Capítulo 4

Implementación en Matlab

4.1. Consideraciones Iniciales

En este capitulo mostraremos el trabajo desarrollado en MATLAB y SIMULINK para simular y probar el desempeño de los algoritmos descritos en el capítulo anterior. Se utilizara el software *MATLAB R2007a* para desarrollar filtros de Cancelación Activa del Ruido usando los algoritmos **LMS** y **RLS**.

Para poder comparar de una manera más efectiva el desempeño de los algoritmos mencionados se requiere establecer parámetros fijos en la simulación del filtrado. Para ello se usara como fuente primaria un tono senoidal con variación lineal de frecuencia, con ecuación:

$$x[n] = \sin(2\pi t^2) \tag{4.1}$$

La cual se puede observar en la figura 4.1, este será la señal que lleve la información deseada; por otra parte se usaran los propios algoritmos de MATLAB para generar una señal aleatoria que funcione como ruido blanco, la cual será la fuente secundaria de nuestros filtros.

Las condiciones iniciales para todos los casos serán con inicio de cero; esto es tanto la señal de entrada, como el vector de coeficientes tendrán sus valores iniciales en cero.

Los algoritmos serán simulados con un orden de $L = 6$ para las simulaciones en MATLAB y con $L = 24$ para los algoritmos en SIMULINK, y un valor de $\mu = 0.1$ para el caso del algoritmo *LMS* y $\lambda = 1$ para el algoritmo *RLS*.

4.2. Algoritmo LMS

El código del algoritmo LMS en MATLAB es el siguiente cuadro:

```
1 %Simulación del algoritmo LMS para Control Activo del Ruido
```

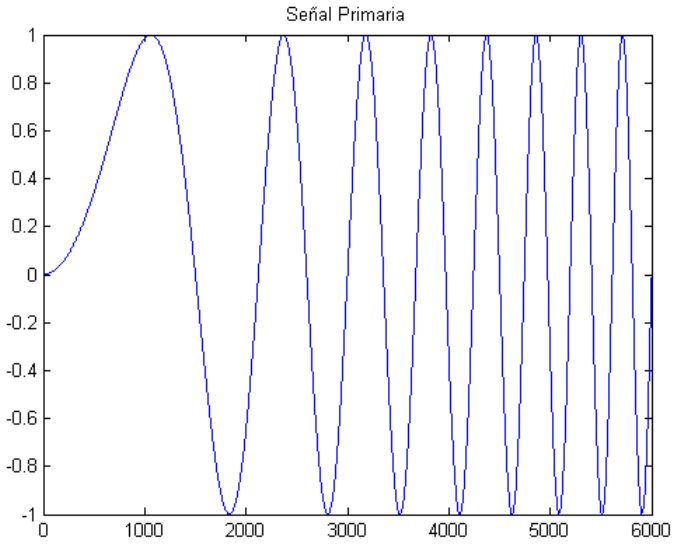


Figura 4.1: Gráfica de la señal primaria.

```
2  %
3  clear all;
4  close all;
5  clc;
6
7  %Establecemos el orden del sistema
8  refGain = 1;
9  worder = 6;
10 %La señal de referencia
11 N = 6000;
12 t=1:N;
13 signal = sin(2*pi.*t.*t/N/N*8);
14 wreal = randn(1,worder);
15 %Y el ruido a cancelar
16 anoise = randn(1,N);
17 ref = conv(anoise,wreal);
18 primary = signal + ref(1:length(signal));
19 fref = anoise*refGain;
20 w(1,:) = zeros(1,worder);
21 mu = .1;
22 %Condiciones iniciales = 0
23 frefpad = [zeros(1,worder -1) fref];
24 %se inicia el algoritmo
25 for n = 1:N;
26     %Movemos el contador para poder inciar en el orden correcto en caso de
27     %iniciar con cero
28     m = n + worder -1;
29     frefblock = frefpad(m-worder+1:1:m)';
30     refP(n) = w(n,:)*(frefblock);
```

```
31      output(n) = primary(n) - refP(n);
32      w(n+1,:) = w(n,:) + mu.*frefblock'.*output(n);
33  end;
34  %Se llama la función de graficación...
35  lms_graph
36
37  %Calculando el SNR
38  snr = 10*log10(sum(signal.^2) ./ sum(anoise.^2))
39  snr2 = 10*log10(sum(signal.^2) ./ sum(ref.^2))
```

Como podemos observar en el código mostrado, se siguió el método establecido en la sección 3.3. Esto es, primero se establece el orden del sistema y se generan las señales que serán usadas en el proceso, esto es la señal de prueba (4.1) y la señal de ruido; que en este caso es una señal de ruido blanco con un valor normalizado. Posteriormente se establecen las condiciones iniciales en cero para el vector de coeficientes y el valor de μ también es establecido (en 0.1 para este caso).

Una vez establecido lo anterior se procede a iniciar el algoritmo como se muestra en las ecuaciones (3.28), (3.29) y (3.30). El algoritmo termina en esta parte el código restante se usa para lograr las gráficas de resultado.

