CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA
CELSO SUCKOW DA FONSECA – CEFET/RJ

DIRETORIA DE PESQUISA E PÓS-GRADUAÇÃO
COORDENADORIA DO PROGRAMA DE PÓS-GRADUAÇÃO EM TECNOLOGIA

DISSERTAÇÃO

APLICAÇÃO DO PROCESSAMENTO DIGITAL DE SINAIS AO CONTROLE ATIVO DE RUÍDO EM BAIXAS FREQUÊNCIAS

Milton Moreira Pinheiro da Fonseca

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO PROGRAMA DE PÓS-GRADUAÇÃO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM TECNOLOGIA

Prof. Paulo Lucio Silva de Aquino D.Sc.
ORIENTADOR

RIO DE JANEIRO, RJ – BRASIL
MAIO / 2007

MILTON MOREIRA PINHEIRO DA FONSECA

APLICAÇÃO DO PROCESSAMENTO DIGITAL DE SINAIS AO CONTROLE ATIVO DE RUÍDO EM BAIXAS FREQUÊNCIAS

Dissertação apresentada ao Centro Federal de Educação Tecnológica CFET/RJ como requisito parcial necessário a obtenção do grau de Mestre em Tecnologia.

Orientador: Paulo Lúcio S. de Aquino D.Sc.

F676 Fonseca, Milton Moreira P.inheiro da
Aplicação do processamento digital de sinais ao controle ativo de ruído em baixas freqüências / Milton Moreira Pinheiro da Fonseca – 2007.
x, 109f. +anexos: il (algumas cor), grafs. tabs.; enc.

Dissertação (mestrado) Centro Federal de Educação Tecnológica Celso Suckow da Fonseca, 2007.
Bibliografia: f. 107-108

1. Processamento de sinais- Técnicad digitais 2. Controle de Ruído 3. Algoritmos I. Título

CDD 621.3822

À minha querida esposa e meus queridos pais e família.

Agradecimentos

- Ao Professor Paulo Lúcio Silva de Aquino, meu orientador, pela dedicação, experiência e conhecimento que me passou para a elaboração deste trabalho e no decorrer de vários anos de trabalho.
- Ao Professor Carlos Henrique Figueiredo Alves pelo conhecimento, profissionalismo, sugestões e críticas.
- Ao Professor Darcy das Neves Nobre pela presteza e gentileza de ter participado da Banca de Avaliação.
- Ao Centro Tecnológico do Exército - CTEx por ter-me confiado o tempo necessário para desenvolvimento e conclusão do trabalho.
- Ao Engenheiro João Ernesto da Costa Ferreira e ao Major José Cerdeira Gonzáles, chefes que compreenderam os momentos de dificuldade.
- A todos os Professores da Pós-graduação do CEFET-RJ, que, de alguma forma, contribuíram para o engrandecimento de meu conhecimento pessoal.
- Aos funcionários da Secretaria do CEFET-RJ e aos funcionários da limpeza.

Resumo da dissertação submetida ao PPTEC/CEFET/RJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Tecnologia (M.T.).

APLICAÇÃO DO PROCESSAMENTO DIGITAL DE SINAIS AO CONTROLE ATIVO DE RUÍDO EM BAIXAS FREQUÊNCIAS

Milton Moreira P. da Fonseca

Maio / 2007

Orientador: Paulo Lúcio Silva de Aquino D. Sc.

Departamento: PPTEC

Esse trabalho tem por objetivo propor um sistema de controle ativo de ruídos, em baixas freqüências, para pequenos ambientes confinados. O sistema pode ser empregado em sistemas de ventilação, cabines automotivas, fones de ouvido, aeronaves e veículos militares dentre outros. Tal sistema faz uso das técnicas do Processamento Digital de Sinais, DSP, identificação de sistemas e do Controle. Para tanto, primeiro é mostrada a solução teórica do problema no que diz respeito à utilização de filtros digitais adaptativos baseados no algoritmo dos mínimos quadrados e suas variantes. Depois são realizadas simulações numéricas e finalmente é implementado um sistema mínimo para a comprovação prática dos resultados teóricos. Esse sistema mínimo é composto por um duto, alto-falantes como atuadores e microfones como sensores. O controle ativo do ruído de um motor foi analisado empregando-se um controlador digital feedfoward do tipo SISO (“single input single output”) baseado no algoritmo FXLMS. O processamento é realizado por meio do sistema de desenvolvimento DSK6713 e a linguagem de programação adotada é o C++.

Palavras-chave: ruído, filtros, algoritmos

Abstract of dissertation submitted to PPTEC/CEFET/RJ as partial fulfillment of the requirements for the degree of Master in Technology (M.T.).

DIGITAL SIGNAL PROCESSIG APPLIED TO ACTIVE NOISE CONTROL IN LOW FRQUENCIES

Milton Moreira P. da Fonseca

May / 2007

Supervisor: Paulo Lúcio Silva de Aquino D. Sc.

Department: PPTEC

ABSTRACT

This paper proposes a system for active noise cancellation, in low frequencies, for small confined spaces. The system can be employed in ventilation systems, automotive cabins, aircrafts and military vehicles among others. The system makes use of the techniques on Digital Signal Processing, DSP, System Identification and Control. In this sense, the first step is the study of the theoretical solution of the problem in respect to the use of adaptive digital filters, especially the FXLMS. Then, numeric simulations are shown and, finally, a minimum system is implemented for the practical proof of the theoretical results. That minimum system is composed by a duct, speakers as actuators and microphones as sensors. Active control of fan noise in short ducts was then investigated using a single input / single output feedforward adaptive digital controller based on the Filtered-X LMS algorithm. The processing is accomplished through the DSK6713 development system and the adopted programming language was C++.

Keyword: noise, filters, algorithms

Lista de Figuras

Lista de Tabelas

SUMÁRIO

INTRODUÇÃO

O controle do ruído acústico envolve, tradicionalmente, métodos passivos, tais como barreiras acústicas e silenciadores para atenuar o ruído. Essas técnicas utilizam os conceitos da mudança de impedância acústica e da dissipação de energia empregando-se materiais absorventes acústicos. Esses métodos não se mostram eficazes quando aplicados ao cancelamento de ruídos de baixa freqüência (menor que 500Hz). A técnica para superar este problema é o Controle Ativo de Ruídos Acústicos - CARA, que modifica o campo acústico por meio de atuadores e sensores eletroacústicos e de um sistema eletrônico de controle.

O objetivo desta tecnologia é conseguir o cancelamento de ruídos indesejáveis que venham a deteriorar as comunicações, degradar a inteligibilidade da palavra ou mesmo evitar danos físicos e fisiológicos aos humanos.

A teoria envolvida baseia-se no princípio da superposição destrutiva: dois campos sonoros, a origem do ruído e a fonte secundária de interferência, se sobrepõem para cancelar, em uma região ou pontualmente, o ruído indesejado. O cancelamento, em termos práticos, deve ser entendido como minimização, e valores de até 20 dB vêm sendo alcançados nas pesquisas mais recentes.

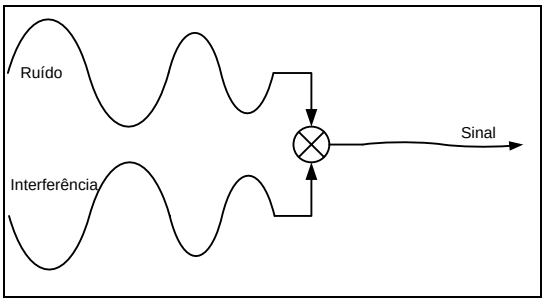
O método ativo de controle de ruídos utiliza um dispositivo acústico, um atuador secundário no sistema, gerando o anti-ruído de igual amplitude e fase oposta (180º), necessário para o cancelamento do ruído original. Os campos sonoros se combinam acusticamente, resultando no cancelamento acústico dos mesmos.

1º. Campo Sonoro - Origem do Ruído:

- Motores
- Ventiladores
- Transformadores
- Compressores
- Maquinário

2º. Campo Sonoro – Fonte de Interferência:

- Alto-falantes
- Atuadores Mecânicos (Vibradores)



Conceito Físico do CARA

A qualidade do sistema é então determinada pela capacidade de geração do sinal secundário no que concerne à fase, amplitude e velocidade.

Dentre as várias aplicações do sistema podemos destacar:

- Controle de ruído em dutos (exaustores, ar condicionado, ventilação);
- Comunicações: Fones de ouvido com utilização em ambientes ruidosos, aeronaves e veículos militares;
- Diminuição do ruído em pequenos ambientes fechados como cabine de aeronaves, automóveis e veículos militares.

A primeira patente de um sistema de controle ativo de ruído foi registrada nos EUA em 1936, quando o físico alemão Paul Lueg propôs o seguinte arranjo:

Sistema Apresentado por Lueg em 1936

A perturbação gerada pela fonte *A* é medida pelo microfone *M,* e enviada pelo sistema de controle *V* e ao alto-falante *L* de forma que a pressão do campo S_2 interfira destrutivamente com a perturbação incidente S_1. A falta de tecnologia da época impossibilitou sua aplicação prática.

Como as características da fonte acústica de ruído e do meio onde se propagam não são constantes, a freqüência, fase, amplitude e velocidade não são estacionárias: o sistema de controle ativo deve então ser adaptativo para poder controlar estas variações. O processamento digital de sinais (DSP) veio facilitar o processo de implementação deste sistema ativo em função do:

- Aumento da capacidade de processamento;
- Processamento em tempo real;
- Desenvolvimento de novos algoritmos;
- Baixo custo.

Esse trabalho pretende aplicar e combinar técnicas já consagradas como ponto de partida de estudo de um sistema hibrido multicanal para redução de ruído em pequenos ambientes confinados. Serão abordados conceitos teóricos e posteriormente sua aplicação prática para um sistema de simulação em tempo real.

CAPÍTULO I – Conceitos e Estruturas para o Controle Ativo de Ruído Acústico(CARA)

I.1 – Acústica

Para entendermos o cancelamento de ruídos é necessário relembrar alguns conceitos sobre o comportamento de ondas e vibrações. O som é uma onda de pressão viajando por um meio, no caso específico, o ar. Normalmente é originado pela vibração de um corpo, como as cordas vocais do ser humano ou um alto-falante. As ondas geradas pelo corpo em vibração são propagadas para o meio adjacente. O adjetivo “ruído” é normalmente aplicado ao som indesejado e que deve ser reduzido ao máximo, sem comprometimento do sinal original [1].

I.1.1 – Características do Som

As características mais importantes que definem o som são: freqüência, comprimento de onda, velocidade de propagação e amplitude. A relação da freqüência é inversamente proporcional ao comprimento de onda, a amplitude se relaciona com a pressão sonora e a velocidade de propagação, eq.1.1, depende do tipo, da temperatura e pressão do meio no qual a onda se propaga.

$$c = \sqrt{\frac{E}{\rho}} \tag{1.1}$$

Onde *E* é o Módulo de Young para sólidos, ou constante de módulo de elasticidade volumétrica e ρ é massa específica em repouso. Note que a propagação depende também da forma geométrica da fonte, fato que não será abordado neste estudo. A tabela I.1 mostra a velocidade de propagação em alguns meios.

Meio	Velocidade, c, ms^{-1}
Hidrogênio (20ºC)	1300
Ar (20ºC)	344
Alumínio	5200
Dióxido de Carbono (20ºC)	258
Ferro	5000
Concreto	3100
Madeira	3320
Cortiça	500
Poliestireno	1800

Tabela I.1 – Velocidade do som em meios específicos

I.1.2 – Som, Tom e Ruído

O tom é definido como a vibração de uma onda sônica puramente senoidal de freqüência f_0. O espectro de freqüência de um tom é representado por uma linha discreta em f_0 Hz. O som é uma superposição de ondas harmônicas, cujas freqüências são múltiplos inteiros da freqüência mais baixa, a freqüência fundamental. Então o espectro do som contém várias linhas discretas em:

$$f = (f_0 + n.f_0)\,Hz,\ n \in \mathrm{N} \wedge n \geq 0 \tag{1.2}$$

O ruído não possui tal periodicidade. Ele tem freqüências espalhadas em um intervalo do espectro de freqüências. O exemplo mais comum é o ruído branco, que possui sua energia distribuída por todas as bandas de freqüências.

I.1.3 – Nível de Pressão Sonora (SPL)

A mudança de pressão causada pela frente de onda sonora é chamada de pressão sonora. Essa pressão é imposta à pressão existente no meio pelo qual se propaga. Seu valor de referência é próximo de 2 10^{-5} Pa, menor nível sonoro que o ser humano pode perceber.

$$P_s = 10\log\frac{p^2}{p_{ref}^2}\,(dB)\ ou\ P_s = 20\log p + 94\,(dB) \tag{1.3}$$

I.1.4 – Potência Sonora (SWL)

O nível de potência sonora, de forma similar ao nível de pressão sonora, é calculado por referência a um nível padrão igual a 10^{-12}W.

$$S_s = 10\log\left(\frac{W}{W_0}\right)(dB)\ ou\ S_s = 10\log W + 120\,(dB) \tag{1.4}$$

I.1.5 – Sensibilidade e Audibilidade

A indicação da pressão sonora em decibel seria suficiente se a sensibilidade humana fosse independente da freqüência, mas isso não ocorre. Por exemplo, um som de 100 dB e freqüência de 100 Hz é percebido de forma menos intensa do que um de 100 dB e 1000 Hz. Para compensar as variações de sensibilidade com a freqüência, foram criadas curvas padrões

(A, B, C e D) conforme gráfico da figura I.3. Os valores em decibéis indicam as atenuações em relação à freqüência de referência de 1000 Hz. Os dados atenuados são indicados em dB seguido da letra da curva correspondente (dBA, dBB, etc). Exemplo: uma fonte sonora de 25 Hz e 50 dB de pressão corresponde a 50 - 44,7 = 5,3 dBA. Isso significa que ela é percebida com a mesma intensidade de uma fonte de 1000 Hz e 5,3 dB. As fontes sonoras usuais não emitem uma única freqüência. Na realidade, são espectros em uma determinada faixa de freqüências. Os instrumentos que medem pressão sonora (popularmente chamados de decibelímetros) fazem uma correção ponderada de acordo com as freqüências predominantes do espectro para dar um resultado na curva desejada. A unidade de nível de audibilidade é denominada *fon* e é dada pelos números que se encontram no interior do gráfico. O *fon* é a medida do nível audibilidade correspondente à pressão sonora em decibéis na freqüência de referência de 1000 Hz. Por exemplo: um nível de audibilidade de 100 fons equivale a 100 dB com 1000 Hz e aproximadamente 110 dB com 50 Hz. Observar que, quanto menores os níveis de audibilidade, maiores as diferenças, isto é, as curvas são menos "planas". Esta é a razão para a existência de várias curvas de compensação para medições. A curva A (dBA) é usada para níveis até 55 fons. A curva B (dBB) usada para a faixa de 55 a 85 fons. A curva C (dBC) usada para níveis acima de 85 fons. A curva D (dBD) é especial e usada para sons de alta freqüência e intensidade como turbinas de jatos. Na prática, é quase sempre usada a curva A (dBA) para medidas de pressão sonora, independentemente do nível.

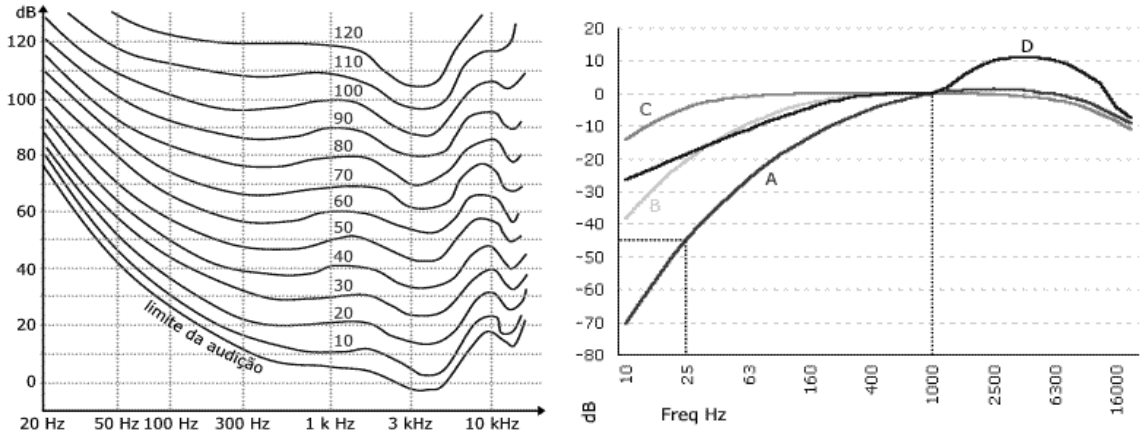


Fig. I.3 – Curvas de Audibilidade e Sensibilidade

I.1.6 – Fontes de Ruído

De modo geral os ruídos podem ser classificados pela forma de como o sinal varia no tempo. Alguns exemplos são mostrados na tabela abaixo.

Característica	Fonte	Nota
Constante e contínuo	Motores, bombas, caixas de redução	Pequenas variações, dentro de 5dB, por um longo período de tempo
Constante e intermitente	Compressores, maquinário com ciclos de trabalho	Pequenas variações ocorrem em períodos determinados
Flutuante	Produção em massa, escavações	Grandes flutuações
Flutuante não periódico	Trabalho manual, solda, montagem de componentes	Grandes flutuações irregulares
Impulsos Repetitivos	Maquinário pneumático	Impulsos repetitivos com menos de 1s
Impulsos Simples	Prensa, martelos	Duração < 1s

Tab. I.2 – Classificação de Ruídos

O ruído também pode ser classificado segundo seu conteúdo de freqüências:

- Ruído de banda estreita ("narrowband"): concentra sua energia em freqüências especificas. Pode ser considerado como um trem de impulsos filtrado. Uma das maiores dificuldades dos sistemas de cancelamento ativo de ruído consiste na previsão desse sinal. No entanto, este problema não se apresenta quando os sinais são perió PodsTT1o8 51Tdtuouls2 -1.727

- Ruído de banda larga (“broadband”): possui natureza randômica, sua energia está distribuída uniformemente em uma faixa de freqüências. Por exemplo, o ruído branco (“white noise”).

A maioria das aplicações de sistemas de cancelamento ativo de ruído são sistemas de banda estreita. As aplicações de sistemas de banda larga se resumem a sistemas de dutos de ar condicionado e a fones de ouvido ativos. Isso está relacionado com a maior dificuldade de implementação destes sistemas, a restrição de causalidade do controlador, e a maior dificuldade de obtenção de sinais de referência bem correlacionados com o sinal de ruído. Os problemas de causalidade estão muitas vezes ligados a fenômenos de reverberação, que surgem em sistemas multidimensionais, os quais podem ser minimizados por meio da utilização de sistemas multicanal[3].

I.2 - Estruturas de Sistemas CARA

O CARA se baseia no fenômeno da interferência de ondas. Quando duas ondas coerentes de mesma magnitude e defasadas de 180º se propagam na mesma direção, são neutralizadas por meio de uma interferência destrutiva[2].

O cancelamento de ruído pode ser dividido em três grupos:

- Cancelamento Global em Espaço Livre: é o total cancelamento do campo sonoro em três dimensões. A fonte sonora de cancelamento deve ser posicionada próxima a fonte de ruído, normalmente dentro de 0.1 vezes o comprimento de onda desta fonte. Consegue-se até 20 dB de redução global na intensidade do som para qualquer freqüência[1];
- Cancelamento em Dutos e Cavidades: é o cancelamento em espaços confinados como, por exemplo, cabines e dutos de ventilação. Em espaços confinados, as reflexões das paredes criam respostas modais que normalmente estão presentes quando o comprimento de onda se aproxima ou é menor que as dimensões da cavidade. O número de modos acústicos cresce muito com o aumento da freqüência e é dado por:

$$N = \frac{4\pi V}{3c^3} f^3 \qquad (1.5)$$

onde *V* é o volume da cavidade, *c* a velocidade do som e *f* a freqüência;

- Zonas de silêncio: é o cancelamento da intensidade do campo sonoro em uma região determinada e localizada. Uma zona de silêncio típica terá o diâmetro de um décimo do comprimento de onda.

Das considerações anteriores percebe-se que quanto maior a freqüência, maior será número de modos e menor o diâmetro das zonas de silêncio tornando o controle ativo impraticável nesta situação.

