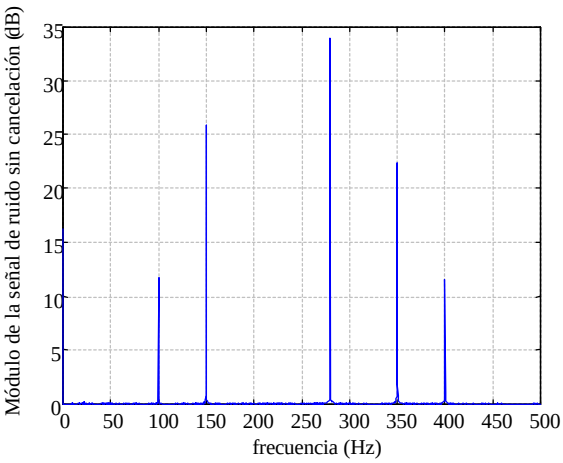

Figura 6: Representación frecuencial del ruido

La cancelación de este 'ruido' se ha llevado a cabo con el algoritmo RLS simplificado, utilizando un filtro de 48 coeficientes.

La gráfica de atenuación para la implementación en C corresponde a la figura 7 y la figura 8 muestra los resultados de la implementación en Matlab. Lo más destacable de ambas es que para las frecuencias de interés (100Hz, 150Hz, 280Hz, 350Hz y 400Hz) la atenuación está entre –30dB y –40dB. En el resto de frecuencias es posible observar valores muy dispares, incluso de amplificación, pero esto es debido a que la propia señal de entrada es insignificante para estas frecuencias.

En conclusión, se puede observar que ambas implementaciones trabajan de forma similar.

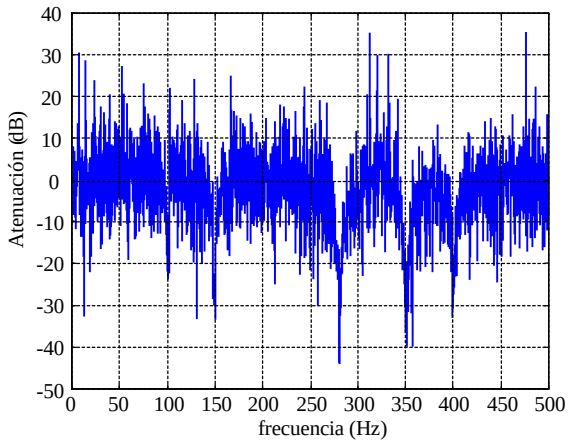

Figura 7: Atenuación en la implementación en C

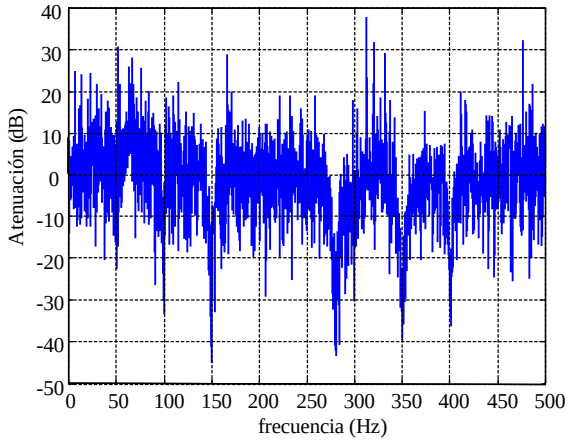

Figura 8: Atenuación en la implementación de Matlab

6 CONCLUSIONES

En este artículo se han comparado dos implementaciones de un mismo algoritmo de cancelación activa de ruido. Cada una de estas implementaciones ha sido programada utilizando lenguajes diferentes. Por un lado se ha utilizado el C, como un lenguaje de alto nivel que permite maximizar la eficiencia de los programas que con él se crean. Por otro lado, se ha utilizado una herramienta de cuarta generación (Simulink y Matlab) que permite al usuario concentrarse en los problemas del algoritmo más que en los de su implementación, pues le oculta a éste detalles de la misma.

En esta comparación se ha podido comprobar que la implementación en Matlab alcanza niveles de eficiencia (medida como el tamaño máximo del filtro de cancelación) comparables a los de la implementación en C.

Esta comparación, por otra parte, nos ha permitido valorar el esfuerzo de realizar un programa en C para una plataforma tan específica como la DSP que utilizamos. Hemos podido constatar que en esta aplicación, y suponemos que en otras, utilizar la plataforma al máximo de su capacidad es poco robusto. Es decir, que si conseguimos cancelar con *n* parámetros, deberíamos utilizar filtros mayores para garantizar robustez en la cancelación. Por ello es poco interesante realizar un esfuerzo importante en la implementación si éste sólo nos va a permitir aumentar el orden del filtro en unos pocos parámetros.

Sin embargo, existen otras soluciones que pueden ser muy fructíferas. La más destacable es la que contempla el cálculo de los parámetros *off-line*, [3]. En esta solución sí que es necesario realizar el programa de control en C, puesto que hay que distribuir de forma cuidadosa las tareas que se realizan en la interrupción y las que se van ejecutando entre tanto.

Agradecimientos

Este trabajo ha sido parcialmente financiado por la CICYT ref. TAP 1999-0748 y por la Generalitat de Catalunya. Los autores son miembros del grupo de investigación consolidado SAC (ref. 1999-SGR-00134 SAC).

Referencias

[1] Åström, K.J., Wittenmark, B. (1989). *Adaptive Control*. Addisson – Wesley.

[2] Kuo, S.M., Morgan, D.R. (1996). *Active Noise Control Systems*. John Wiley & Sons.

[3] Walter, E., Pronzato, L. (1997). *Identification of Parametric Models*. Masson.