En la figura 4.2 se muestra la señal de entrada, la cual es la suma de la señal deseada más el ruido, la señal de salida o la señal filtrada y se gráfica el error cuadrático medio; todas graficadas con respecto al tiempo n

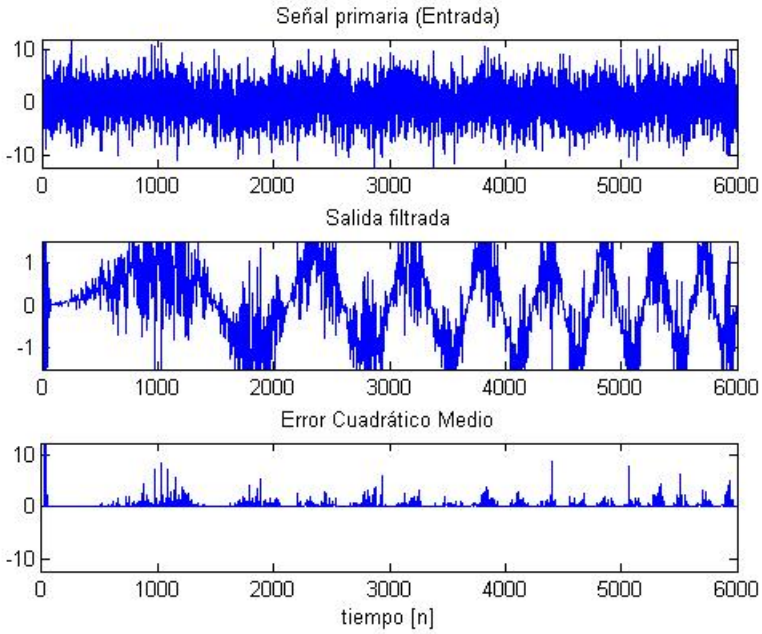


Figura 4.2: Resultado del algoritmo LMS (SNR=13.79 [db])

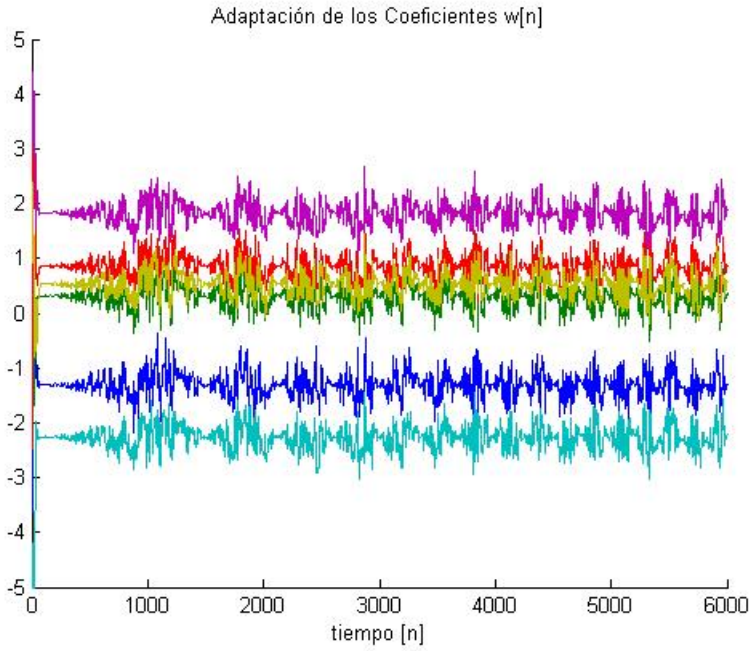


Figura 4.3: Proceso de adaptación de los coeficientes para el algoritmo LMS

Recordemos que la simulación se realizó con un sistema de orden $L = 6$, lo que significa que el sistema tendrá seis coeficientes, el valor de estos coeficientes es actualizado cada iteración con base al algoritmo LMS, la 'evolución' de estos seis coeficientes con respecto al tiempo n se muestra en la figura 4.3.

Haciendo una comparación de la señal de salida en la figura 4.2 y la señal original en la figura 4.1 observamos que el algoritmo LMS fue capaz de obtener la forma de onda de la señal original, sin embargo, esta todavía contiene mucho ruido lo cual puede ser observado facilmente en la gráfica del MSE.

4.2.1. Simulación en SIMULINK

Para la simulación del control activo del ruido con el algoritmo LMS usamos el modelo mostrado en la figura 4.4 el cual fue realizado en SIMULINK[19]. Observamos en el modelo que se uso el bloque "Chirp Generator" el cual genera una señal parecida a la usada en la prueba de anterior, esto es, una señal senoidal con variación lineal de frecuencia. Para este caso, la señal comienza con una frecuencia de $10Hz$ y alcanza un máximo de $30Hz$. Otro bloque que requiere de nuestra atención es el bloque del generador de ruido "Random Source" el cual se encarga de generar la señal contaminante, esta señal de ruido es pasada por un filtro paso bajas antes de ser sumada a la señal primaria; esto con el objetivo de asegurarnos que el contenido espectral de ruido se centre en las frecuencias bajas.

En la figura 4.5 se pueden observar las señales involucradas en el proceso, estas son (de

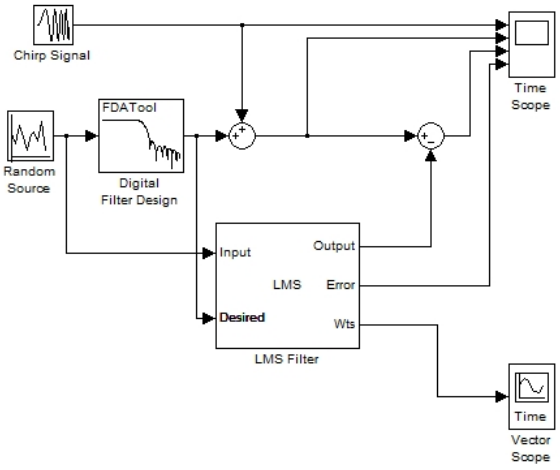


Figura 4.4: Modelo de un filtro con CAR y el algoritmo LMS en SIMULINK

arriba a abajo): la señal original, señal primaria (señal original más ruido), la señal de salida (filtrada) y la señal de error. Es importante notar que el algoritmo tiene un desempeño aceptable presentandose algunos errores cuando la señal original hace cambios de frecuencia.

Por último se graficó el vector de coeficientes $w[n]$ (figura 4.6 en el cual podemos observar el estado de los 24 coeficientes una vez que el sistema ya se estabilizó.