I.2.1 – Componentes Básicos de um Sistema CARA

O CARA está embasado nas áreas do Conhecimento de Controle e Processamento Digital de Sinais e, por assim ser, seus componentes básicos podem ser divididos em quatro classes:

- Planta: o sistema físico a ser controlado. Ex. um duto cheio de ar e a fonte de ruído;
- Sensores: são dispositivos que captam as perturbações aplicadas ao sistema. Microfones, sensores de vibração;
- Atuadores: dispositivos que, fisicamente, alteram o comportamento da planta. Geradores de vibração, alto-falantes;
- Controlador: modificador do comportamento do sistema, dentro de parâmetros especificados. Geralmente recebe as informações dos sensores, tratam o sinal digitalmente ou analogicamente segundo uma “lei”, e geram sinais para os atuadores.

I.2.2 – Modelos para CARA

O desenvolvimento deste trabalho supõe que o sistema a ser controlado é linear e invariante no tempo (LTI). Basicamente existem duas abordagens para o tratamento do problema: sistemas feedforward e feedback[1].

Esses sistemas podem ainda ser divididos em cinco categorias, segundo o tipo de ruído e número de canais:

1. Sistemas de banda larga feedforward;
2. Sistemas de banda estreita feedforward;
3. Sistemas Realimentados;
4. Sistemas multicanais;
5. Sistemas Híbridos.

O sistema de controle feedforward é utilizado quando uma entrada de ruído é amostrada antes que alcance o alto-falante de cancelamento. No sistema feedback, o controlador tenta cancelar o ruído com base em informações de um microfone localizado depois da alto-falante de cancelamento, sem o benefício de uma referência previamente adquirida. O controle feedforward é o mais pesquisado atualmente para sistemas CARA[3].

No controle “feedforward” (utilização de estados pré-calculados), a estratégia é antecipar o efeito de perturbações que vão atingir o processo através de seu sensoriamento e compensação antecipada aos efeitos. Elementos do controle feedforward captam a presença de perturbações e tomam ações corretivas através do ajuste de parâmetros do processo para compensar quaisquer efeitos introduzidos por essa perturbação. No caso ideal, a compensação

[1] A nomenclatura em inglês será mantida por ser amplamente adotada na Área de Engenharia e Controle.

é completamente efetiva. Entretanto, isso não é possível, pois as medidas (sensores), os atuadores, e os algoritmos de controle não são perfeitos. A escolha dos parâmetros, número de bits do processamento e sua velocidade são, então, fatores determinantes para o desempenho do sistema.

O que se pretende é cancelar o ruído em um ponto determinado no espaço. Esse cancelamento é conseguido acusticamente ao invés de eletricamente. O ruído se propaga acusticamente pelo canal para um ponto no espaço onde o cancelamento é desejado. O objetivo é remover os componentes de energia acústica gerados pela fonte de ruído.

I.2.3 – Estruturas Básicas

a. Ruído de banda larga feedforward;

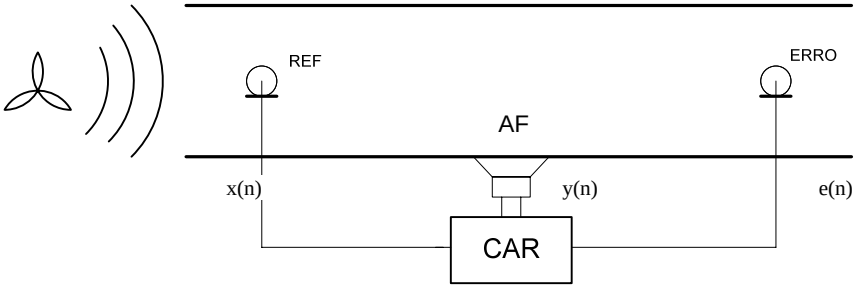


Fig. I.2 – Sistema físico com controle feedforward

O ruído de banda larga pode ser encontrado, por exemplo, em sistemas de ventilação por dutos. O sinal de referência *x(n)* é captado pelo microfone (REF) posicionado fisicamente perto da fonte de ruído e antes da fonte secundária de cancelamento (AF). O sistema CARA utiliza esse sinal de referência para gerar o sinal *y(n)* de amplitude igual, mas defasado em 180º. Esse antiruído é usado para excitar o atuador AF (alto-falante) e produzir um som que venha a atenuar o ruído acústico primário no duto.

O atraso gerado pelo tempo de propagação entre o sensor de referência REF e o atuador AF permite que o sistema re-introduza o ruído em ponto no qual o campo causará o cancelamento. Esse atraso deverá satisfazer os princípios da causalidade e coerência. Basicamente, o sistema deve ser capaz de medir o sinal de referência, tratá-lo e o mandar para o atuador AF no mesmo tempo em que o sinal de ruído chega ao atuador. Da mesma forma o sinal de ruído no AF deve ser muito semelhante ao medido por REF, significando que o canal acústico não modifica o ruído significativamente. Estes temas serão abordados em capítulos específicos no decorrer deste trabalho. O sensor (microfone) ERRO mede o sinal residual *e(n)*, que é utilizado para adaptar os coeficientes do filtro do controlador de modo a minimizar este erro.

Um ponto importante a ressaltar é que o uso de um sensor de erro não significa a implementação de um controle feedback, já que o sinal de erro não é comparado com sinal de referência[4].

b. Ruído de banda estreita feedforward;

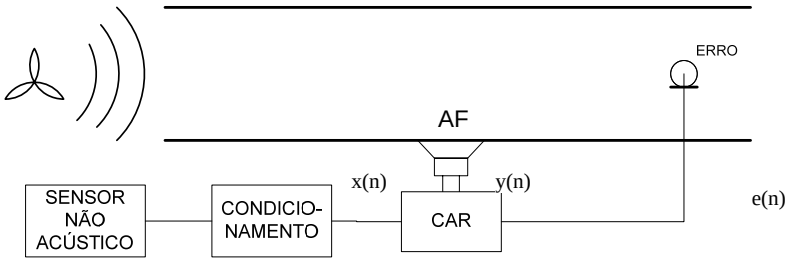


Fig. I.3 – Sistema físico com controle feedforward

Em sistemas onde o ruído primário é periódico, por exemplo, máquinas rotativas, o sensor de referência pode ser substituído por sensores não acústicos como tacômetros, acelerômetros ou sensores óticos. A vantagem de se utilizar esses sensores está no fato de que eles não sofrem influência de realimentações acústicas do sistema. O princípio de funcionamento é o mesmo do controle feedforward. Essa configuração elimina o problema da colocação de microfones em ambientes agressivos ou com alta temperatura e diminui o risco da ocorrência de problemas por causalidade já que o conteúdo de freqüências do sinal é praticamente constante.

c. Feedback.

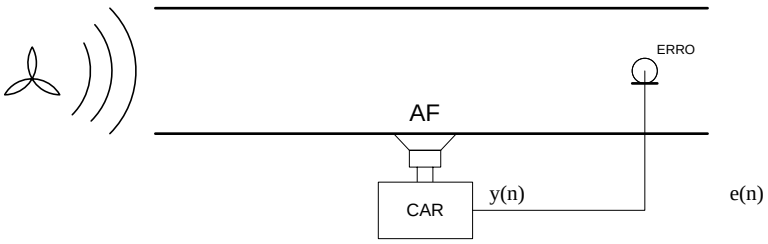


Fig. I.4 – Sistema físico com controle feedback

Nesta configuração apenas o microfone ERRO é utilizado para medir o ruído indesejado. Este sinal é retornado ao sistema, após ter sido tratado pelo sistema de controle, por meio do atuador AF (alto-falante). Essa configuração produz uma atenuação limitada e deve ser utilizada com ruídos periódicos e de banda limitada[4]. Esse sistema é mais eficaz quando o volume da cavidade é pequeno e a distância entre o sensor e atuador pode ser minimizada[5].

d. Multicanal

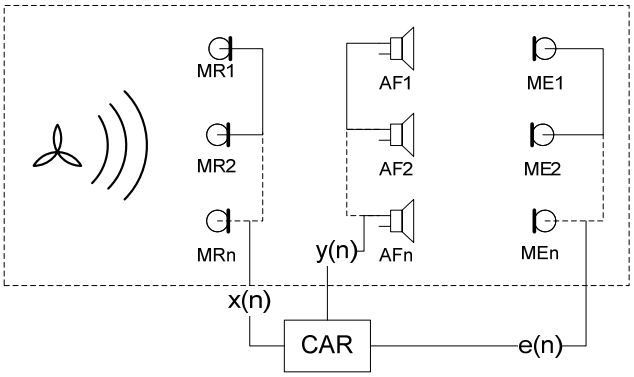


Fig. I.5 – Sistema físico multicanal

Em função do comportamento complexo de alguns ambientes, pode ser necessária a introdução de vários sensores e atuadores para controlar a resposta modal do sistema:

- Controle ativo em cabines de grandes aeronaves ou veículos;
- Controle ativo em dutos e ambientes de grandes dimensões;
- Controle de vibração de corpos rígidos ou estruturas com vários graus de liberdade.

Quando a geometria do campo sonoro é complexa, é insuficiente o emprego de apenas uma fonte para cancelar o ruído e um único microfone de referência e erro. O controle de campos acústicos complexos implica no desenvolvimento de um controlador multicanal, com algoritmos de controle com múltiplas entradas e múltiplas saídas, tornando o sistema muito mais sensível à capacidade computacional e do processamento de sinais[6].

e. Hibrido

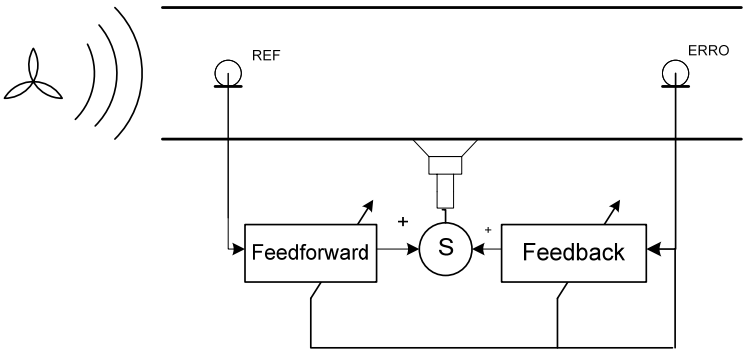


Fig. I.6 – Sistema físico hibrido

Quando o sinal de cancelamento é gerado pela combinação dos sistemas feedforward e feedback e, consequentemente, pelos sinais dos microfones REF e ERRO temos o sistema hibrido para cancelamento de ruído. É importante notar que o sinal de erro é agora utilizado tanto para gerar o sinal de referência para o filtro adaptativo de feedback quanto para ajuste

dos coeficientes dos filtros feedforward e feedback [7]. Por meio desta configuração consegue-se atenuar o ruído primário, que é coerente com o sinal de referência – processo feedforward – enquanto as componentes de banda estreita do sinal primário são canceladas pelo processo de feedback. Esta configuração, no entanto, demanda uma maior capacidade computacional e adiciona uma maior complexidade aos algoritmos adotados.

A tabela I.3 mostra uma comparação entre os diversos modelos para CARA apresentados:

	FEEDFORWARD	FEEDBACK	HIBRIDO FIR	HIBRIDO IIR
Ordem dos Filtros	Moderada	Alta	Baixa	Baixa
Tipo de Ruído	Bandas Estreita e Larga	Banda Estreita	Bandas Estreita e Larga	Bandas Estreita e Larga
Coerência do Campo Sonoro	Coerentes	Coerentes e Incoerentes	Coerentes e Incoerentes	Coerentes e Incoerentes

Tab. I.3 – Comportamento dos Sistemas CARA, adaptado de Kuo e Morgan,1996.

O diagrama da figura I.7 mostra o sistema feedforward detalhado. Esse trabalho abordará o uso de filtros adaptativos do tipo FIR (“Finite Inpulse Response”), IIR (“Infinte Inpulse Response”), algoritmos do tipo LMS (“Least Mean Square”) e algumas de suas variações. O tratamento dos sinais e a característica dos filtros adaptativos serão tratados no Capítulo II.

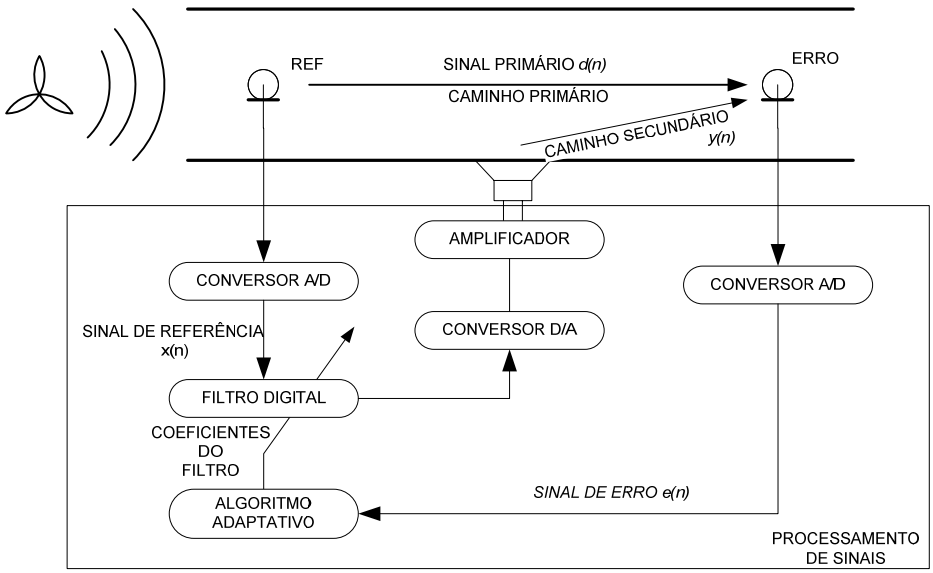


Fig. I.7 – Sistema físico com controle feedforward detalhado

Esquematicamente, este sistema pode ser divido também em caminhos acústico e elétrico como mostrado na figura I.8.

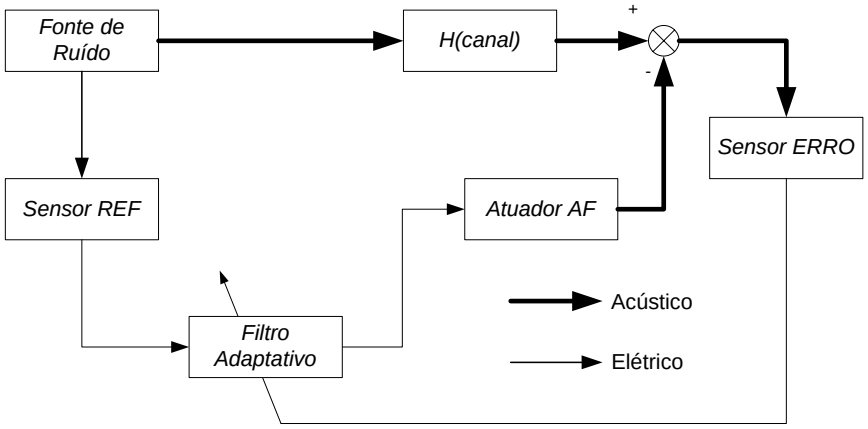


Fig. I.8 – Sistema acústico e elétrico

CAPÍTULO II – Filtros Digitais Adaptativos

II.1 – Filtros Adaptativos para Controle

Os filtros digitais convencionais utilizados para seleção de faixas de freqüências com coeficientes fixos são projetados para produzir uma reposta determinada, modificando o espectro do sinal de entrada e obtendo uma saída especificada. Suas principais características são as seguintes:

- Os coeficientes do filtro são fixos;
- Os filtros são lineares e invariantes no tempo;
- Não é necessário se conhecer amostras do sinal a ser processado;
- Os filtros são seletivos para determinadas freqüências. Esses filtros funcionam bem quando utilizados com sinais cujos componentes de freqüência não se sobreponham. É mais fácil separar o ruído do sinal quando espectros não se sobrepõem.

No entanto, em sistemas onde não se possua informações suficientes sobre o sinal a ser tratado ou do critério de filtragem venha a ser empregado durante o processo, os filtros fixos não são capazes de solucionar o problema. A solução é, então, utilizar filtros adaptativos ou filtros “inteligentes” que podem modificar sua resposta para melhorar o desempenho do sistema sem intervenção externa.

As aplicações típicas dos filtros e exemplos de utilização adaptativos são os seguintes:

- Identificação de sistemas, controle adaptativo, cancelamento de eco, modelagem de sistemas;
- Cancelamento de interferências de um ou múltiplos sensores: controle de ruído acústico, adaptação de feixes acústicos (“beamforming”);
- Inversão de Sistemas: equalização adaptativa;
- Predição de sinais: cancelamento de interferência de radiofreqüência.

Para o desenvolvimento do trabalho serão abordados os problemas de identificação de sistemas e cancelamento de interferências.

A maioria dos filtros adaptativos envolve o conhecimento *a priori* do sinal de entrada e conhecimento da aplicação do filtro[8] Este fato indica que o filtro adaptativo possuirá melhor desempenho se projetado para uma aplicação específica, ou de outro modo, para um tipo de sinal de entrada específico (voz, contínuo, tonal...), uma interferência específica (senoidal, ruído branco, banda larga ou estreita...) e meios conhecidos (invariantes ou variantes no tempo)[9]. Quaisquer erros nessas informações podem levar o filtro adaptativo à instabilidade ou degradação de desempenho[11]. Se as características dos sinais envolvidos são constantes, a meta do filtro adaptativo é chegar a um conjunto de parâmetros que leve ao

melhor desempenho do sistema e então parar o processamento. Caso alguma característica do sinal varie no tempo, o filtro deverá ser capaz de, continuamente, reajustar os parâmetros para acompanhar tais mudanças.

Nos sistemas CARA, os filtros digitais ou filtros de controle, são responsáveis pela geração do sinal de controle de saída em função do sinal de referência de entrada. Os filtros tomam amostras discretas do sinal de referência atual e utilizam-se de amostras passadas que, multiplicadas por um conjunto de coeficientes (ou pesos) e somadas, geram uma amostra de saída. O valor dos coeficientes determina como o sinal de referência é modificado pelo controle para se chegar a um sinal de saída desejado.

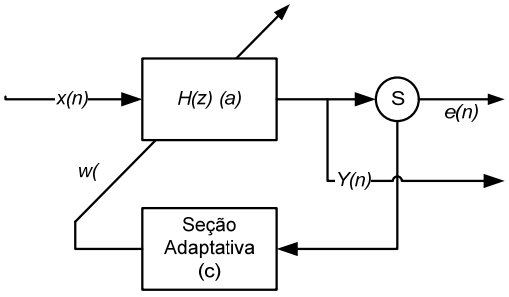
As estruturas mais comuns de filtros adaptativos podem ser classificadas como[10]:

- Filtros Transversais;
- Filtros Combinadores Lineares;
- Filtros Recursivos;
- Filtros Treliça;
- Filtros Não Lineares.

No presente trabalho, serão utilizados filtros que se encontram dentre as três primeiras categorias. A estrutura mais utilizada no caso CARA, são os filtros transversais por serem mais estáveis e de menor complexidade computacional[5][6]. A estrutura de combinação linear é uma versão generalizada da estrutura transversal, sendo mais utilizada em sistemas com múltiplas entradas de sinal.

II.2 - Estrutura Básica de um Filtro Adaptativo

Um filtro adaptativo se constitui basicamente de três seções, como mostrado na figura II.1:



FIR é mais comum. Os filtros adaptativos são também denominados como combinadores lineares adaptativos (“Adaptive Linear Combiner”);

b) Seção de Erro: essa seção computa a estimação do erro *e[n]* comparando a saída do filtro *y[n]* e o sinal desejado *d[n].* No entanto, em muitas aplicações práticas, *d[n]* não está instantaneamente disponível e deve ser obtido a partir do processamento de um outro sinal;

c) Seção adaptativa: usa a estimação do erro obtida em (b) e computa o peso (“weight”) para o filtro (a).

Vários algoritmos diferentes podem ser utilizados na seção adaptativa. Eles variam muito em sua complexidade computacional e desempenho. Muito do trabalho no projeto do filtro adaptativo está relacionado com a escolha do algoritmo que ofereça um equilíbrio aceitável entre desempenho, custo de implementação e operação. De modo geral os algoritmos utilizados para os filtros FIR e IIR são diferentes, onde os mais simples são utilizados com os filtros FIR. Os algoritmos mais utilizados são: Steepest Descent (SD), Least-mean-square (LMS) e Recursive Least-mean-square (RLS) e algumas de suas variações.