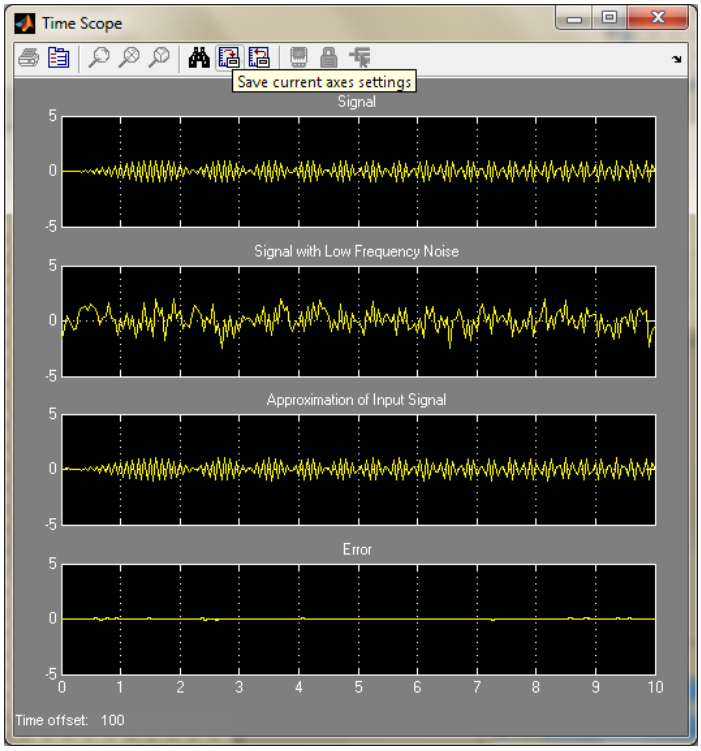


Figura 4.5: Gráficas de las señales de entrada (original y ruido) y salidas (señal filtrada y error) del modelo en SIMULINK

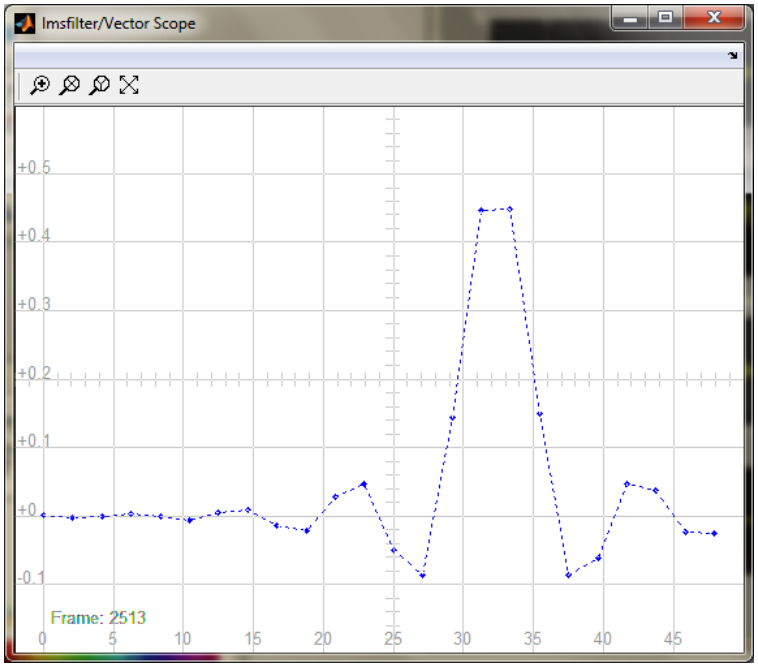


Figura 4.6: Vector de coeficientes del filtro con $L = 24$ coeficientes (Modelo en SIMULINK)

4.3. Algoritmo RLS

Para el código del algoritmo *RLS* se utilizó una estructura similar al *LMS*, como ya se mencionó al principio de este capítulo se utilizan los mismos parámetros; esto es, la misma señal original e igual orden del sistema ($L = 6$). El código se muestra a continuación.

```
%Simulación del algoritmo RLS para Control Activo del Ruido
%
clear all;
close all;
clc;

%Establecemos el orden del sistema
refGain = 1;
worder = 6;
%La señal de referencia
N = 6000;
t=1:N;
signal = sin(2*pi.*t.*t/N/N*8);
wreal = randn(1,worder);
%Y el ruido a cancelar
anoise = randn(1,N);
ref = conv(anoise,wreal);
primary = signal + ref(1:length(signal));
fref = anoise*refGain;
%El valor de lambda
lambda = 1;
%Condiciones iniciales = 0
w(1,:) = zeros(1,worder);
init = 100;
rinv = diag( ones(1,worder) * init );
frefpad = [zeros(1,worder -1) fref];
%se inicia el algoritmo
for n = 1:N;
    %Movemos el contador para poder inciar en el orden correcto en caso de
    %iniciar con cero
    m = n + worder -1;
    frefblock = frefpad(m-worder+1:1:m)';
    refP(n) = w(n,:)*(frefblock);
    output(n) = primary(n) - refP(n);
    k = lambda * rinv * frefblock / (1 + lambda*frefblock'*rinv*frefblock);
    w(n+1,:) = w(n,:) + k'*output(n);
    rinv = lambda*rinv - lambda*k*frefblock'*rinv;
end;
%Se llama la función de graficación...
rls_graph

%Calculando el SNR
snr = 10*log10(sum(signal.^2) ./ sum(anoise.^2))
snr2 = 10*log10(sum(signal.^2) ./ sum(ref.^2))
```

Podemos observar que este código comparte con el código del algoritmo LMS la inicialización de los parámetros de simulación, así como la nomenclatura de algunas señales. Esto con el objetivo de facilitar la comparación entre ambos códigos. El algoritmo para el cálculo del RLS fué tomado de las ecuaciones (3.69) a la (3.73).

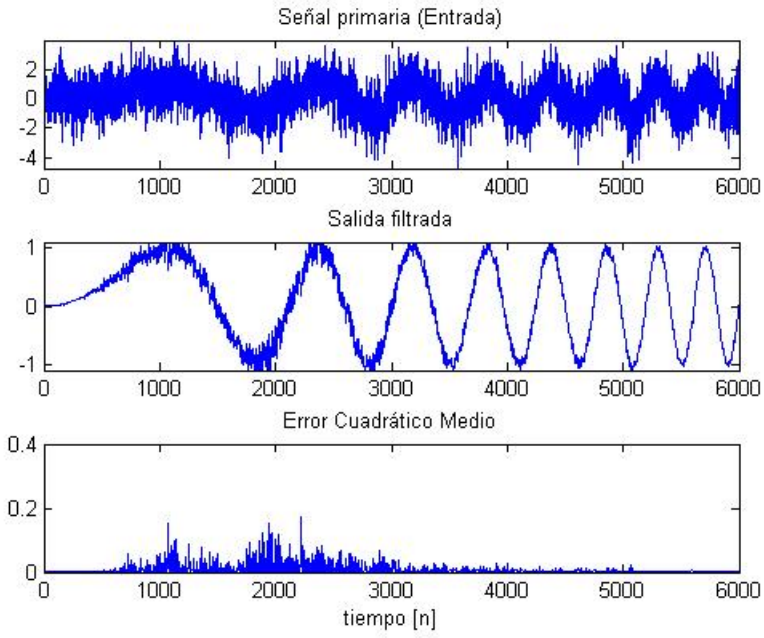


Figura 4.7: Resultado del algoritmo RLS (SNR=12.93 [db])

Una vez que se ejecutó el código observamos en la figura 4.7 los resultados. Aquí se grafican la señal de entrada (señal original más el ruido), la señal filtrada y el error cuadrático medio. Se nota inmediatamente la diferencia entre los dos algoritmos, al ofrecer el algoritmo *RLS* un mejor desempeño en el filtrado del ruido en la señal de salida, la cual a pesar de tener algo de ruido en ella es considerablemente menor que en el caso anterior (ver figura 4.2). Además de ofrecer un error cuadrático medio nulo.

Como en el algoritmo anterior la simulación es un sistema de orden $L = 6$, lo que significa que el sistema tendrá seis coeficientes, el valor de estos coeficientes es actualizado cada iteración con base al algoritmo RLS, la 'evolución' de estos seis coeficientes con respecto al tiempo n se muestra en la figura 4.8. Notamos que en este caso los coeficientes son más estables y convergen mucho más rápido que con el algoritmo LMS.

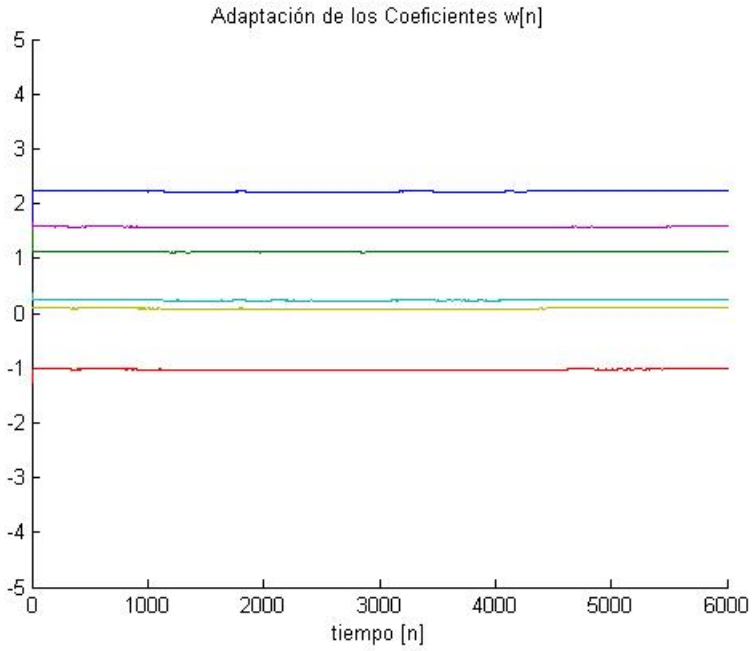


Figura 4.8: Proceso de adaptación de los coeficientes para el algoritmo RLS

4.3.1. Simulación en SIMULINK

Para la simulación del control activo del ruido con el algoritmo RLS usamos el modelo mostrado en la figura 4.9 el cual fue realizado en SIMULINK.

Al igual que con el código en MATLAB, la estructura de este modelo es similar al modelo para el algoritmo LMS, pudiendo destacar el uso de los bloques que generan la señal, el "Chirp Generator" que genera la señal senoidal con variación lineal de frecuencia. De esta manera la señal generada tiene una frecuencia inicial de $10Hz$ y alcanza una frecuencia máxima de $30Hz$. También conservamos el bloque del generador de ruido "Random Source" cuya señal es pasada por un filtro paso bajas antes de ser sumada a la señal primaria; esto con el objetivo de asegurarnos que el contenido espectral de ruido se centre en las frecuencias bajas.

En la figura 4.10 se pueden observar las señales involucradas en el proceso, estas son (de arriba a abajo): la señal original, señal primaria (señal original más ruido), la señal de salida (filtrada) y la señal de error. Notamos que a diferencia del modelo del algoritmo LMS, este modelo presenta mayor precisión al filtrar la señal primaria, con un error nulo, mayor velocidad de convergencia y mayor estabilidad en los coeficientes, los cuales pueden ser observados en la figura 4.10.