II.2.1 – Estruturas para Cancelamento e Identificação

Os filtros adaptativos podem ser configurados conforme sua aplicação. Cancelamento de ruído, cancelamento de eco, identificação de sistema, tratamento de imagem, dentre outras, são variantes do mesmo princípio de aplicação de filtros digitais.

II.2.1.1 - Cancelamento de Ruído

A chave para essa aplicação é o emprego de um ou múltiplos sensores para remover a interferência indesejada do sinal de entrada (sinal primário). Normalmente, o sinal de entrada já possui o ruído incorporado ao sinal desejado. Os sensores são utilizados para captar do sistema sinais de referência, de modo a se conseguir o cancelamento das interferências. O nível de correlação entre o sinal primário e de referência medido é utilizado para estimar a interferência no sinal primário, a qual, no decorrer do processo adaptativo, é eliminada [9].

A estrutura mostrada na figura II.2 é implementada para a aplicação em cancelamento de ruído. O sinal desejado *d* é corrompido, sem correlação, por um sinal de ruído *n.* A entrada do filtro adaptativo é um sinal de ruído *n’* correlacionado com o sinal *n*. A saída do filtro adaptativo *y* é adaptada ao ruído *n.* Quando esta situação ocorrer, o sinal de erro *e* se

aproxima do sinal desejado *d.* A saída que se deseja é então o sinal de erro e não o sinal de saída do filtro *y.* Se *d* não esta correlacionado com *n,* a estratégia é minimizar o erro $E(e^2)$, onde *E()* representa o valor estimado deste erro.

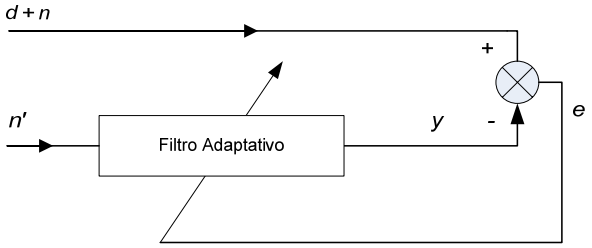


Fig. II.2 – Cancelamento

II.2.1.2 - Identificação de Sistema

A identificação de sistemas ou modelagem de sistemas se caracteriza pelo fato de que o sinal de entrada não possui ruído e a resposta desejada é corrompida por um sinal de ruído não correlacionado com o sinal de entrada. Em aplicações de controle, o filtro adaptativo é utilizado para se obter a estimação dos parâmetros ou estados do sistema e, então, projetar o controlador. No processamento de sinais é empregado para estimar a resposta do sistema, de acordo com o critério de desempenho especificado.

A estrutura apresentada na figura II.3 é utilizada na identificação de sistemas. A mesma entrada *x* é aplicada ao filtro e ao sistema. O sinal de erro e é a diferença entre as respostas do sistema *d* e do filtro adaptativo *y*. Esse erro retorna na forma de realimentação ao filtro e é utilizado para atualizar os coeficientes, até que a condição *y=d* seja alcançada. Quando esta situação é alcançada o erro tende a zero. Se o sistema desconhecido for linear e invariante no tempo, então depois da adaptação ter sido completada, as características do filtro não mudam mais.

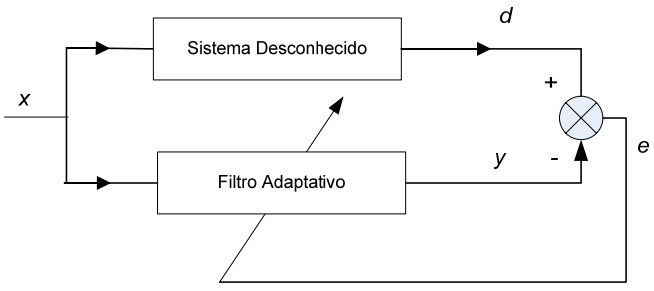


Fig. II.3 – Identificação

II.2.2 - A Função de Transferência Discreta

A função de transferência de um modelo de filtro digital é conhecida como função de transferência discreta, e é normalmente calculada por uma série de multiplicações e adições realizadas com amostras dos sinais envolvidos. Essa função de transferência reflete “o trabalho” que é realizado pelo filtro. Por exemplo, se o sistema for um amplificador que

multiplica o sinal por 10, então a função de transferência do amplificador, teoricamente, poderia ser representa por “x10” para todas as freqüências. No entanto na maioria dos sistemas físicos que envolvam o controle adaptativo, essa representação se dá por meio de equações diferenciais, matematicamente complexas. Uma forma de contornar esse processamento complexo é utilizar o processamento digital que se resume a multiplicações e adições do sinal amostrado. Um exemplo é o filtro transversal mostrado na figura II.4.

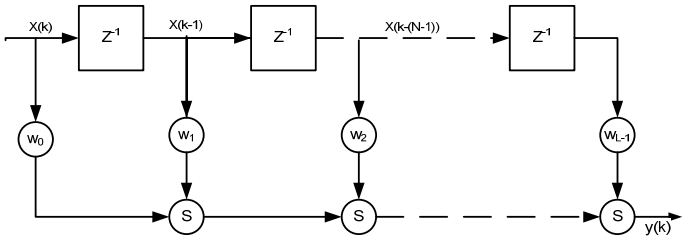


Fig. II.4 – Esquema básico de um filtro transversal

Na sua forma mais geral, a saída de um filtro *y[k]* pode ser representada por valores ponderados da soma dos valores presentes de entrada e saída e de valores passados de saída, conforme apresentado na eq. 21:

$$y(k) = b_0 x(k) + b_1 x(k-1) + b_2 x(k-2) + \ldots + b_n x(k-n) + a_1 y(k-1) + a_2 y(k-2) + \ldots + b_m y(k-m) \tag{2.1}$$

Nesta expressão, os valores de *a* e *b* representam os coeficientes do filtro ou seus *pesos*. São construídas então derivações (‘taps”) formadas pela multiplicação da amostra do sinal pelo peso do filtro e depois realizado um somatório destes valores. Essa operação é conhecida como uma operação de multiplicar/acumular (MAC – multiply/accumulate). Esta estrutura é representada na figura II.5.

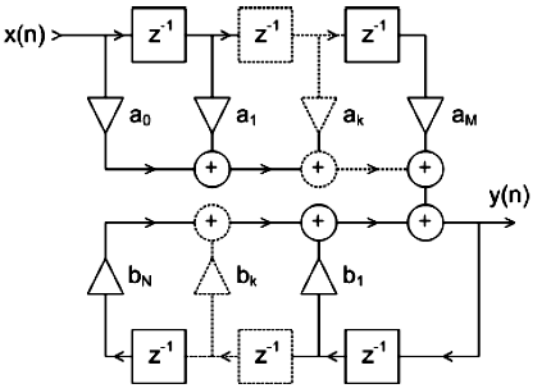


Fig. II.5 – Estrutura de Filtro IIR

O filtro mostrado na figura II.5 está representado em sua forma “direta”. Embora existam várias representações matemáticas para representar os filtros digitais, a forma direta é a mais usada e a mais simples de apresentar os cálculos que estão por trás do processamento de filtros digitais.

A função de transferência associada a este filtro pode então ser descrita pela eq.2.2:

$$H(z)=\frac{y(z)}{x(z)}=\frac{a_0+a_1Z^{-1}+a_2Z^{-2}+\cdots+a_nZ^{-N}}{1-b_1Z^{-1}-b_2Z^{-2}-\cdots-b_MZ^{-M}} \tag{2.2}$$

onde Z^{-x} relaciona o coeficiente do filtro com a amostra tomada *x* amostras atrás. O filtro da figura II.5 pode ser resumido por uma linha de retardos (“delays”), um conjunto de pesos e um acumulador. Essa estrutura é conhecida como filtro transversal ou linha de retardo com derivação (“tapped delay line”). O número de estágios representa o número de taps do filtro. Reescrevendo a eq. 2.2 temos que:

$$H(z)=\frac{Y(z)}{X(z)}=\frac{\sum_{k=0}^{M}a_k.z^{-k}}{1-\sum_{k=1}^{N}b_k.z^{-k}} \tag{2.3}$$

A eq. 2.3 pode ser representada de uma forma diferente:

$$\frac{Y(z)}{W(z)}\cdot\frac{W(z)}{X(z)}=\frac{\sum_{k=0}^{M}a_k\cdot z^{-k}}{1}\cdot\frac{1}{1-\sum_{k=1}^{N}b_k\cdot z^{-k}} \tag{2.4}$$

Desta representação observam-se duas relações:

$$Y(z)=W(z)\cdot\sum_{k=0}^{M}a_k\cdot z^{-k}$$

$$W(z)=X(z)+W(z)\cdot\sum_{k=1}^{N}b_k\cdot z^{-k} \tag{2.5}$$

As relações anteriores no domínio *z* podem ser escritas em sua forma equivalente no domínio do tempo, reconhecendo-se que z^{-k} no domínio Z representa um atraso de *k* amostras no domínio do tempo.

$$y(n) = \sum_{k=0}^{M} a_k \cdot w(n-k) \tag{2.6}$$

$$w(n) = x(n) + \sum_{k=1}^{M} b_k \cdot w(n-k) \tag{2.7}$$

A estrutura da figura II.5 pode, então, ser representada como:

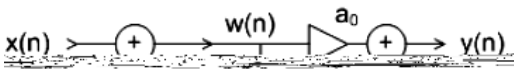


Fig. II.6 – Representação do filtro no domínio do tempo

II.3 – Filtros FIR e IIR

Como citado anteriormente, os filtros mais utilizados para controle adaptativo são os IIR e o FIR. A diferença básica é que o filtro IIR possui realimentação. O exemplo das figuras II.4 e II.5 representam, respectivamente, os filtros FIR e IIR. O nome destes filtros está associado com a seção de realimentação do filtro: se ela existe, então a resposta ao impulso vai gerar uma saída infinita. Se não existir, o impulso se propaga pela linha de retardo e desaparece. A saída então, só existe durante um intervalo de tempo determinado.

O filtro FIR também é conhecido como filtro não recursivo ou “all-zero filter”, pois não possuem pólos ou como filtros de média variável (moving average – MA). O filtro IIR é um filtro recursivo ou “pole-zero filter” e é um filtro de média variável auto-recursiva (ARMA).

Quanto ao número de taps a ser utilizado no filtro, não existe uma regra definitiva para o critério de seleção. Existem apenas algumas referências. Quando o sinal de referência for puramente senoidal (tonal) e o sinal de controle não o corrompa, na teoria, é possível se obter um ganho arbitrário e mudança de fase com apenas dois taps em um filtro FIR. Na prática, o uso de apenas dois taps implica em valores de coeficientes muito grandes. Isso é mais sentido quando as taxas de amostragem são significativamente grandes (mais de 20 vezes) em

relação ao sinal senoidal. Valores entre 4-20 taps podem ser tentados. Se múltiplos sinais senoidais estão presentes, então 4-20 taps por freqüência são valores bons para se começar. Quando o resultado é insatisfatório então normalmente o número de taps deve ser aumentado[6].

Se o espectro de freqüências do sinal for amplo e a ocorrência de realimentação acústica da fonte de controle ao sensor de referência estiver presente, o caso é ainda mais complexo. Deve-se então pensar no emprego de um filtro IIR, quando valores entre 4-10 taps por pico de ressonância podem ser testados tanto para a linha de entrada quanto para linha de realimentação[4][6][12].

O custo computacional do filtro IIR é menor, já que consegue modelar sistemas complexos com um número de coeficientes menor do que o filtro FIR. No entanto, esta vantagem deve ser avaliada se levarmos em consideração a instabilidade intrínseca do filtro IIR, menor velocidade de convergência e possibilidade de convergência para um mínimo local na superfície de erro ao invés de um mínimo global[5]. Essa instabilidade é conseqüência da presença da realimentação: se o ganho de realimentação aumenta muito o sistema pode ser tornar instável, em oposição ao filtro FIR, onde o ganho pode aumentar sem causar instabilidade. O filtro FIR possui função de transferência do tipo *all-zero*. Esse modelo não permite que pólos se localizem fora do círculo unitário, daí sua estabilidade[13].

II.3.1 – Implementação

No item II.3 foram mostradas as vantagens e desvantagens dos filtros FIR e IIR. Nesta seção serão apresentadas algumas questões em relação à implementação destes filtros. A precisão da representação dos sinais e coeficientes e se a aplicação será em tempo real, ou não, devem ser estudadas. Esta é uma abordagem mais voltada ao processamento de sinais, mas que será importante para entendimento do Capítulo IV – Implementação em Sistema de Desenvolvimento TMS.

Como a implementação de filtros se dá por do uso da aritmética de precisão finita, seu desempenho pode ser influenciado por vários fatores: quantização do sinal de entrada, quantização do valor dos coeficientes e arredondamento dos valores resultantes das operações aritméticas para implementar o filtro.

II.3.1.1 – Representação dos Sinais

Os processadores DSP[2] (“Digital Signal Processing”) trabalham com representação numérica em ponto fixo ou ponto flutuante. As necessidades do sistema, velocidade de

[2] A nomenclatura DSP será adotada por ser corrente na área de Processamento de Sinais

processamento e custo irão determinar qual processador será utilizado. Parte integrante dos sistemas DSP são os conversores A/D e D/A que trabalham com representação em ponto fixo dos sinais. Essa representação pode ser realizada de várias formas:

$$0011000_2=152_{10} \text{ – Binário} \tag{2.8}$$

$$10011000_2=-104_{10} \text{ – C2} \tag{2.9}$$

$$1001.1000_2=9.5_{10} \text{ – Ponto Binário} \tag{2.10}$$

Existem vantagens e desvantagens para cada representação. A representação de números com sinal é necessária na maioria das aplicações em DSP e o modelo C2 facilita essa tarefa para adição e subtração, mas para multiplicação cuidados devem ser tomados. A representação em ponto binário, utilizada na maioria dos processadores DSP comerciais, funciona bem para operações de adição, subtração e multiplicação pois garante que não ocorrerá overflow.

Uma questão que deve ser abordada é a relação sinal-ruído (SNR) de um sistema digital. Em um sistema digital o SNR é diretamente proporcional aos bits de quantização utilizados.

$$SNR_{dB} = 6.02 \cdot B + 20 \cdot log(\sigma_n / A_{\max}) + 10.8 \tag{2.11}$$

Onde B representa o número de bits usado para representar o sinal digitalizado enquanto σ_n e $A_{\max}$,o desvio padrão e máximo valor do sinal. Normalmente a amplitude do sinal de entrada é ajustada para que a razão $\sigma_n / A_{\max}$ seja aproximadamente ¼. Se essa condição for alcançada, é assumido que o sinal possui distribuição Gaussiana e que os limites do quantizador serão excedidos apenas 0.064% do tempo[7]. Valores maiores para esta relação poderão gerar overflow e valores menores implicam no sacrifício da faixa dinâmica do sistema. Substituindo o valor ¼ em 2.11:

$$SNR_{dB} = 6.02 \cdot B - 1.2 \tag{2.12}$$

Da eq.2.12 podemos deduzir que a cada bit acrescentado à quantização, melhoramos o SNR em 6dB. Em um sistema onde se deseja um SNR de 90dB, então o conversor D/A deverá possuir, no mínimo, 16 bits.

II.3.1.2 – Representação dos Coeficientes

A representação dos coeficientes em sistema DSP deve ser decisiva na escolha da arquitetura a ser utilizada. Essa representação deve possuir acurácia suficiente para trabalhar com os valores dos coeficientes, que determinam os locais dos pólos e zeros do sistema, de modo a garantir as características de projeto do filtro no momento da implementação. Se a representação dos valores dos coeficientes for comprometida, a resposta dos filtros aparecerá distorcida e a instabilidade dos filtros IIR ainda mais enfatizada.

A resposta de freqüência do filtro também está relacionada com a representação dos coeficientes. A eq. 2.13 indica como determinar o truncamento dos valores em função do número de bits utilizados.

$$w(n) = Round\left(\frac{w(n)\cdot(2^{B-1}-1)}{w(n/2)}\right)\cdot\frac{w(n/2)}{(2^{B-1}-1)} \qquad (2.13)$$

Desta equação determina-se que quanto menor o número de bits, maior será o erro no valor dos coeficientes. Modificando o valor dos coeficientes, modifica-se o valor dos pólos e zeros do sistema, modificando assim sua resposta de freqüência[4][14].

II.3.1.3 – Acurácia e Estabilidade

Para os filtros FIR e IIR a representação interna dos coeficientes e do processamento dos sinais reflete, diretamente, na acurácia e estabilidade do sistema. Nestes filtros os valores de saída são guardados em um registrador que acumula o valor sinal de saída. Esse acumulador deve possuir o maior número de bits possível porque ele determina a acurácia do processamento. Overflow e underflow são problemas potenciais para esse acumulador já que é difícil prever a natureza do sinal de entrada. A maioria dos processadores DSP possui um controle de entrada onde grandes variações temporárias do sinal são acomodadas para evitar a propagação de erro para o restante do processamento. Esse processo é conseguido por meio de um bit de escalonamento que é utilizado, temporariamente, como um bit extra para o acumulador para evitar overflow. No entanto, caso o overflow seja inevitável, é melhor utilizar um processador que simplesmente sature no valor máximo do que permitir a ocorrência de overflow, que pode ser interpretado como uma mudança de um valor positivo para um valor negativo.

Em sistemas recursivos que utilizem representação de precisão finita, pode ocorrer um problema chamado oscilação dos ciclos limite. Existem dois tipos de ciclos limite: overflow e quantização. Os ciclos limite de overflow (também chamado de ciclos limite para grandes sinais

– “large signals limit cycles”) são devidos ao problema não controlado de overflow do acumulador durante o processamento. A maioria dos problemas do ciclo limite de overflow podem ser contornados por meio do bit de escalonamento. Os ciclos limite de quantização (também chamado de ciclos limite para pequenos sinais – “small signals limit cycles”) são resultados do tratamento da quantização dentro do sistema e são notados quando a saída deve ser constante ou igual a zero. Esse problema ocorre quando a variação do sinal de entrada é menor que o nível de quantização. Uma forma de minimizar esse efeito, que pode não ser prática, é aumentar o número de bits de quantização. Outra forma sugere que valores resultantes de operações aritméticas sejam truncados e não arredondados enquanto são acumulados.

O algoritmo é dito como numericamente estável se o erro de arredondamento se propaga de forma estável no sistema. O incremento da precisão numérica não melhora a estabilidade do sistema e pode mesmo só aumentar o tempo de convergência do algoritmo[15]. A acurácia indica, no estado estacionário, a diferença entre os valores obtidos e os esperados em função dos erros de arredondamento. A acurácia numérica pode ser melhorada pelo incremento do tamanho da palavra. A falta de estabilidade numérica pode levar a erros de overflow, divergência ou estouro do algoritmo[16].

II.3.2 – Algoritmos para Sistemas CARA

Esta seção pretende apresentar os algoritmos clássicos empregados no controle ativo de ruído. O algoritmo adaptativo utilizado pelo controlador é responsável por determinar os coeficientes ótimos do filtro para minimizar o sinal de erro de saída. Um outro algoritmo adaptativo é responsável por obter o modelo da resposta ao impulso do caminho de cancelamento. Mais adiante neste capítulo, os métodos para se modelar os caminhos acústicos serão detalhados. O sistema de duto [17] exemplificará, na prática, tais conceitos.

Para a obtenção de uma solução com coeficientes ótimos, deve-se lembrar que tanto as entradas quanto as saídas do sistema são processos estocásticos. Isso implica em não se conseguir uma solução única que produza erro zero todas às vezes. Então a solução é procurar um método que consiga alcançar o menor erro médio quadrático.

II.3.2.1 – Modelo Básico para Estudo dos Algoritmos Adaptativos

A forma geral do filtro adaptativo para estudo dos algoritmos é apresentada na figura II.7 onde *d(n)* é a resposta desejada ou o sinal de perturbação proveniente da fonte primária e *y(n)* a saída do filtro digital. A referência é o sinal *x(n).* O sinal de erro *e(n)* é a diferença entre a

saída do controlador e a perturbação. O algoritmo adaptativo é utilizado para minimizar o erro médio quadrático do sinal *e(n).*

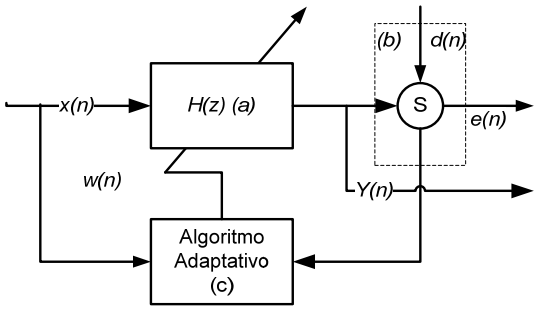


Fig. II.7 – Modelo básico para aplicação dos algoritmos CARA

O modelo utilizado é uma derivação do modelo apresentado na figura II.1, onde os coeficientes são agora atualizados amostra após amostra. A eq. 2.6 pode então ser reescrita como:

$$y(n) = \sum_{k=0}^{M-1} a_k(n) \cdot x(n-k) \tag{2.14}$$

Para facilitar o desenvolvimento, a notação vetorial será utilizada[4]:

$$x(n) = [x(n)\ x(n-1) \cdots x(n-M+1)]^T \tag{2.15}$$

$$w(n) = [w_0(n)\ w_1(n) \cdots w_n(n)]^T \tag{2.16}$$

A saída pode então ser representa pelo produto dos vetores (2.15) e (2.16):

$$y(n) = w(n)^T \cdot x(n) = x(n)^T \cdot w(n) \tag{2.17}$$

O sinal de erro pode ser obtido por:

$$e(n) = d(n) - w(n)^T \cdot x(n) \tag{2.18}$$

II.3.2.2 – Minimização do Erro Médio Quadrático (MSE – Mean Square Error)

O MSE é definido em [2] como a expectativa do valor quadrático do erro:

$$\xi(n) = E(e^2(n)) \tag{2.19}$$

O filtro será adaptado em função da evolução de $\xi(n)$. O sinal de entrada pode ser representado por sua matriz de autocorrelação [18]:

$$R = E[x(n)x^T(n)] \tag{2.20}$$

A relação entre a entrada e o sinal desejado *d(n)* é dada pelo vetor de correlação cruzada:

$$p = E[x(n)d(n)] \tag{2.21}$$

Para que o sinal de erro seja minimizado, os parâmetros a serem ajustados são os coeficientes do filtro determinando que o erro é função desses coeficientes. Os coeficientes ótimos, w_o, são dados por [9]:

$$w_o = R^{-1}p \tag{2.22}$$

Finalmente chega-se a forma final da expressão de $\xi(n)$:

$$\xi(n) = E[d^2(n)] - 2 \cdot p^T \cdot w(n) + w(n)^T \cdot R \cdot w(n) \tag{2.23}$$

A função $\xi(n)$ é uma função quadrática dos coeficientes do filtro. Para exemplificar, tomemos um sistema com apenas dois pesos, M=2. A superfície de erro pode ser representada como função dos pesos $w_o(n)$ e $w_1(n)$, como apresentado há figura II.8:

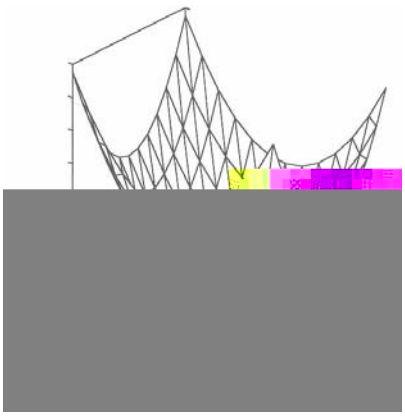


Fig. II.8 – Superfície de Erro para M=2

O conjunto de coeficientes $w^o(n) = (w_0^o(n) \;\; w_1^o(n))$ são os coeficientes ótimos obtidos na amostra *n* que produz o menor MSE, $\xi_{\min}(n)$. Dadas as características das funções quadráticas, esse vetor será único para qualquer valor de M.

A implementação desta solução em sistemas DSP é complicada em função do contínuo processamento de cálculos matriciais e suas inversões (R e p). Foram então desenvolvidos métodos recursivos que permitem o cálculo dos coeficientes ótimos, com menor esforço computacional, mesmo em sistemas com número de pesos muito elevado. Esses algoritmos serão apresentados a seguir.

II.3.3 – Algoritmo Adaptativo

A função do algoritmo adaptativo, como já visto anteriormente, é ajustar os coeficientes do filtro adaptativo de forma a diminuir a correlação entre o sinal de referência e o sinal de erro. Para tanto, um ou mais sinais são aplicados ao sistema de controle e o filtro tentará, então, por meio da modificação dos coeficientes, minimizar o sinal de erro.

Na verdade, o que se deseja é diminuir a potência do sinal do erro quadrático do sistema. Esses algoritmos operam adicionando ao valor atual dos coeficientes um pequeno percentual do gradiente negativo da superfície de erro, para calcular um conjunto de coeficientes mais adequados. A maioria dos algoritmos adaptativos possuem a seguinte forma:

$$\begin{bmatrix} Novo\,vetor \\ de \\ coeficientes \end{bmatrix} = \begin{bmatrix} Antigo\,vetor \\ de \\ coeficientes \end{bmatrix} + \begin{bmatrix} Vetor\,de \\ ganhos\,do \\ filtro\,adaptativo \end{bmatrix} . \begin{pmatrix} sinal \\ de \\ erro \end{pmatrix}$$

A escolha do algoritmo a ser utilizado deve levar em consideração as propriedades mais importantes de cada um dos algoritmos: a estabilidade, a curva de aprendizagem, a taxa de convergência, o erro residual, a capacidade de acompanhamento e capacidade de rejeição de ruído. Estas propriedades determinam o desempenho de um filtro ótimo, que minimiza o valor esperado do erro quadrático médio num determinado instante, e que em geral é variante no tempo. Estas variações estão relacionadas com variações no sistema físico que produziu os sinais os quais são processados pelo filtro adaptativo.

As propriedades são as seguintes:

1. A estabilidade do algoritmo corresponde ao conjunto de valores dos seus parâmetros e das variáveis externas para garantir a convergência dos coeficientes para um dado filtro adaptativo;
2. A curva de aprendizagem consiste na evolução, ao longo do tempo, do erro quadrático médio, desde quando o algoritmo é iniciado até o tempo que este atinge um dado nível de desempenho;
3. A taxa de convergência corresponde à derivada da curva de aprendizagem;
4. O tempo de convergência é o intervalo de tempo em que a curva de aprendizagem atinge um determinado valor de erro;

5. O erro residual corresponde ao menor valor do erro que a curva de aprendizagem atinge;
6. Variação de erro: é a diferença entre o erro residual e o valor mínimo do erro obtido por um filtro ótimo invariante no tempo;
7. A capacidade de rejeição de ruído reflete a capacidade do algoritmo de lidar com interferências, com o mínimo de alteração do seu comportamento. A interferência é qualquer ruído que impossibilite a saída do filtro acompanhar exatamente o sinal desejado. Esta propriedade está relacionada com o erro residual;
8. A capacidade de seguimento reflete a capacidade do algoritmo de acompanhar, mais ou menos rapidamente, alterações do filtro ótimo em ambientes livres de ruído. Esta propriedade está ligada à taxa de convergência e ao tempo de convergência, que são propriedades distintas. É fácil imaginar um algoritmo, que convirja rapidamente, e que não seja capaz de se adaptar às alterações referidas. A taxa de convergência corresponde a uma propriedade transitória, que apenas diz respeito ao período inicial do algoritmo. A capacidade de acompanhamento corresponde a uma propriedade estacionária, na medida em que diz respeito ao funcionamento estacionário do algoritmo, depois de se terminarem os regimes transitórios iniciais. De fato, um aumento da taxa de convergência pode ser obtido, em parte, através de um aumento inicial do passo de adaptação associado ao algoritmo, com uma redução posterior. A capacidade de acompanhamento combina a capacidade de acompanhamento na ausência de ruído com a capacidade de rejeição de ruído. Não é suficiente que o algoritmo seja capaz de seguir rapidamente alterações do sistema físico se disso resultar uma amplificação ou elevada sensibilidade ao ruído. Assim, em geral, é necessário resolver o compromisso entre uma rápida adaptação a alterações no sistema físico e uma boa rejeição de ruído, o que resulta numa boa capacidade de acompanhamento. As propriedades mais relevantes de um dado algoritmo dependem da aplicação. Propriedades como a capacidade de seguimento e de rejeição de ruído são importantes em problemas com sistemas variáveis no tempo, como é o caso dos sistemas CARA.

II.3.3.1 – Algoritmo Steepest Descent

Esse processo pode ser, simplificadamente, expresso por:

a) Perturbe os coeficientes de *H(z)* e teste se o valor do erro cresceu ou diminuiu
b) Se diminuiu, vá para o próximo coeficiente
c) Se aumentou, troque o sinal da mudança no coeficiente, e vá para o próximo coeficiente
d) Repita esse o procedimento

Essa aproximação, chamada de "*steepest descente algorithm"* é lenta e ineficaz, mas é a base para algoritmos mais eficazes como o LMS ("Least Mean Square") e o RLS ("Recursive Least Square"). Imagine uma bola sendo jogada dentro desta superfície a partir de algum ponto da superfície de erro mostrada na figura II.8. Quando solta, a bola irá rolar para baixo pela parede da superfície e eventualmente irá parar, depois de alguma oscilação, no fundo da superfície. Isso é exatamente o que o algoritmo procura realizar. Quando solta, a bola viaja na direção da mudança máxima de variação da inclinação (slope), ou gradiente ∇, da superfície de erro. Se examinarmos a posição da bola em instantes discretos, enquanto desce, sua nova posição é igual a anterior (um instante discreto anterior) mais alguma distancia abaixo em direção ao gradiente negativo da superfície. Ou seja, o algoritmo adiciona ao valor estimado atual uma porção do gradiente negativo da superfície de erro. Assim, o valor do erro quadrático "desce" pela superfície de erro, eventualmente chegando a seu ponto mais baixo, correspondendo aos coeficientes ótimos para minimização do erro. Matematicamente este conceito é definido por:

$$w(n+1) = w(n) - \mu \nabla w(n) \tag{2.24}$$

Onde ∇w é o gradiente da superfície de erro no ponto dado pelo vetor de pesos do coeficiente e μ é um número positivo que indica a velocidade da convergência. A porção do gradiente negativo a ser adicionado é conhecida como coeficiente de convergência.

A escolha do parâmetro μ é de extrema importância para o desempenho global do sistema adaptativo. Se μ for pequeno, a convergência será lenta, mas o erro de estado estacionário também será pequeno. Um valor maior de μ leva a velocidades maiores de convergência, mas implica em um erro de estado estacionário maior. Valores muito grandes de μ podem levar à instabilidade, já que podem introduzir realimentação no sistema. O critério de estabilidade deste algoritmo indica que[9][13]:

$$0 < \mu < \frac{2}{\lambda_{\max}} \tag{2.25}$$

onde $\lambda_{\max}$ é o maior valor característico da matriz R na eq.2.20.

II.3.3.2 – Algoritmo LMS (“Least Mean Square”)

O algoritmo LMS utiliza uma aproximação na qual o cálculo do valor do próximo coeficiente é baseado somente no valor da mostra do sinal atual. É uma variação estocástica do algoritmo Steepest Descent, onde o valor médio do coeficiente converge para uma solução ótima. O filtro LMS e suas variantes são estatísticos, pois trabalham com a expectativa de valores futuros.

A velocidade de convergência é maior para ruídos mais próximos do ruído branco com média zero e variância unitária. Como esse ruído possui seus valores característicos iguais a um, possui uma matriz de autocorrelação com diagonal um. Quanto mais “colorido” for o espectro do sinal de entrada, mais vagarosa será a convergência, devido ao espalhamento dos autovalores da matriz de correlação. Para o filtro LMS é necessário que exista um sinal de referência *d[n]* representando a saída do filtro.
A diferença entre o sinal de referência e a saída atual do filtro é o sinal de erro:

$$e(n) = d(n) - w^T(n) \cdot x(n) \tag{2.26}$$

O erro quadrático e seu valor esperado são dados por:

$$e^2(n) = (d(n) - w^T(n).x(n))^2 = d^2(n) - 2d(n)w^T(n)x(n) + w^T(n)x(n)x^T(n)w(n) \tag{2.27}$$

$$\begin{aligned} E(e^2(n)) &= E(d^2(n)) - E(2d(n)w^T(n)x(n)) + E(w^T(n)x(n)x^T(n)w(n)) \\ &= E(d^2(n)) - w^T(n)2E(d(n)x(n)) + w^T(n)E(x(n)x^T(n))w(n) \end{aligned} \tag{2.28}$$

Mais uma vez, o erro quadrático é função do vetor de coeficientes *w*, e possui apenas um mínimo global, que, teoricamente, pode ser alcançado se os valores esperados em 2.28 forem conhecidos. Retornando a abordagem II.3.2.1, os coeficientes devem ser movidos na direção “steepest descent”, isto é, no gradiente negativo da função de custo $\xi(n) = E(e^2(n))$ em relação ao vetor de coeficientes (a dependência do tempo *n* foi suprimida para simplificar a notação):

$$-\nabla_w E(e^2) = 2E(d \cdot x) - 2E(x \cdot x^T) \cdot w \tag{2.29}$$

onde ∇_w representa as derivadas parciais em relação ao vetor *w*, $\nabla_w = \left[\frac{\partial}{\partial w_1}, \frac{\partial}{\partial w_2}, \ldots, \frac{\partial}{\partial w_n} \right]^T$.

Os valores esperados na eq. 2.29, $E(d.x)$=p e $E(x.x^T)$=R, normalmente são estimados

utilizando-se um grande número de amostras de *d* e *x*. No algoritmo LMS, no entanto, como são utilizados os valores da amostra corrente, as seguintes aproximações são validas: $E(d \cdot x) \approx d \cdot x$ e $E(x \cdot x^T) \approx x \cdot x^T$, levando à equação de atualização dos coeficientes:

$$\begin{aligned} w^{novo} &= w^{antigo} + \mu/2(-\nabla_w E(e^2)) \\ &= w^{antigo} + \mu x(d - x^T w) \\ &= w^{antigo} + \mu x e \end{aligned} \tag{2.30}$$

As equações finais para o cálculo dos coeficientes são dadas por:

$$\hat{w}(n+1) = \hat{w}(n) - \mu x(n) e(n) \tag{2.31}$$

$$0 < \mu < \frac{2}{ME[x^2(n)]} \tag{2.32}$$

Custo Computacional	
Memória	2L+4
Multiplicação	2L+1
Adição	2L
Divisão	0

Tab. II.1 - L- ordem do filtro adaptativo

As figuras II.9 e II.10 seguintes mostram o acompanhamento do filtro LMS de ordem 6 e valores de $\mu = 0.05$ e $\mu = 0.01$, em relação a sistema exemplo $H(z)$

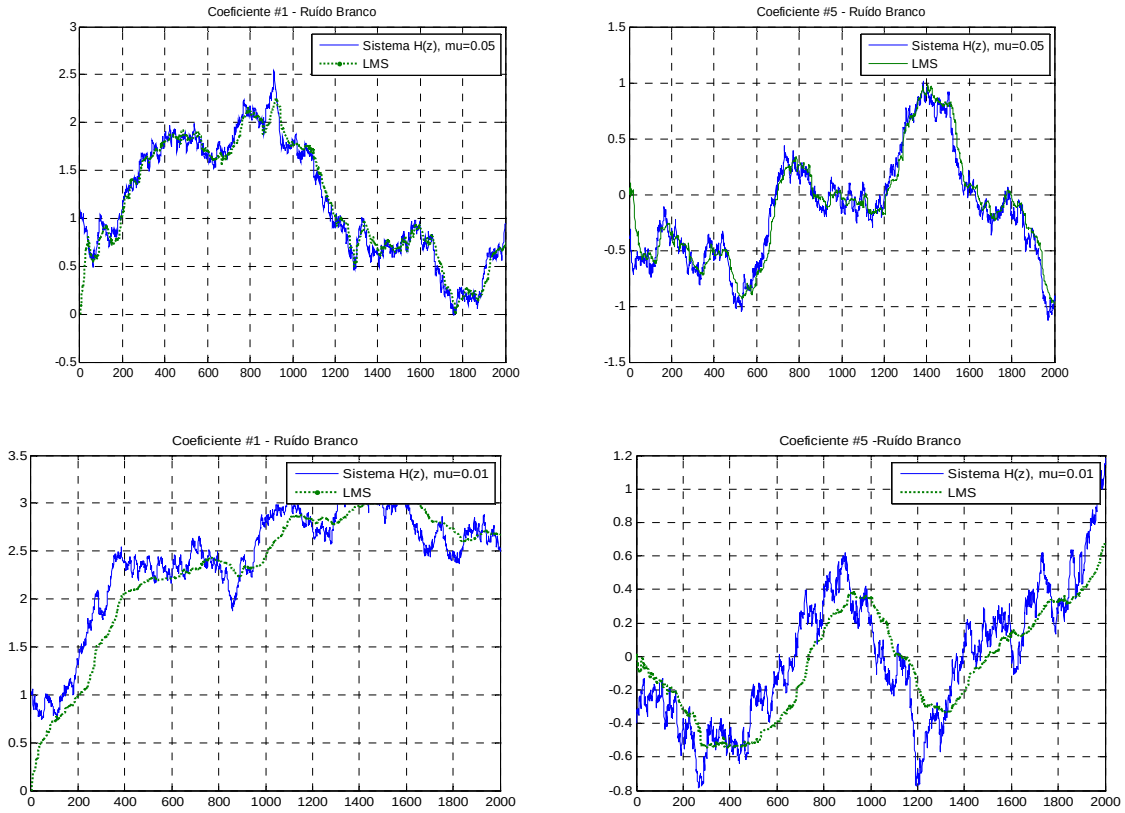


Fig II.9 – Acompanhamento do Algoritmo LMS – Ruído Branco – mu=0.05 e 0.01

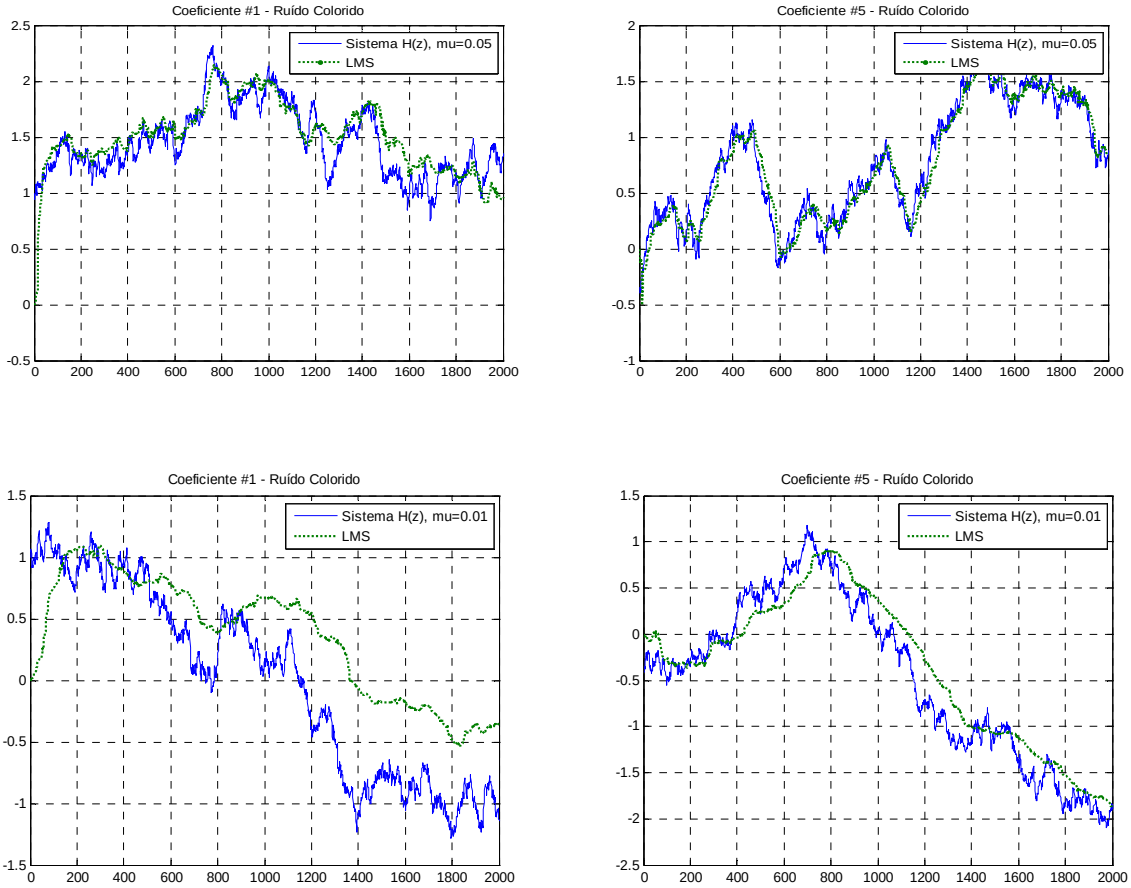


Fig II.10 – Acompanhamento do Algoritmo LMS – Ruído Colorido – mu=0.05 e 0.01

As simulações confirmam a sensibilidade ao parâmetro μ. O código, lms_track.m, encontra-se no apêndice A.