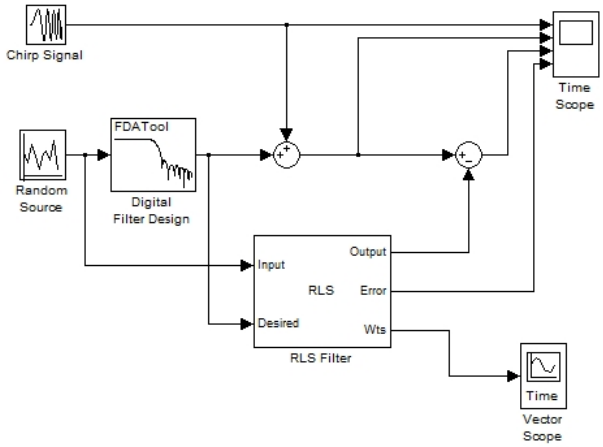


Figura 4.9: Modelo de un filtro con CAR y el algoritmo RLS en SIMULINK

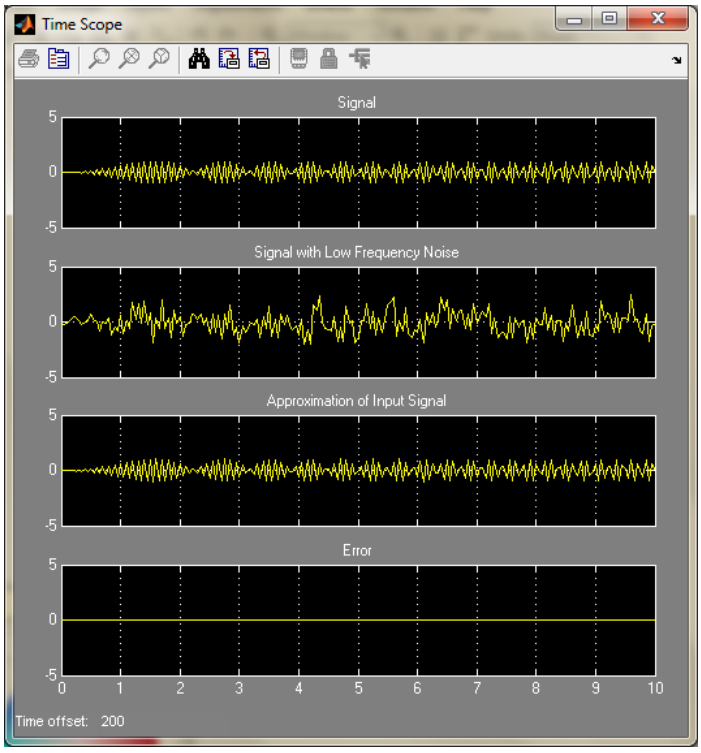


Figura 4.10: Gráficas de las señales de entrada (original y ruido) y salidas (señal filtrada y error) del modelo en SIMULINK

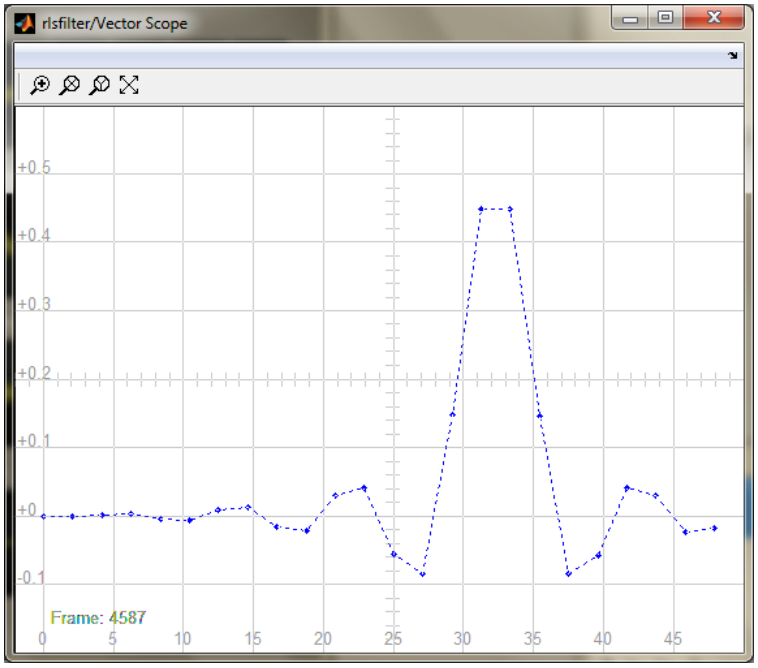


Figura 4.11: Vector de coeficientes del filtro con $L = 24$ coeficientes (Modelo en SIMULINK)

4.4. Conclusiones

De las simulaciones en MATLAB de los filtros adaptables con algoritmos LMS y RLS encontramos que se presentaron marcadas diferencias entre ambos, después de procesar la señal primaria encontramos que el algoritmo RLS se adapta mejor al ruido presente para lograr eliminar la mayor parte de este, esto es mucho más evidente al comparar las gráficas del error cuadrático medio en las figuras 4.2 y 4.7. Además la velocidad de adaptación del algoritmo RLS también son mucho mejores, comparando las figuras 4.3 y 4.8 podemos observar que el algoritmo tiene un mejor comportamiento en la adaptación de los coeficientes del filtro durante todo el proceso, al alcanzar los valores optimos ($w^o[n]$) mucho más rápido que el algortimo LMS y con menos variaciones durante todo el proceso.

Para el caso de los modelos en SIMULINK, con un orden de filtro mucho mayor ($L = 24$) al del caso en MATLAB (con $L = 6$), la diferencia de comportamiento estre ambos algoritmos es muy pequeña; al alcanzar el estado estable ambos algoritmos son capaces de minimizar el ruido de la señal primaria a niveles satisfactorios, presentando una ligera desventaja el algoritmo LMS el cual exhibe pequeños niveles de error cuando la señal original hace cambios de frecuencia lo cual es evidencia de la baja velocidad de adaptación característica del algoritmo.

En conclusión podemos inferir que el algoritmo LMS ofrece una opción simple para un proceso de adaptación, la calidad de la cancelación de las señales no deseadas es baja. De manera contraria, el algoritmo RLS ofrece mayor calidad en la atenuación de las señales no deseadas a cambio de mayor peso computacional en la adaptación de los coeficientes del filtro.

4.5. Control Activo del Ruido en el Dominio de la Frecuencia

Una vez que analizamos el comportamiento de los algoritmos LMS y RLS, simularemos el comportamiento de dos algoritmos que usan análisis en frecuencia para eliminar el ruido de la señal primaria, para estos casos se utilizó la transformada Discreta de Fourier y transformada Discreta de Cosenos para lograr el efecto deseado. Como en el caso del CAR en el dominio del tiempo, se realizaron 2 simulaciones para cada transformada, una utilizando código en MATLAB y la otra con un modelo en SIMULINK.

En la simulación de MATLAB, usamos la misma señal de entrada que en los experimentos de CAR en el dominio del tiempo; esto es, una señal *chirp* o tono senoidal con variación lineal en frecuencia, como es descrito en la ecuación 4.1 y que se muestra en la figura 4.1. Para las simulaciones en SIMULINK se utilizaron señales de audio previamente grabadas y digitalizadas, la señal primaría es una grabación de voz, mientras que la señal contaminante es la grabación de una trompeta, ambas señales en archivos WAV a 176 *kbps*.

Antes de realizar la simulación en SIMULINK, las señales de entrada son adaptadas para

poder ser usadas en el modelo; esto se realiza por medio de la instrucción *wavread*, la cual lee el contenido de un archivo WAV y coloca los valores en el espacio de trabajo de MATLAB. Las señales de entrada "voz.wav" y "trompeta.wav" fueron procesadas con un código especial el cual puede ser observado en el apéndice A.1.

Una vez que se tienen las señales de entrada preparadas, podemos proceder con la simulación en SIMULINK, los resultados pueden ser observados a continuación.

4.5.1. Transformada Coseno Discreta

Para la Transformada Coseno Discreta, el código utilizado de MATLAB es el siguiente:

```
clc; clear all;
D  = 16;                          % Retraso de las muestras
ntr= 1000;                        % Numero de iteraciones
s  = chirp(randn(1,ntr+D));       % Señal de entrada (chirp)
n  = 0.1*(randn(1,ntr+D));        % Señal de Ruido
r  = s+n;                         % Señal recibida
x  = r(1+D:ntr+D);                % Señal de entrada
d  = s(1:ntr);                    % Señal deseada
L  = 32;                          % Orden del filtro
mu = 0.01;                        % Coeficiente del filtro adaptable
ha = adaptfilt.tdafdct(L,mu);     % Cálculo del filtro
[y,e] = filter(ha,x,d);           % Se realiza el filtrado
%Graficando las señales de entrada y salida
figure
subplot(3,1,1)
plot(1:ntr,real(d))
title('Señal deseada')
subplot(3,1,2)
plot(1:ntr,real(y));
title('Señal de salida (recuperada)')
subplot(3,1,3)
plot(1:ntr,abs(e));
title('Error')
```

Al ejecutar el código anterior obtenemos como resultados la gráfica que se muestra en la figura 4.12.

Como podemos observar en la figura la gráfica de la parte superior muestra la señal de entrada al sistema, esto es la señal original más la señal ruidosa. En la gráfica de enmedio se observa la señal recuperada por el filtro adaptable notando que el filtro no recupera inmediatamente la señal, esto es más evidente al observar la ultima gráfica, marcada como la señal de error; en ella observamos como el error comienza con niveles muy elevados (casi 100 %) al inicio del proceso y sin embargo comienza a disminuir de manera pareja esta que el sistema se estabiliza.

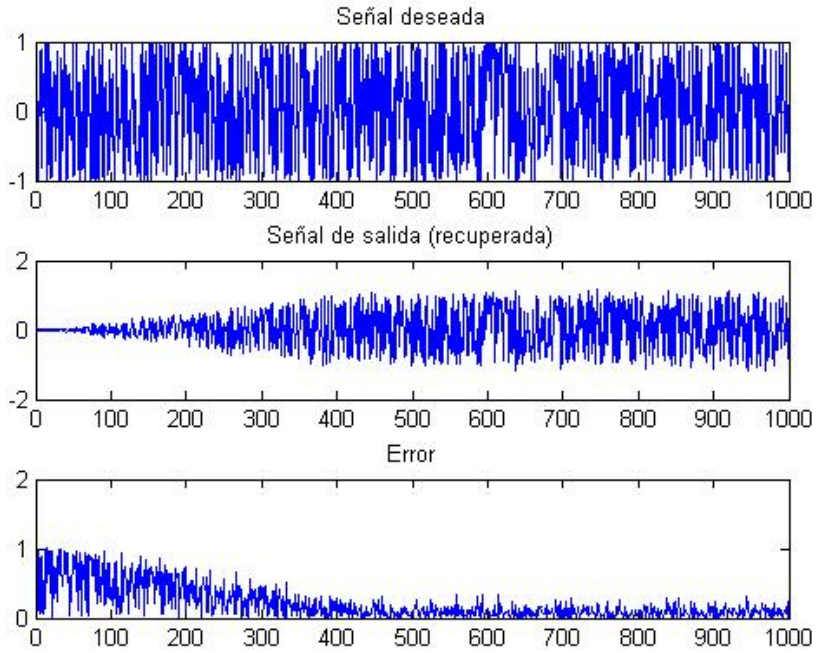


Figura 4.12: Gráficas de la Simulación en MATLAB

Una vez que observamos el comportamiento en MATLAB pasamos al modelo en SIMULINK utilizando la configuración mostrada en la figura 4.13. Las muestras de las señales primaria y de ruido son tomadas del espacio de trabajo de MATLAB y procesadas por simulink para formar a travez del operador suma la señal secundaria. Ambas señales son alimentadas a un buffer de 16 bits para posteriormente ser analizadas por el filtro por medio de la transformada coseno digital. A la salida de la señal de salida y de error se coloca un "Unbuffer" (de 16 bits) para poder realizar la visualización de las señales por medio de un osciloscopio.