II.3.3.3 – Algoritmo NLMS (“Normalized Least Mean Square”)

De modo a garantir que o algoritmo LMS não venha a divergir, o critério de estabilidade dado pela eq.2.27 mostra que os limites de μ são inversamente proporcionais a potência do sinal de entrada. Em aplicações práticas essa potência não é conhecida, tornando impraticável determinar um valor adequado para μ. Esse problema pode ser contornado utilizando-se o valor de μ variante no tempo e normalizado:

$$\mu(n) = \frac{\mu}{\delta + M \cdot \hat{P}(n)} \tag{2.33}$$

onde $\hat{P}(n)$ é a potência instantânea estimada e δ uma pequena constante para evitar a divisão por zero. A potência é estimada usando-se uma janela com o mesmo tamanho do filtro adaptativo:

$$\hat{P}(n) = \frac{1}{M} \sum_{i=0}^{i=M-1} x^2(n-i) \tag{2.34}$$

Custo Computacional	
Memória	2L+7
Multiplicação	2L+4
Adição	2L+2
Divisão	1

Tab. II.2 - L- ordem do filtro adaptativo

II.3.3.4 – Algoritmo RLS (“Recursive Least Square”)

O algoritmo RLS é uma versão recursiva do algoritmo LMS. Em contraste com esse, o algoritmo RLS utiliza informações das amostras passadas do sinal de entrada (não apenas do sinal atual) para estimar a matriz de autocorrelação do vetor de entrada R(eq. 2.20). O filtro RLS é determinístico e baseia-se em dados já observados. Para diminuir a influência do sinal de entrada relativo a amostras adquiridas em um intervalo passado longo, um fator de peso para cada amostra é utilizado. Esse fator também é conhecido como fator de esquecimento. Mais uma vez o propósito é diminuir o MSE e a eq. 2.19 é agora expressa por:

$$\xi(n) = \lambda^{n-i} \cdot E(e^2(n)) \tag{2.35}$$

onde λ é uma constante positiva, $0 \le \lambda \le 1$, e indica que a influência dos valores passados irá cair exponencialmente. Quando λ=0, a estimação de R e p é igual ao do algoritmo LMS. O desenvolvimento encontra-se em [9] e a equação final é dada por:

$$w(n) = w(n-1) + k(n)(d(n) - x^T(n)w(n-1)) \qquad (2.36)$$

Custo Computacional	
Memória	L^2+2N+4
Multiplicação	$2L^2$+4N
Adição	$1.5L^2$+2.5N
Divisão	L

Tab. II.3 - L- ordem do filtro adaptativo, N – ordem do filtro estimador de *S*

As figuras II.11 e II.11a seguintes mostram o acompanhamento do filtro RLS com comprimento 6 e valores de $\lambda = 0.95$ e $\lambda = 0.8$, em relação a sistema exemplo $H(z)$

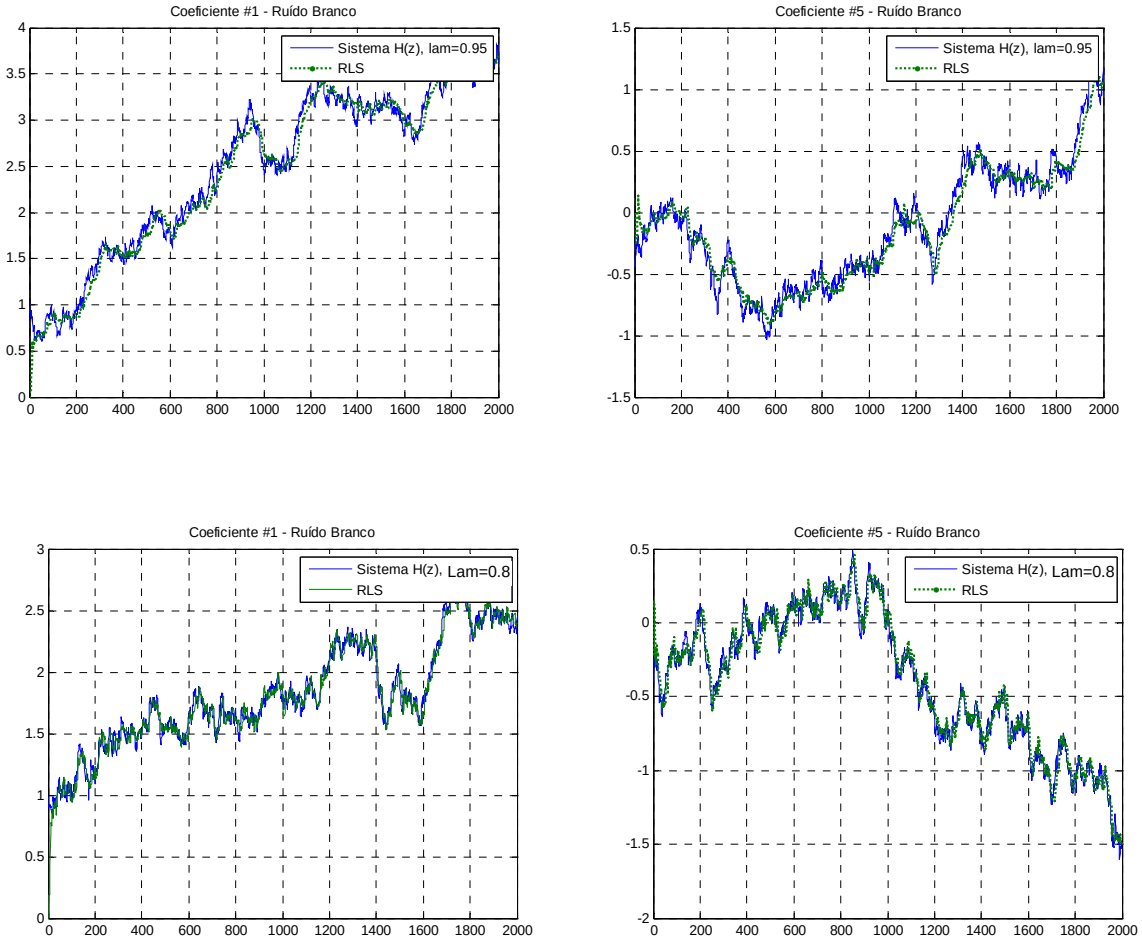
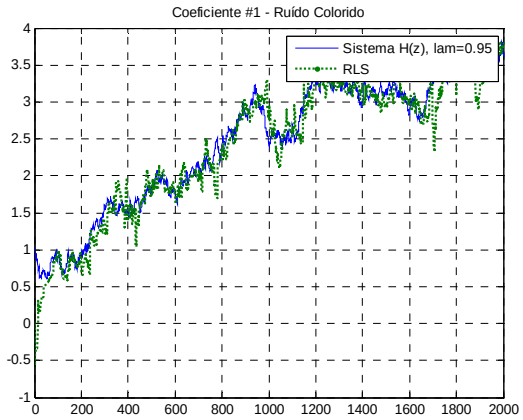
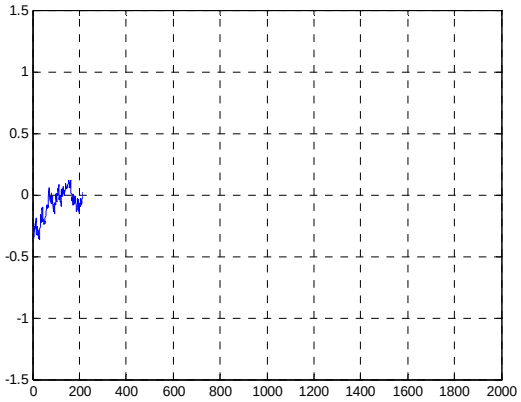


Fig II.11 – Acompanhamento do Algoritmo RLS – Ruído Branco


Coeficiente #1 - Ruído Colorido
Sistema H(z), lam=0.95
RLS
4
3.5
3
2.5
2
1.5
1
0.5
0
-0.5
-1
0
200
400
600
800
1000
1200
1400
1600
1800
2000


1.5
1
0.5
0
-0.5
-1
-1.5
0
200
400
600
800
1000
1200
1400
1600
1800
2000

Burgess [15] sugere a utilização deste algoritmo para compensar os efeitos do caminho secundário em sistemas CARA:

$$e(n) = d(n) - \sum_{i=0}^{N-1} s_i(n)\, y(n-i) \tag{2.37}$$

$$e(n) = d(n) - s(n) * [w^T(n) x(n)] \tag{2.38}$$

onde *n* é o índice de tempo, *s(n)* a resposta ao impulso do caminho secundário *S(z)*, * convolução, $w(n) = [w_0(n)\ w_1(n) \cdots w_{N-1}(n)]^T$ e $x(n) = [x(n)\ x(n-1) \cdots x(n-N+1)]^T$. *N* é a ordem do filtro. Assumindo que a função de custo $\xi(n) = E(e^2(n))$, o filtro adaptativo minimiza o erro quadrático segundo a eq. 2.30. Substituindo 2.38 em 2.30 temos então que

$$w(n+1) = w(n) - \mu x'(n) e(n) \tag{2.39}$$

$$x'(n) = s(n) * x(n) \tag{2.40}$$

Percebe-se que a forma final do algoritmo FXLMS é idêntica ao algoritmo LMS, substituindo-se $x(n)$ pela versão filtrada $x'(n)$. Esse algoritmo exige o conhecimento da resposta ao impulso do caminho secundário $S(z)$. A determinação da resposta ao impulso do caminho de cancelamento, ou caminho secundário, será abordada mais adiante no trabalho.

É importante notar que na eq. 2.39 o sinal (-) para aplicações acústicas é utilizado ao invés do sinal (+) convencional em algoritmos LMS. Isso ocorre porque em um sistema CARA o sinal de erro é $e(n) = d(n) + y'(n)$, em função do fato que o erro $e(n)$ é resultado de uma adição acústica ao invés de uma subtração elétrica[4].

Custo Computacional	
Memória	2L+2N+5
Multiplicação	2L+N+4
Adição	2L+N
Divisão	1

Tab. II.4 - L- ordem do filtro adaptativo, N – ordem do filtro estimador de *S*

II.3.4 – Implementação Prática do Algoritmo FXLMS para Sistemas CARA

Nesta seção será detalhada a utilização dos algoritmos LMS e sua variação FXLMS para uso nos filtros adaptativos e para identificação dos caminhos acústicos dos sistemas CARA. A figura II.12 mostra o sistema CARA feedforward. Na seção anterior foi mostrado que a modelagem do caminho secundário do sistema requer a utilização do filtro FXLMS. Uma solução é um filtro idêntico no caminho do sinal de referência[2], observado na figura II.12

como $\hat{s}(z)$. As equações a serem utilizadas são eq. 2.39 e eq. 2.40. A função de transferência do caminho secundário $S(z)$ é desconhecida e variante no tempo em função de alinhamento do alto-falante atuador, mudança de temperatura, variação do fluxo, entre outros. É necessário identificar essa função de transferência. Para tanto, são propostos dois métodos: identificação on-line ou off-line. De fato, a fase de identificação precede a fase de controle e permite estimar o caminho secundário [4][15].

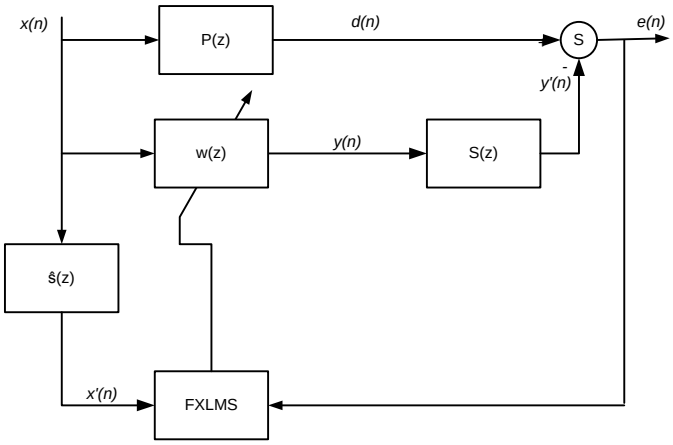


Fig. II.12 – Modelo CARA com algoritmo FXLMS

II.3.4.1 – Identificação Off-line

A fonte secundária (alto-falante de cancelamento) é alimentada com um sinal de ruído branco gerado internamente no próprio DSP. Essa operação é mostrada, esquematicamente, na figura II.13:

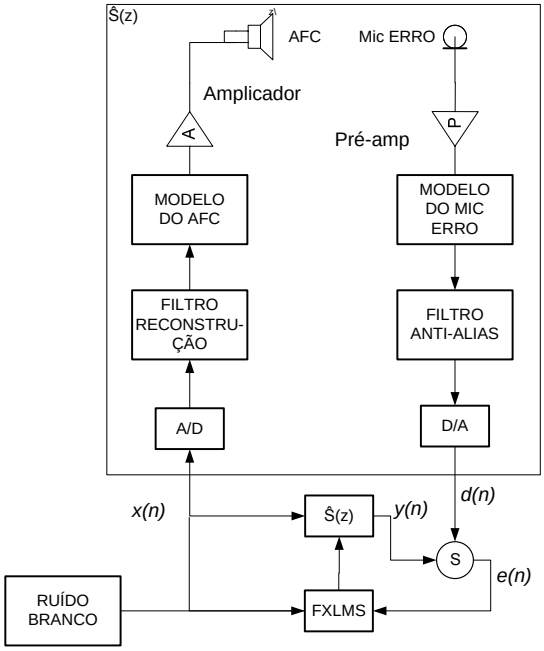


Fig. II.13 – Identificação off-line do caminho secundário

O diagrama da figura II.13 pode ser representado por:

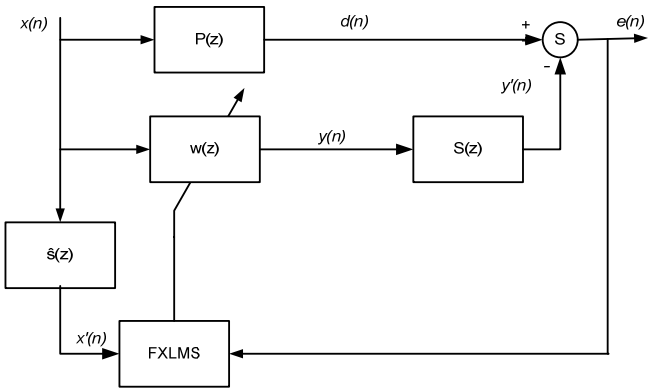


Fig. II.14 – Identificação off-line do caminho secundário (diagrama de blocos)

Neste processo de identificação percebem-se duas características que devem ser ressaltadas. Primeiramente, nota-se que os modelos dos atuadores, filtros, sensores e conversores já serão incluídos no modelo a ser identificado. Segundo, se as características dinâmicas do caminho secundário mudarem durante a fase de controle, elas não serão compensadas, diminuindo assim a eficiência do sistema. É mostrado que erros na identificação do caminho secundário diminuem a velocidade de convergência. No entanto para sistemas SISO a estabilidade ainda é assegurada para erros de fase de +/- 90º [2]. Os passos para identificação off-line do caminho secundário são os seguintes:

1. Tomar uma amostra do vetor de ruído branco gerado e assinalar o valor à $x(n)$
2. Enviar este sinal ao alto-falante
3. Ler o sinal $d(n)$ no microfone de erro micERRO
4. Aplicar o algoritmo LMS:
    a. Calcular $y(n)$ a partir do filtro $\hat{S}(z)$:

$$y(n) = \sum_{i=0}^{M-1} w_i(n)x(n-1)$$

    b. Computar o sinal de erro:

$$e(n) = d(n) - y(n)$$

    c. Atualizar coeficientes:

$$w_i(n+1) = w_i(n) + \mu x(n-i)e(n), \quad i = 0 \ldots M-1$$

    d. Retornar ao passo (a) até que o algoritmo convirja, ou seja, $e(n)$ minimizado.

Essa fase off-line deve ter duração suficiente para que os coeficientes sejam os mais estáveis possíveis, já que serão aproveitados na fase de processamento em tempo real do algoritmo CARA.

Possuindo agora o modelo estimado do caminho secundário o algoritmo FXLMS pode ser aplicado ao sistema mostrado na figura II.13. Os passos resumidos são os seguintes:

1. Tomar uma amostra dos microfones de referência e erro, $x(n)$ e $e(n)$
2. Computar $y(n)$:

$$y(n) = \sum_{i=0}^{M-1} w_i(n)x(n-i)$$

onde $w_i(n)$ é o coeficiente de índice i do filtro adaptativo W(z) no tempo n e M a ordem deste filtro.

3. Enviar o sinal $y(n)$ ao alto-falante de cancelamento
4. Computar $x'(n)$:

$$x'(n) = \sum_{i=0}^{M-1} w_i x(n-i)$$

onde w_i é o vetor de coeficientes estimado no processo off-line descrito acima.

5. Atualizar os coeficientes do filtro W(z) usando o algoritmo FXLMS:

$$w_i(n+1) = w_i(n) + \mu x'(n-i)e(n)$$

6. Realizar a próxima iteração no passo 1.

II.3.4.2 – Identificação On-Line – Método de Widrow e Stearns

Em alguns sistemas o modelo do caminho secundário pode variar no tempo e pode haver necessidade de um melhor desempenho, estabilidade e velocidade de convergência. Nestes casos pode ser realizada uma estimação do caminho secundário de forma on-line[17]. A figura II.15 mostra o esquema proposto Widrow e Stearns[19] para modelar diretamente o caminho secundário:

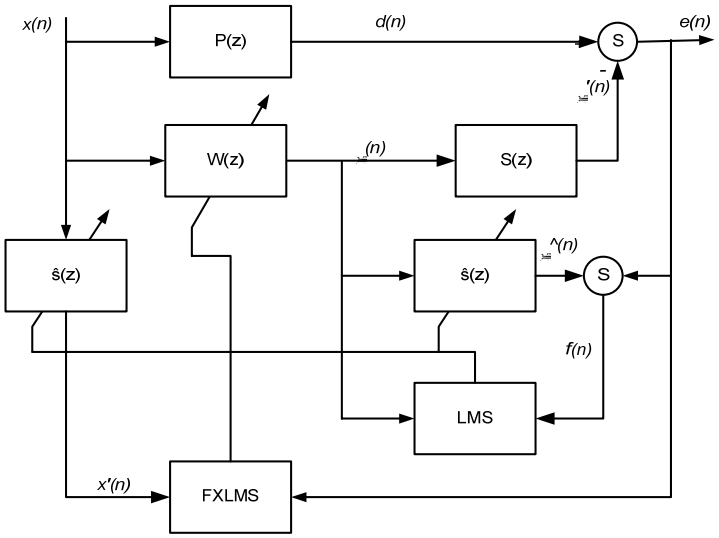


Fig. II.15 – Diagrama de blocos da identificação on-line

Esse modelo utiliza um filtro adaptativo $\hat{s}(z)$ conectado em paralelo com o caminho secundário $S(z)$. O sinal $y(n)$ gerado pelo filtro W(z) é aproveitado para modelar o caminho secundário. O filtro adaptativo $\hat{s}(z)$ é adaptado por um algoritmo LMS para minimizar $f(n)$. O sinal $f(n)$ é a diferença entre $e(n)$ e $\hat{y}(n)$. O sinal de referência $x(n)$ é então filtrado por uma cópia de $\hat{s}(z)$ para atualizar os coeficientes do filtro W(z) usando o algoritmo FXLMS. Da figura II.15, $f(n)$ é representado por

$$\begin{aligned} F(z) &= -E(z) - \hat{S}(z)Y(z) \\ &= -[P(z)X(z) - S(z)Y(z)] - \hat{S}(z)Y(z) \\ &= [S(z)W(z) - P(z) - \hat{S}(z)W(z)]X(z) \end{aligned} \tag{2.41}$$

Para que o modelo proposto na eq. 2.41 funcione e $f(z)$ tenda a zero, são necessárias algumas condições: $\hat{S}(z)$ deve ser de grande ordem, $x(n)$ deve ser persistente e diferente de zero e $P(z)$ e $S(z)$ sistemas invariantes no tempo[4]. Então a eq. 2.41, em seu estado estacionário, torna-se

$$\hat{S}^{o}(z) = S(z) - \frac{P(z)}{W(z)} \tag{2.42}$$

Essa solução não é adequada, já que $\hat{S}(z)$ só identifica $S(z)$ se $P(z) = 0$. Assim, o método utilizado para superar este problema é mostrado a seguir.

II.3.4.3 – Identificação On-Line – Método da Adição do Ruído Branco

Para contornar as limitações do método proposto por Widrow e Stearns, Eriksson e Allie[12] propuseram a adição de um ruído branco $g(n)$, gerado internamente, ao sinal secundário $y(n)$. Esse modelo é mostrado na figura II.16. O sinal que agora enviado ao filtro $\hat{S}(z)$ é apenas $g(n)$, e não é mais dependente de $y(n)$.

$$e(n) = d(n) - y'(n) - g'(n) \tag{2.43}$$

Como $y'(n) \equiv s(n) * y(n)$, componente do ruído secundário em função do ruído original, e $g'(n) \equiv s(n) * g(n)$, componente do ruído secundário em função do ruído adicionado, o resultado da estimação do filtro $\hat{S}(z)$ é dada por

$$\hat{g}'(n) = \hat{s}(n) * g(n) \tag{2.44}$$

onde $\hat{s}(n)$ é a resposta ao impulso do filtro $\hat{S}(z)$. O desenvolvimento é mostrado em [8], e chega-se as seguinte equações finais:

$$f(n) = e(n) - \sum_{i=0}^{N_s-1} \hat{s}_i(n) g(n-i) \tag{2.45}$$

$$\hat{s}(n+1) = \hat{s}(n) + \mu f(n) g(n-i) \tag{2.46}$$

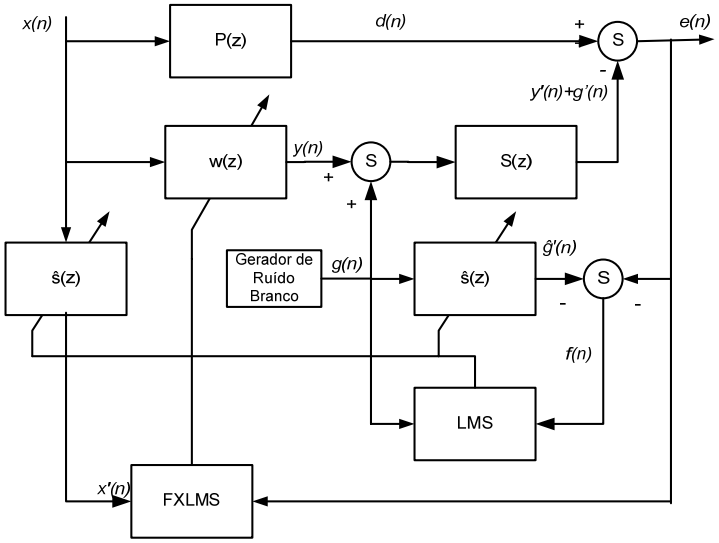


Fig. II.16 – Diagrama de identificação on-line com adição de ruído

II.3.5 – Simulação de Sistema Exemplo: Redução de Ruído Sobre Sinais de Voz

Para exemplificar alguns dos conceitos desenvolvidos acima, um programa genérico foi desenvolvido para simular o comportamento do filtro adaptativo em função da variação do tipo de ruído de entrada, μ e da ordem do filtro. O programa foi desenvolvido no Matlab© versão 6.0. O modelo utilizado para testes é o seguinte:

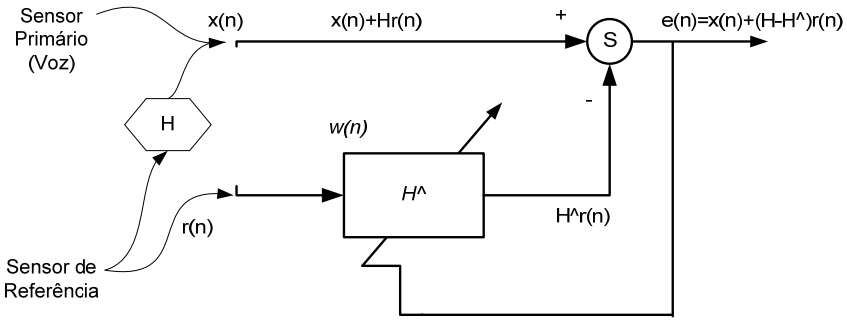


Fig. II.17 – Modelo para simulação de controle adaptativo

O modelo acima pode representar, por exemplo, a comunicação dentro de um veículo sendo afetada pela presença do ruído de fundo do motor. O sinal de interesse $x(n)$ é corrompido por um sinal aditivo de ruído não correlacionado $r(n)$ que passou através de um meio cujo modelo é dado por H, gerando sinal primário $x(n)+Hr(n)$. Um sensor de referência, colocado em um ponto diferente, capta o sinal de erro $r(n)$, não correlacionado com $x(n)$, mas correlacionado com $Hr(n)$. Para que o desempenho no cancelamento seja melhorado, deve-se modelar o caminho, H, do sinal da fonte de ruído ao sensor primário por meio de um filtro FIR. É um modelo de duas entradas onde o sensor primário não apenas capta o sinal desejado, mas também capta uma versão com delay (ou filtrada) do sinal de ruído.

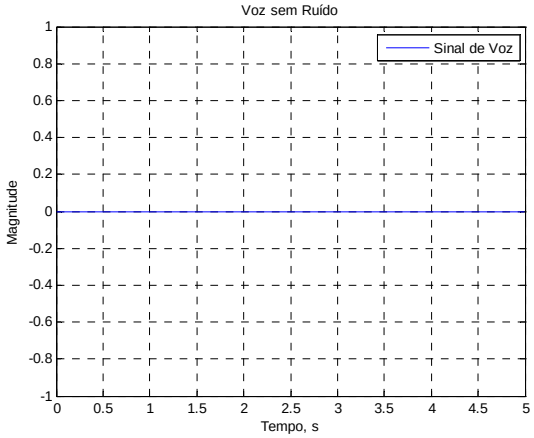
O filtro adaptativo processa o ruído $r(n)$, e seus coeficientes são ajustados pelo sinal de erro $e(n)$. Da figura II.17 temos que o componente de ruído principal é $(H-\hat{H})r(n)$. Esse termo pode ser minimizado, $H \approx \hat{H}$, o que leva a $e(n) \approx x(n)$ [2].

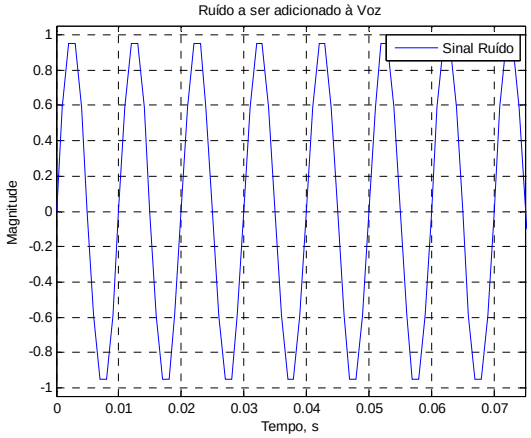
O programa de simulação possui as seguintes características:

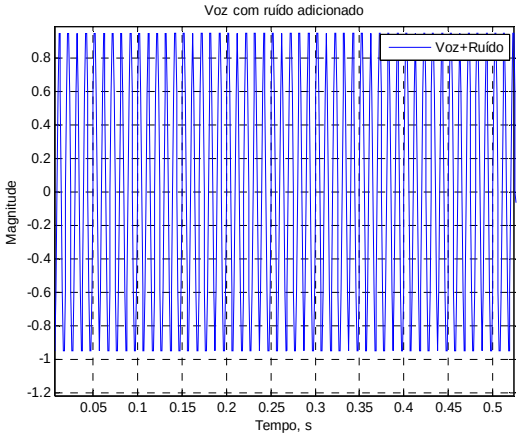
- Três opções de tipo de sinal de entrada: Valor constante, senoidal, randômico ou sinal previamente gravado em arquivo;
- As mesmas opções estão disponíveis para o sinal de ruído;
- Opções como número de amostras, freqüência de amostragem e freqüência do sinal;
- Opções de controle da ordem dos filtros e de μ;
- Possibilidade de mostrar os resultados graficamente ou gravá-los em arquivo.

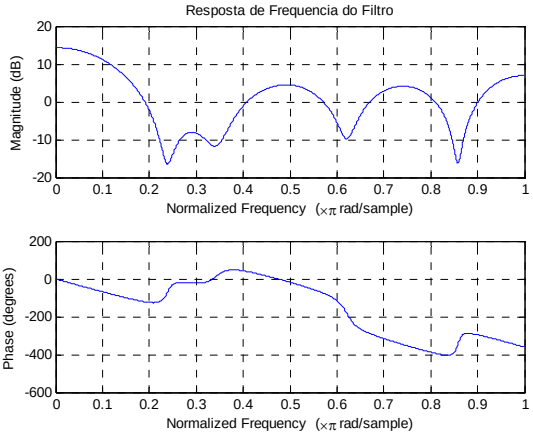
O código encontra-se no Apêndice A. As figuras II.18 a II.21 mostram o resultado de algumas simulações realizadas por meio do programa no MATLAB.

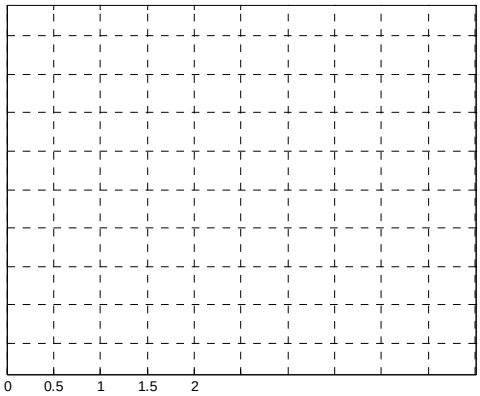
a. Sinal Constante e Ruído Senoidal



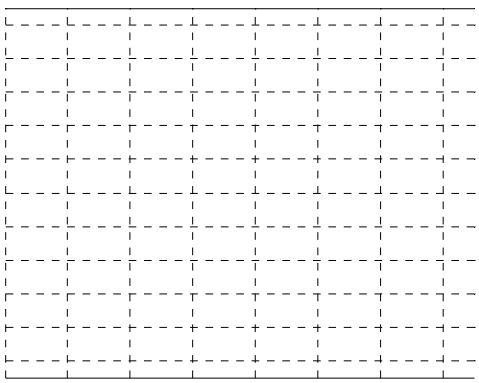




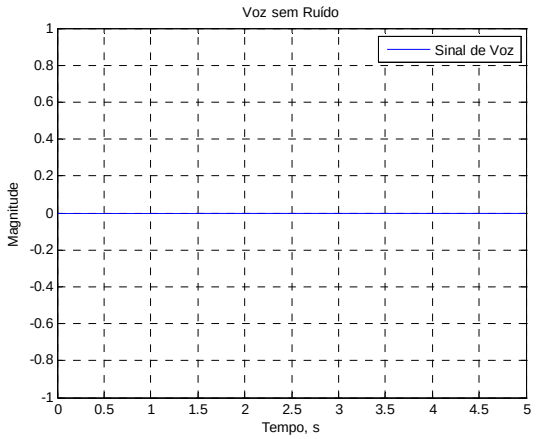


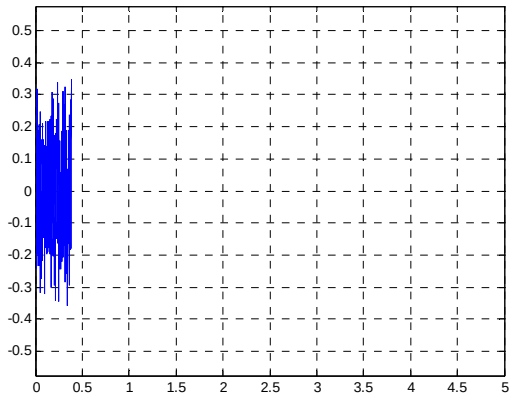


b. Sinal Senoidal e Ruído Senoidal



c. Sinal Constante e Ruído Branco





d. Sinal Gravado de Voz e Ruído de Motor (gravado em ambiente real)

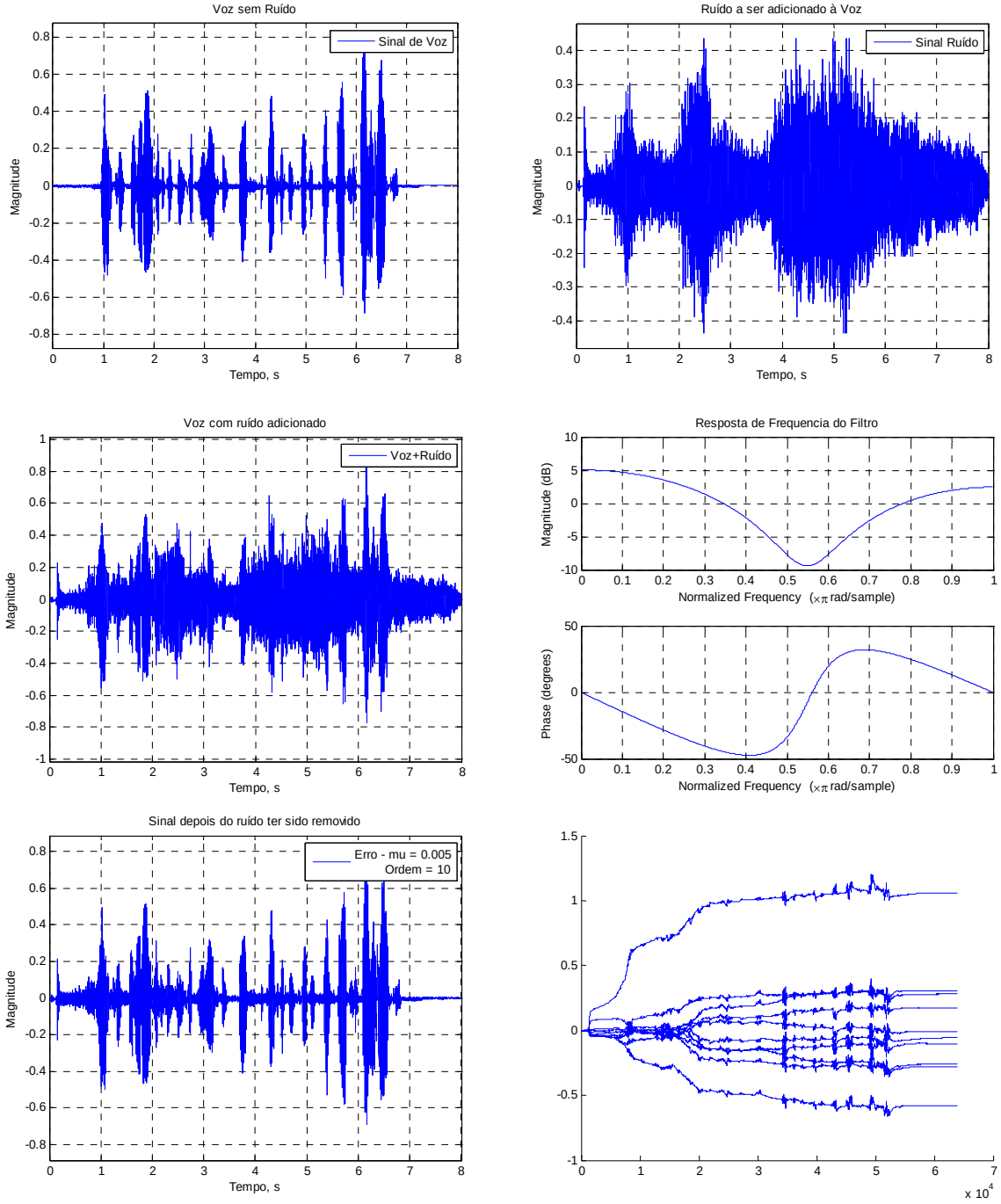


Fig. II.21 – Simulação: sinal de voz e ruído de motor

A análise dos gráficos das figuras II.18 a II.21, resultados obtidos nas simulações, mostram que o comportamento dos algoritmos está de acordo com a teoria anteriormente apresentada. Percebe-se a sensibilidade ao parâmetro μ: quanto maior seu valor, mais rápido os coeficientes se tornam estáveis. No entanto, o tempo de adaptação cresce e o valor de estado estacionário possui grande variação. Nota-se também que a ordem do filtro adaptativo deve ser bem estimada: acima de valores determinados o acréscimo na ordem do filtro não influencia na qualidade do sinal de saída filtrado, apenas aumenta o custo computacional.

CAPÍTULO III – Proposição de Sistema CARA Multicanal

III.1 – Sistema Proposto

O controle ativo de ruído proposto deve ser empregado em ambientes confinados de pequeno volume tais como cabines de aviões, carros de combate, automóveis, etc. O sistema é composto por um microfone de referência Ref, dois microfones de erro (Erro1 e Erro2) e dois alto-falantes(AF1 e AF2).

O modelo é um sistema CARA multicanal 1x2x2, onde os microfones de erro se localizam logo atrás da posição da cabeça (região de cancelamento) e os alto-falantes atuadores acima dos assentos. O ruído, gerado externamente e considerado uniformemente distribuído, é captado por meio de um microfone de referência localizado próximo a entrada da cabine. Tal dispositivo está esquematizado na figura III.1.

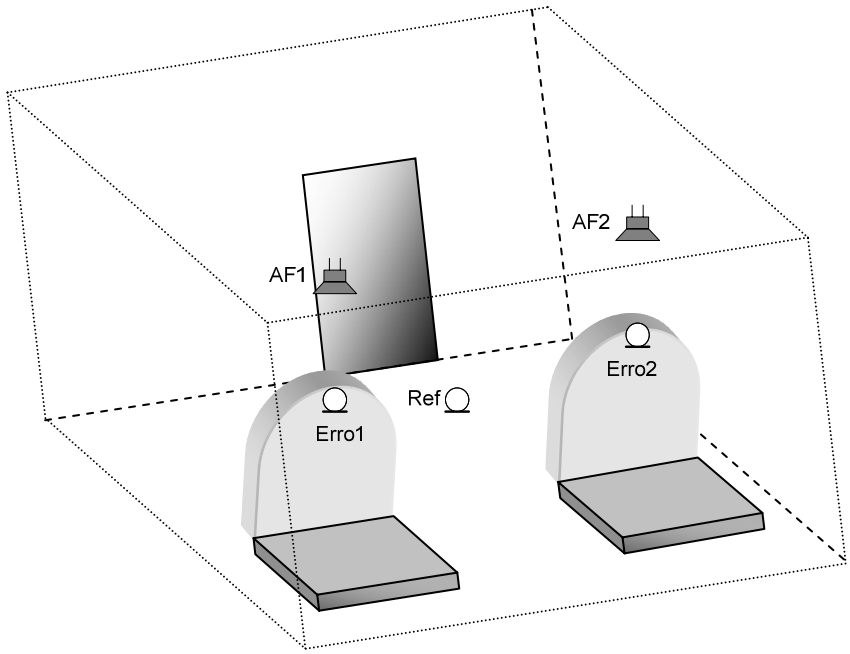


Fig. III.1 – Sistema CARA aplicado a ambientes de pequeno volume

Por extensão dos conceitos mostrados no Capítulo II, percebe-se que o campo sonoro em uma cabine assemelha-se a um duto de grandes dimensões e que, para seu controle, será necessário um número maior de sensores e atuadores.

III.1.1 – Restrições do Sistema Proposto

Como já exposto em I.2, o controle de ruídos em espaço tridimensional está diretamente ligado ao volume e a freqüência de interesse, refletindo no desempenho e controle do sistema. Como regra geral, o número de fontes secundárias deve ser igual ao número de modos acústicos do ambiente[4].

A escolha do tipo de cancelamento do campo sonoro, global ou local, é de fundamental importância, já que dela dependerá a escolha dos algoritmos e complexidade do sistema.

O sistema proposto possuí as seguintes restrições:

- Freqüências de interesse: < 500 Hz;
- Cancelamento: local;
- Tipo de ruído: banda larga (“broaband”);
- O sinal de referência não é afetado pelas fontes secundárias: o problema de realimentação não será tratado.