En cada uno de los cuatro osciloscopios se grafica una señal diferente como se muestra en la figura 4.14 Podemos observar la señal original (fig.4.14a) seguida de la señal deseada (señal original más ruido) en la fig. 4.14b; para posteriormente observar la señal de salida y error (figs. 4.14c y 4.14d respectivamente).

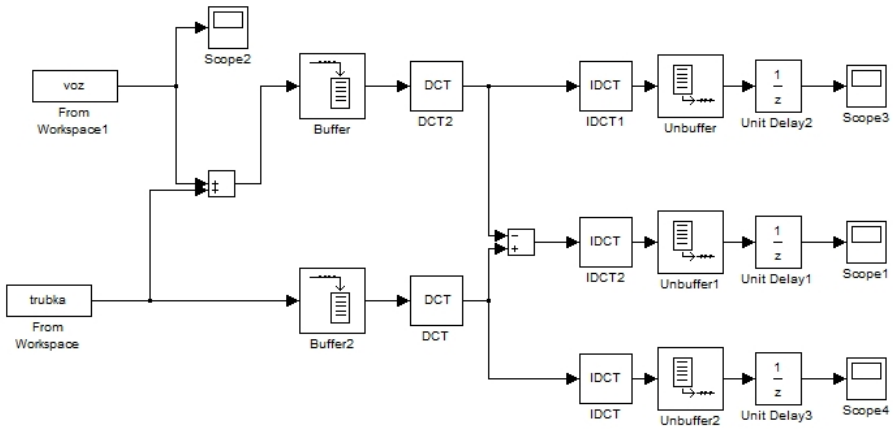
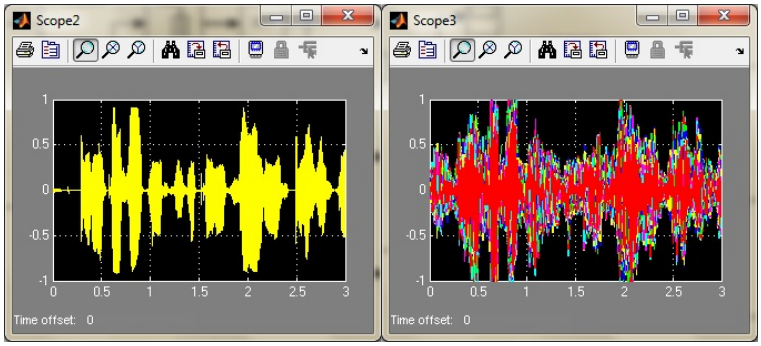
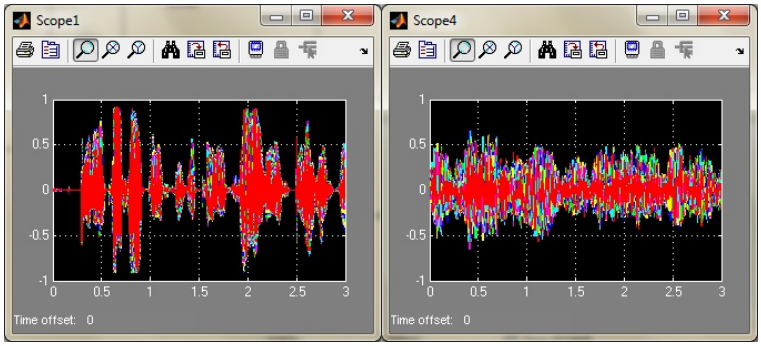


Figura 4.13: Modelo del filtro con análisis en frecuencia por medio de la Transformada Discreta de Cosenos.



(a) Señal Original **(b)** Señal Contaminada



(c) Señal de Salida **(d)** Señal de Error

Figura 4.14: Señales de salida del modelo anterior.

4.5.2. Transformada Discreta de Fourier

Posteriormente pasamos a la simulación usando la Transformada Discreta de Fourier utilizando e siguiente código en la simulación en MATLAB

```
clc; clear all;
D  = 16;                          % Retraso de las muestras
ntr= 1000;                        % Numero de iteraciones
s  = chirp(randn(1,ntr+D));       % Señal de entrada (chirp)
n  = 0.1*(randn(1,ntr+D));        % Señal de Ruido
r  = s+n;                         % Señal recibida
x  = r(1+D:ntr+D);                % Señal de entrada
d  = s(1:ntr);                    % Señal deseada
L  = 32;                          % Orden del filtro
mu = 0.01;                        % Coeficiente del filtro adaptable
ha = adaptfilt.tdafdft(L,mu);
[y,e] = filter(ha,x,d);
%Graficando las señales de entrada y salida
figure
subplot(3,1,1)
plot(1:ntr,real(d))
title('Señal deseada')
subplot(3,1,2)
plot(1:ntr,real(y));
title('Señal de salida (recuperada)')
subplot(3,1,3)
plot(1:ntr,abs(e));
title('Error')
```

Despues de ejecutar el código obtenemos los resultados en la forma de la gráfica 4.15

Al igual que en el caso de la Transformada Coseno Discreta en la primera gráfica encontramos la señal deseada (original más ruido), debajo de ella se grafica la señal de salida del filtro que corresponde a la señal recuperada, notamos que el filtro con Transformada Discreta de Fourier actúa más rápido que el filtro con Transformada Coseno Discreta este se aprecia mejor al comparar las señales de error que genera cada simulación donde observamos que el filtro de CAR con TDF alcanza mucho más rápido la estabilidad.