III.2 – Algoritmo e Modelo

Usualmente, os modelos CARA multicanais são descritos pelo número de atuadores e sensores presentes no sistema. O número de sensores será representado por J sensores de referência, K filtros adaptativos paralelos e M

um vetor que contém todos os vetores dos K coeficientes dos K filtros adaptativos de ordem L.

$$w(n) \equiv [w_1^T(n)\, w_2^T(n) \cdots w_k(n)]^T \tag{3.04}$$

onde

$$w_k(n) \equiv [w_{k,0}^T(n)\, w_{k,1}^T(n) \cdots w_{k,L-1}(n)]^T,\ k = 1, 2, \ldots, K \tag{3.05}$$

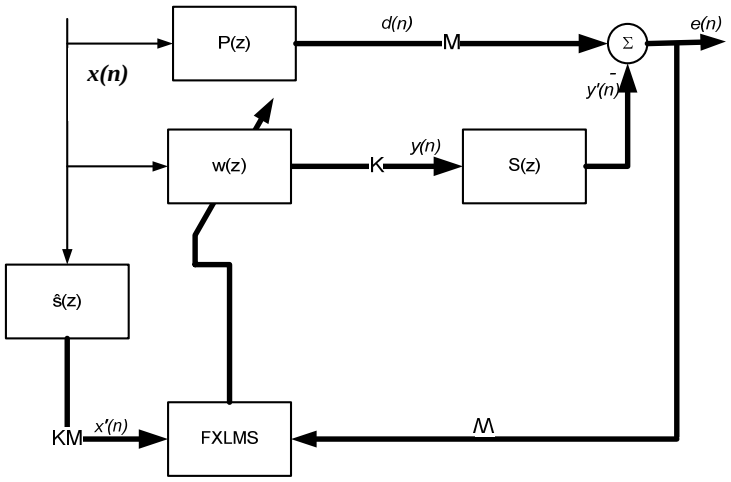


Fig. III.2 – Diagrama de blocos de um sistema CARA multicanal

Os sinais de saída para os atuadores $y_k(n)$ são obtidos pela filtragem do sinal de referência $x(n)$ pelo filtro FIR correspondente $w_k(n)$. Como para o modelo proposto só existe um sinal de referência, $x(n)$ pode ser descrito por

$$x(n) \equiv [x(n)\ x(n-1) \cdots x(n-L+1)]^T \tag{3.06}$$

Na forma matricial

$$y(n) = X^T(n) w(n) \tag{3.07}$$

onde $X(n)$ é a matriz $KLxK$ definida por

$$X(n) = \begin{bmatrix} x(n) & 0 & \cdots & 0 \\ 0 & x(n) & 0 & \vdots \\ \vdots & 0 & \ddots & 0 \\ 0 & \cdots & 0 & x(n) \end{bmatrix} \tag{3.08}$$

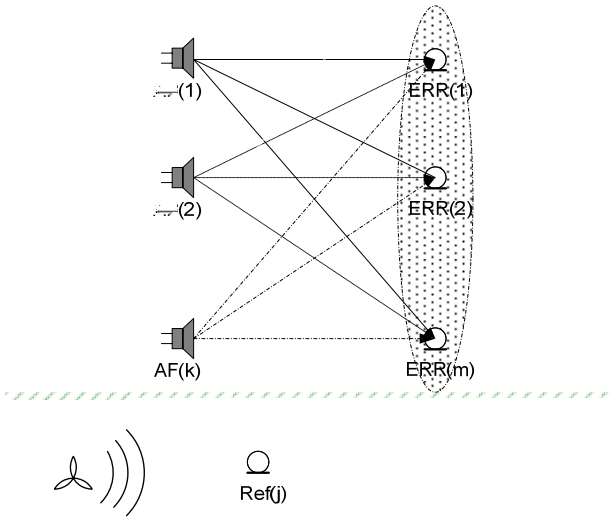
Do diagrama da figura III.2 tem-se que

$$e(n) = d(n) - y'(n) \tag{3.09}$$

$$y'(n) = S(n) * y'(n) \tag{3.10}$$

onde $d(n) \equiv [d_1(n)\ d_2(n) \cdots d_M(n)]^T$ e $y'(n) \equiv [y'_1(n)\ y'_2(n) \cdots y'_M(n)]^T$. A matriz S, MxK, representa a função de resposta ao impulso dos vários caminhos secundários envolvidos como representado na figura III.3.

$$S(n) = \begin{bmatrix} s_{11}(n) & s_{12}(n) & \cdots & s_{1k}(n) \\ s_{21}(n) & s_{22}(n) & \cdots & s_{2k}(n) \\ \vdots & \vdots & \ddots & \vdots \\ s_{M1}(n) & s_{M2}(n) & \cdots & s_{MK}(n) \end{bmatrix} \tag{3.11}$$



Combinando as equações 3.07 e 3.10 temos que

$$e(n) = d(n) - S(n) * [X^T(n) w(n)] \tag{3.13}$$

Baseado nos princípios anteriormente descritos, a solução ótima w^o pode ser obtida pela minimização da função de custo $\xi(n)$ empregando-se o ajuste dos coeficientes dos filtros por meio do algoritmo steepest descent (eq. 2.24) ou seus variantes

$$W(n+1) = w(n) - \frac{\mu}{2} \nabla \hat{\xi}(n) \tag{3.14}$$

O cálculo do gradiente deve ser realizado por todos os subvetores de $w(n)$, ou seja, $w_k(n)$, aplicando-se o algoritmo FXLMS

$$w\ (n+1) = w(n) + \mu X'(n) e(n) \tag{3.15}$$

Aplicando a eq. 3.15 para os K filtros chega-se a

$$w_k(n+1) = w_k(n) + \mu \sum_{m=1}^{M} X'_{km}(n) e_m(n),\ k = 1, 2, \ldots, K \tag{3.16}$$

que vem a ser a equação final de ajuste dos coeficientes dos filtros para um sistema multicanal utilizando o algoritmo FXLMS.

Uma questão importante reside na estabilidade do sistema. As distâncias entre as fontes secundárias e seus respectivos sensores devem ser controladas. A distância entre uma fonte secundária e seu respectivo sensor deve ser menor que a distância entre este sensor e outra fonte secundária [24]. Na figura III.4, o sistema é estável, obedecendo os critérios anteriores. Na figura III.5, o sistema é instável já que, apesar do conjunto AF1-ERR1 estar obedecendo ao critério de estabilidade d1<d2, no par AF2-ERR2 isso não ocorre, d3<d4.

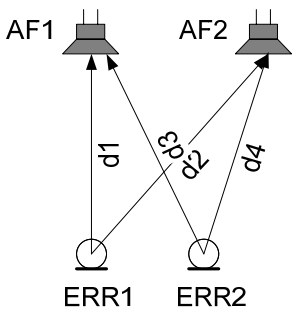


Fig. III.4 – Estabilidade do sistema – estável

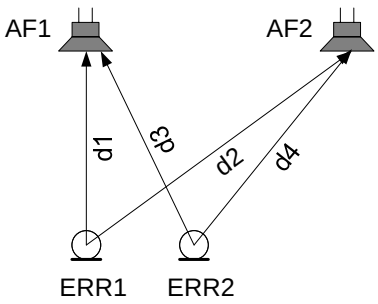


Fig. III.5 – Estabilidade do sistema – não estável

O diagrama da figura III.6 detalha o sistema proposto na configuração 1x2x2.

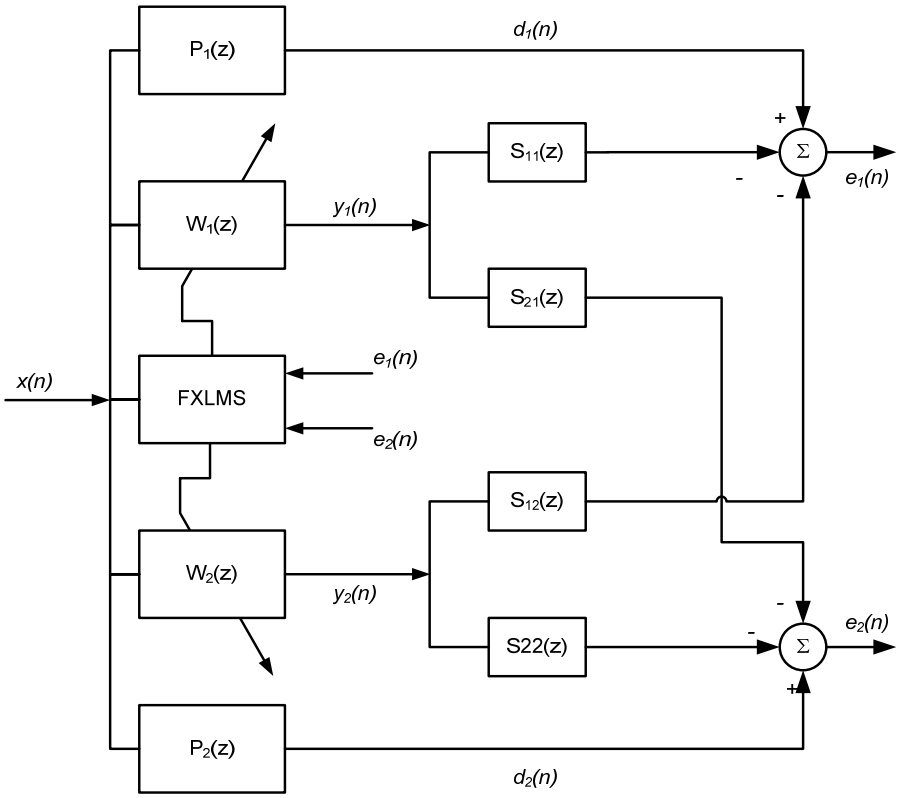


Fig. III.6 – Sistema CARA, 1x2x2

$x(n) \rightarrow$ sinal de referência

$y_1(n)$ *e* $y_2(n) \rightarrow$ sinais de cancelamento gerados pelos filtros adaptativos $\mathrm{W}_1(z)$ e $\mathrm{W}_2(z)$, respectivamente

$e_1(n)$ e $e_2(n) \rightarrow$ sinais de erro medidos pelos dois sensores de erro

$\mathrm{P}_1(z)$ e $\mathrm{P}_2(z) \rightarrow$ caminhos primários da fonte de ruído aos sensores de erro

$S_{11}(z) \rightarrow$ caminho secundário entre $y_1(n)$ e seu respectivo sensor de erro

$S_{22}(z) \rightarrow$ caminho secundário entre $y_2(n)$ e seu respectivo sensor de erro

$S_{12}(z) \rightarrow$ caminho secundário entre $y_1(n)$ e o outro sensor de erro

$S_{21}(z) \rightarrow$ caminho secundário entre $y_2(n)$ e o outro sensor de erro

Aplicando o algoritmo FXLMS e incluindo as estimativas $\hat{S}_{mk}$ dos caminhos secundários $S_{mk}(z), m = 1,2$ e $k = 1,2$, o diagrama da figura III.6 é representado na figura III.7, por

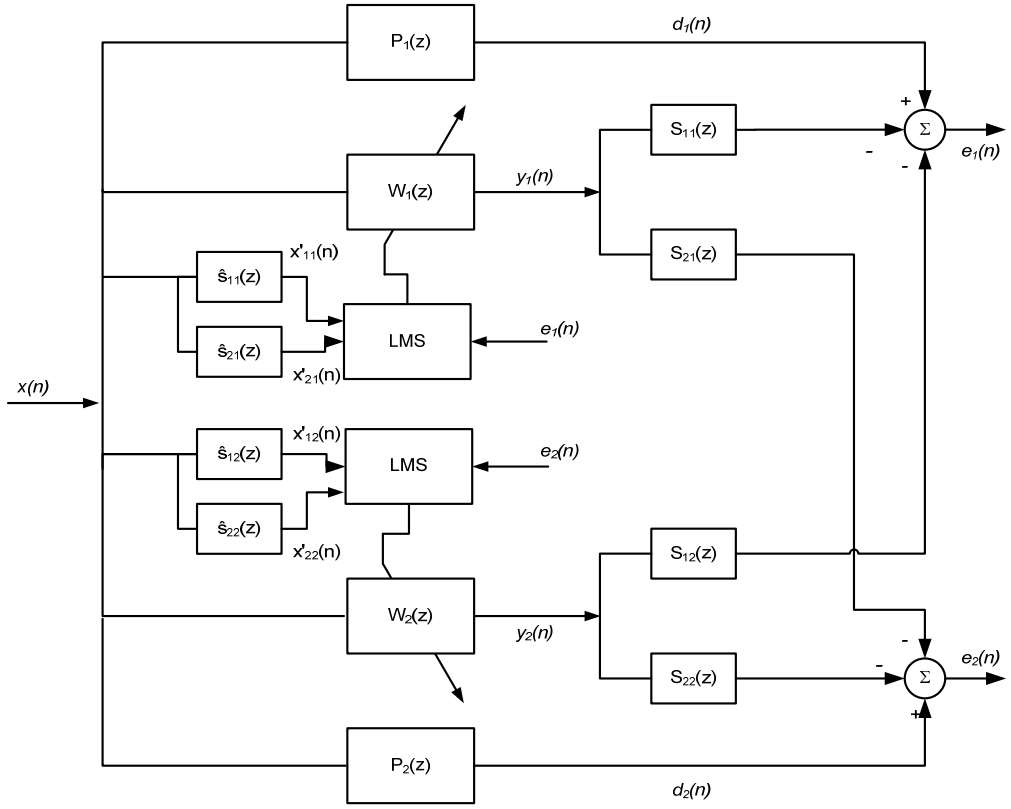


Fig. III.7 – Modelo FXLMS do Sistema CARA 1x2x2

III.3 – Estimação dos Caminhos Secundários

Os sinais de erro são captados em uma região onde estão atuando todas as fontes secundárias do sistema como mostrado na figura III.4. A estimação dos caminhos $S_{mk}(z)$ deve ser compensada para todos os caminhos $y_k(n)$ - $e_m(n)$, $m = 1,2,\ldots,M$ e $k = 1,2,\ldots,K$

A figura III.8 mostra o diagrama de blocos do sistema CARA 1x2x2 proposto.

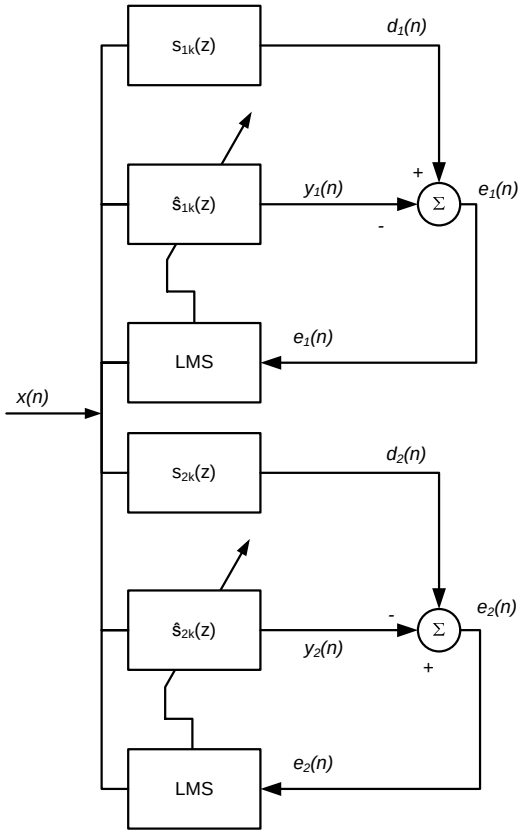


Fig. III.8 – Modelagem dos caminhos secundários do sistema 1x2x2

onde $k = 1,2$ e $m = 1,2$.

$s_{mk}(z) \rightarrow$ caminho secundário do sensor m ao atuador k

$\hat{s}_{mk}(z) \rightarrow$ filtro adaptativo do caminho secundário do sensor m ao atuador k

$x(n) \rightarrow$ ruído branco gerado pelo sistema

$dm(n) \rightarrow$ saída de $s_{mk}(z)$

$ym(n) \rightarrow$ saída de $\hat{s}_{mk}(z)$

$em(n) = dm(n) - ym(n) \rightarrow$ usado para atualizar $\hat{s}_{mk}(z)$

A implementação em software se dará então em duas fases: a primeira quando o sistema modela os caminhos secundários, que é a fase de treinamento ou off-line. Na segunda fase, utilizando os coeficientes da primeira fase, o sistema entra no modo contínuo ou on-line.

III.4 – Simulação do Sistema na Linguagem C e MATLAB

O sistema proposto será simulado utilizando a linguagem C para implementar o filtro e a simulação do algoritmo FXLMS. Os coeficientes serão gerados no MATLAB por meio da

ferramenta FDATOOL e os resultados gráficos gerados no programa C. O código C seguirá o seguinte pseudocódigo:

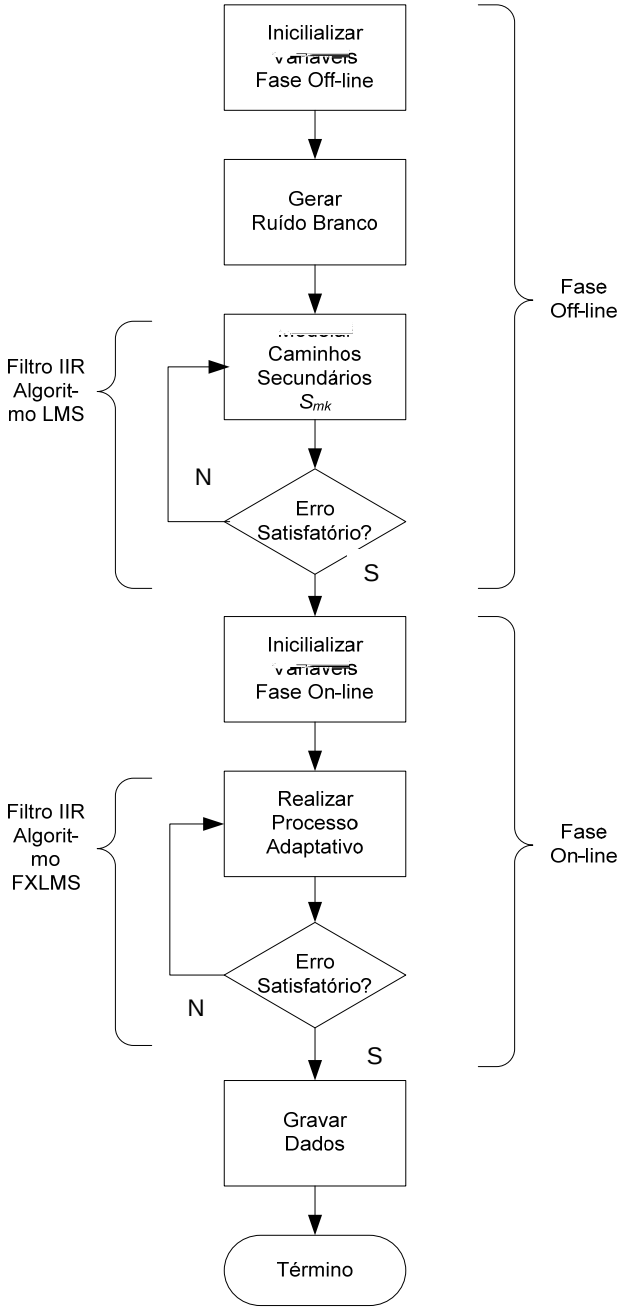


Fig. III.9 – Fluxograma do programa de simulação do Sistema CARA proposto

III.4.1 – Resultados da Simulação

O programa em Linguagem C que simula o sistema proposto, figura III.7, e que foi implementado segundo o fluxograma da figura III.9, pode ser encontrado no Apêndice B. Na simulação foram variados os parâmetros μ, velocidade de adaptação, ordem dos filtros e tipo de sinal de entrada. Os dados gravados em arquivos foram recuperados e analisados no MATLAB.

III.4.2 – Fase off-line

Segundo o diagrama da figura III.8, determina-se as seguintes relações:

a) $d1(n)$ é obtido pela filtragem de $x(n)$. Para tal foi implementado um filtro IIR, cujos coeficientes foram calculados usando algoritmo LMS;
b) $y_1(n)$ é calculado por meio de um filtro FIR e coeficientes calculados segundo o algoritmo FXLMS;
c) da mesma forma são calculados $d_2(n)$ e $y_2(n)$;
d) o ruído branco é gerado internamente.