Para el modelo en SIMULINK utilizamos una configuración similar al caso anterior, utilizando como señales de entrada la información proveniente del espacio de trabajo de MATLAB, creando la señal deseada previamente a la acción de filtrado. La configuración usada se muestra en la figura 4.16.

En cada uno de los cuatro osciloscopios se grafica una señal diferente como se muestra en la figura 4.17

Podemos observar la señal original (fig.4.17a) seguida de la señal deseada (señal original más ruido) en la fig. 4.17b; para posteriormente observar la señal de salida y error (figs. 4.17c y 4.17d respectivamente).

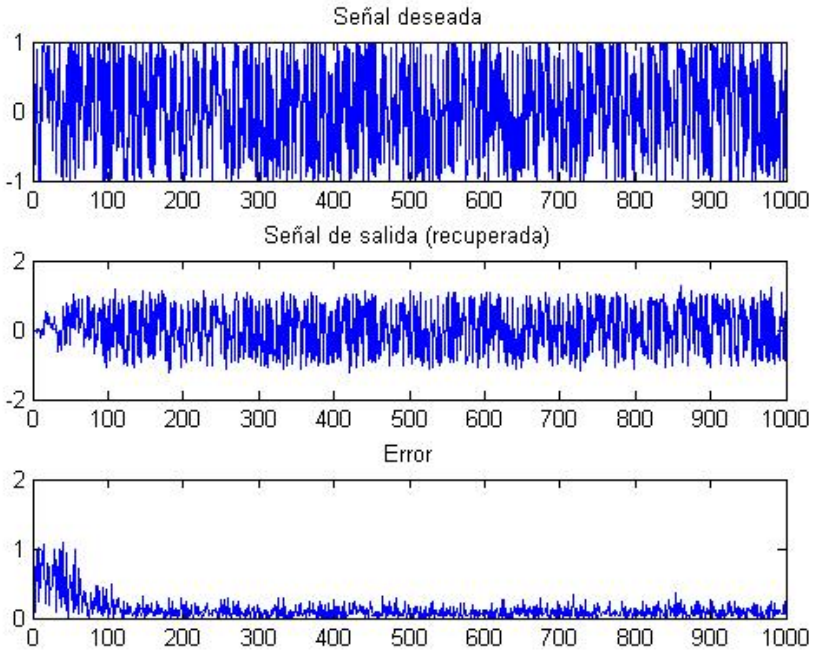


Figura 4.15: Gráficas de la Simulación en MATLAB

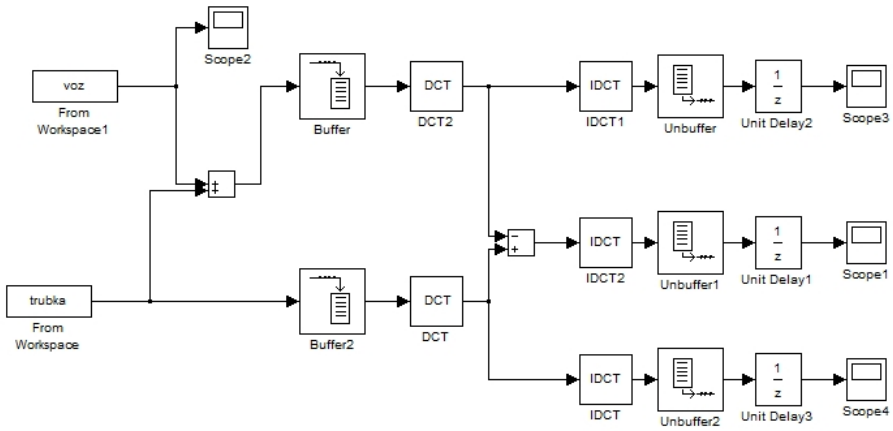
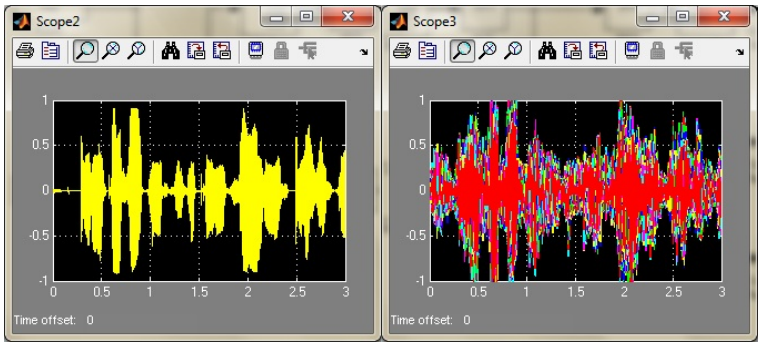
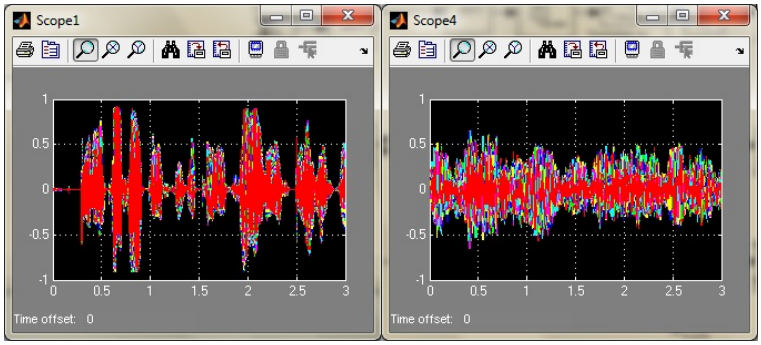


Figura 4.16: Modelo del filtro con análisis en frecuencia por medio de la Transformada Discreta de Fourier.



(a) Señal Original **(b)** Señal Contaminada



(c) Señal de Salida **(d)** Señal de Error

Figura 4.17: Señales de salida del modelo anterior.