As simulações para o processo off-line, utilizando uma freqüência de amostragem $f_s = 8kHz$, indicaram que com 10.000 interações os coeficientes já estavam estáveis. Tal situação indica que para o modelo proposto, o tempo da Fase Off-line estaria em torno de 1.25 segundos $(1/f_s * 10000)$. Os filtros IIR e FIR foram calculados segundo suas equações características[13][21]:

Filtro IIR:

$$y(n) = \sum_{i=0}^{ntap} a_i.x(n-i) - \sum_{j=1}^{nb} b_j.y(n-j) \qquad (3.17)$$

onde se $ntap = nb = n$, a função de transferência é dada por

$$H(z) = \frac{Y(z)}{X(z)} = \frac{a_o + a_1 Z^{-1} + \cdots + a_n Z^n}{1 + b_1 Z^{-1} + \cdots + b_n Z^{-n}} \qquad (3.18)$$

Filtro FIR:

$$y(n) = \sum_{i=0}^{ntap-1} w_i x(n-i) \qquad (3.19)$$

Os gráficos seguintes mostram o resultado das simulações com combinações dos parâmetros: ordem do filtro e μ. As figuras III.10 a III.15 são resultados da simulação de um sinal de entrada $x(n)$ puramente senoidal de freqüência igual a 200Hz. As figuras III.16 a III.25 mostram o resultado de simulações de um sinal de entrada de mesma freqüência mas com SNR=20dB.
A interface gráfica foi desenvolvida no ambiente C++ Builder 5.0® da Borland.

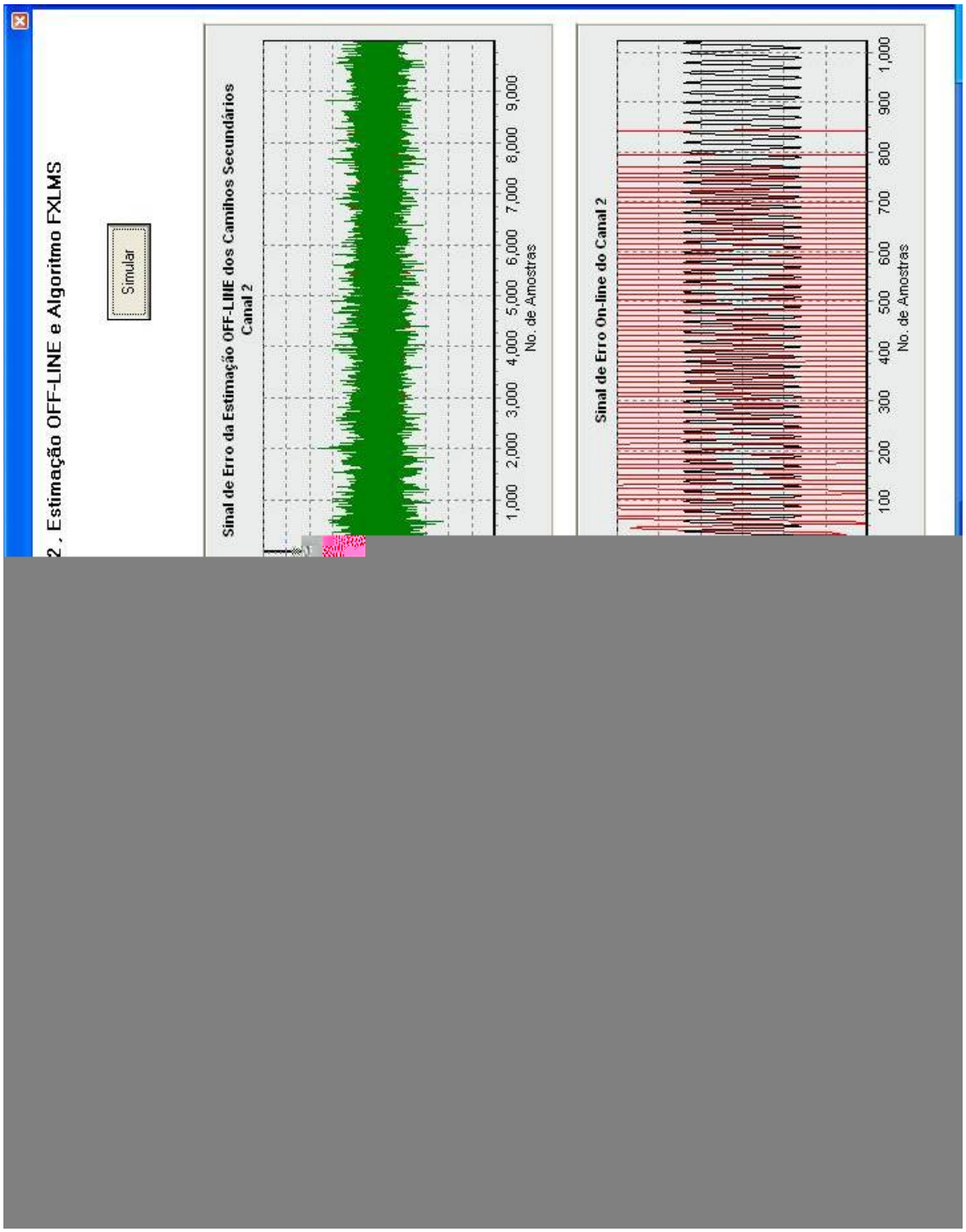


Fig. III.10 – Simulação CAR 1

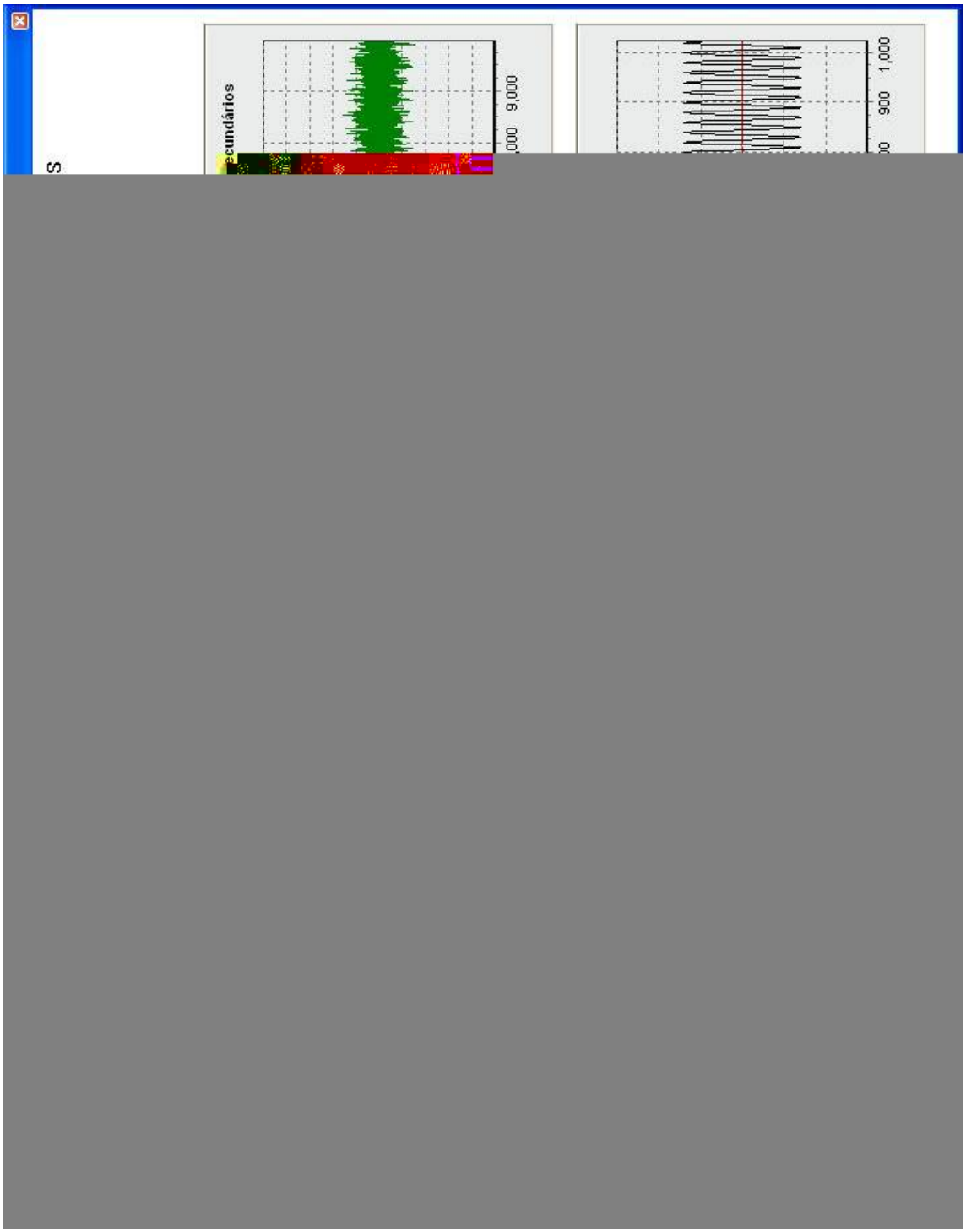


Fig. III.11 – Simulação CAR 2

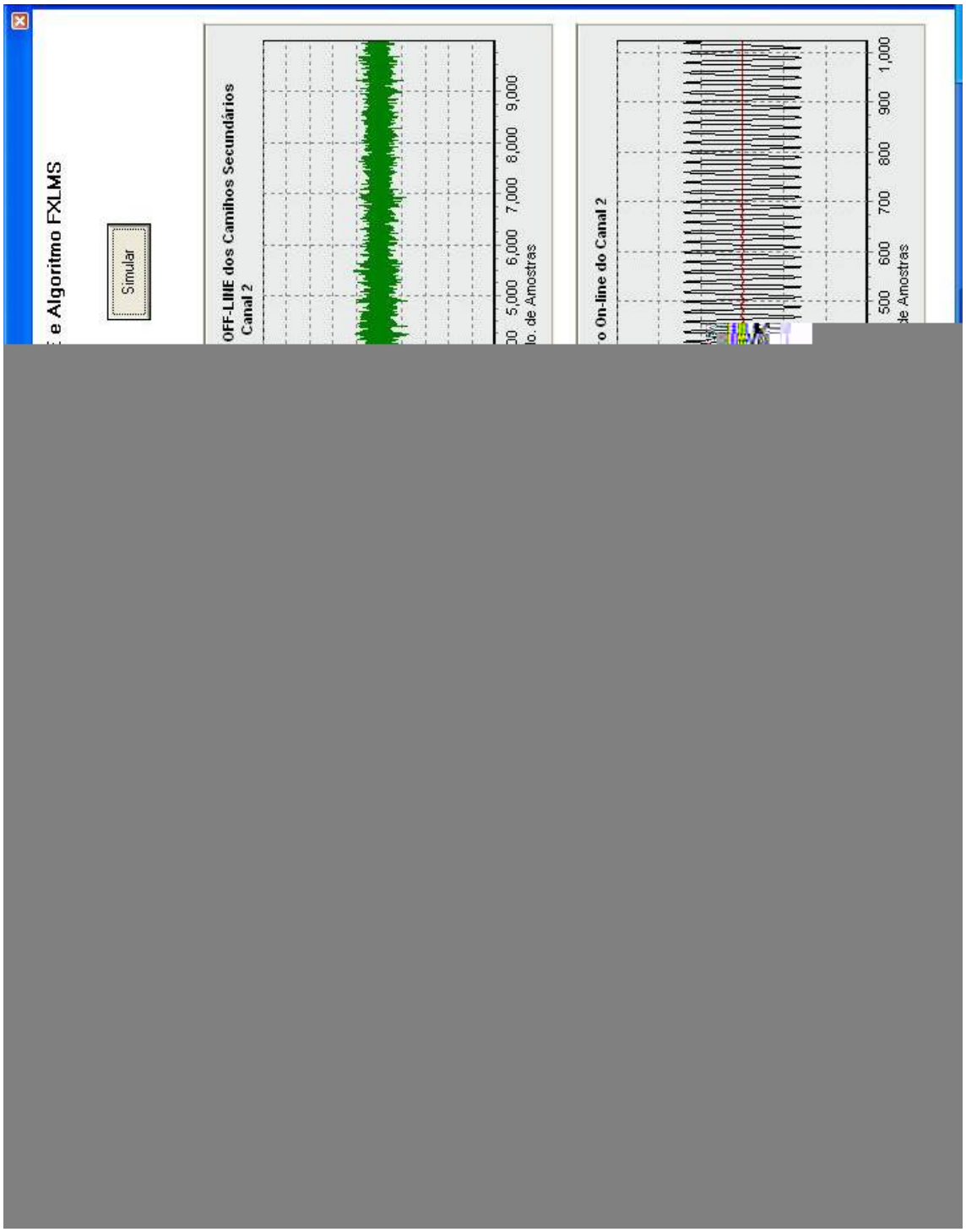


Fig. III.12 – Simulação CAR 3

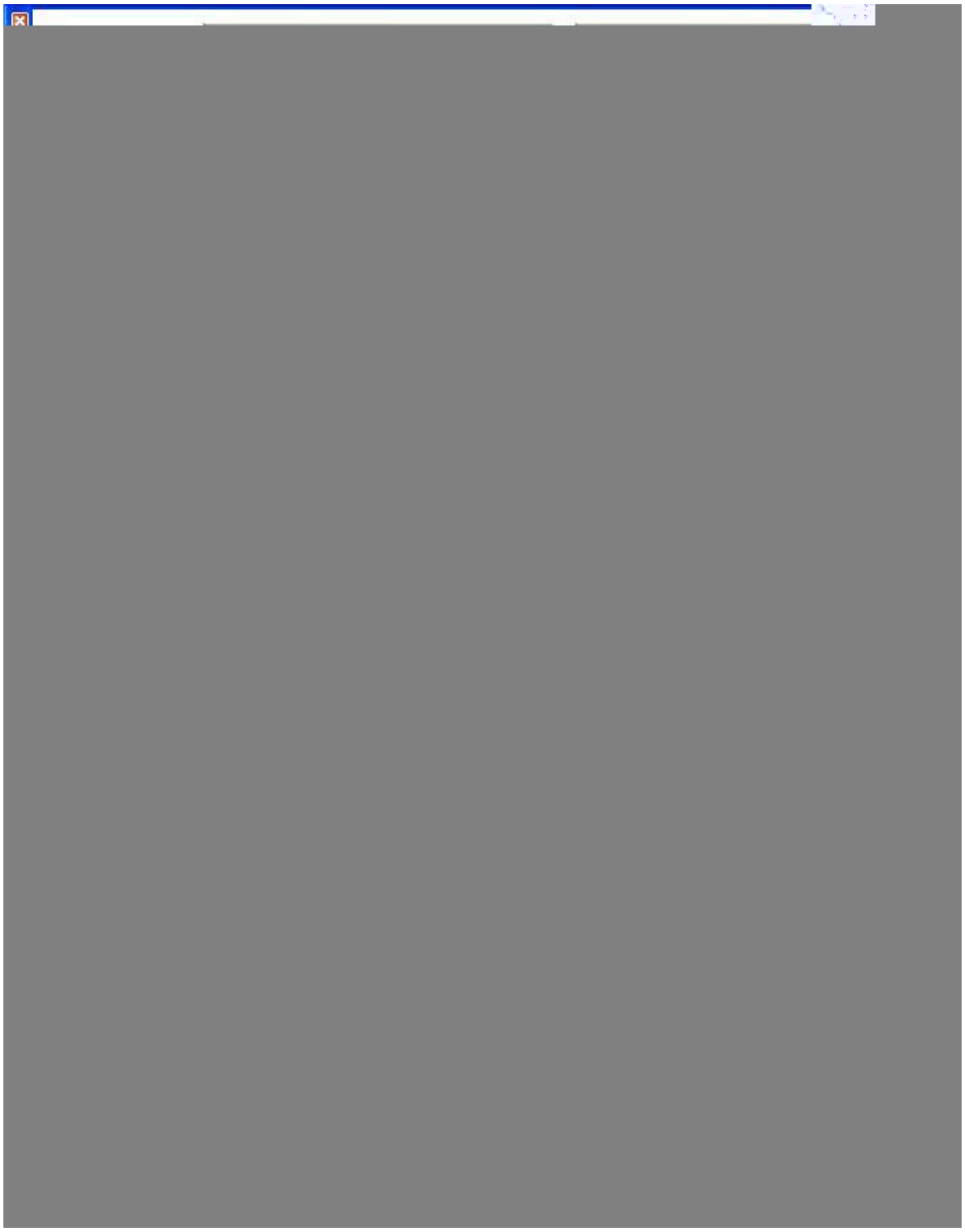


Fig. III.13 – Simulação CAR 4

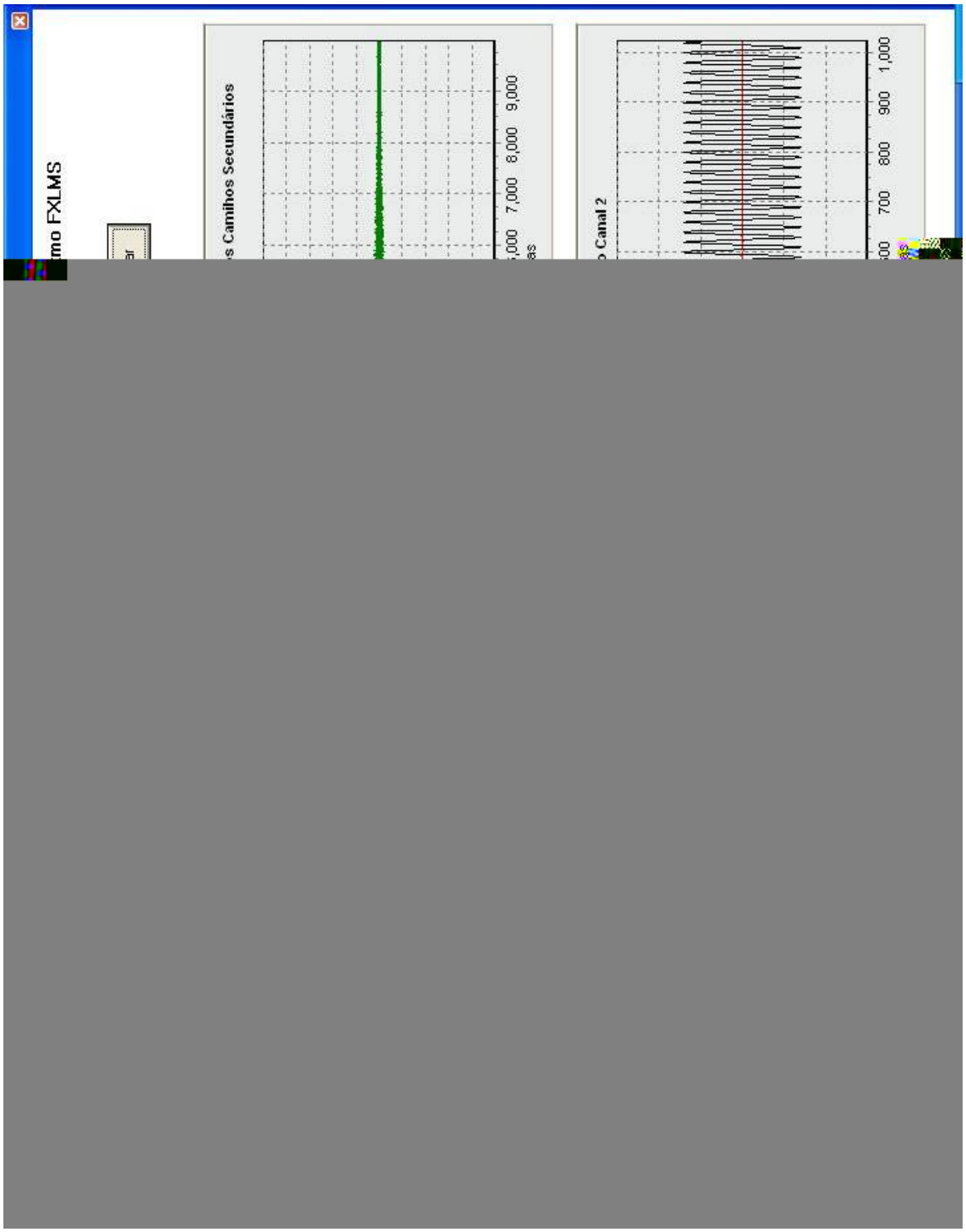


Fig. III.14 – Simulação CAR 5

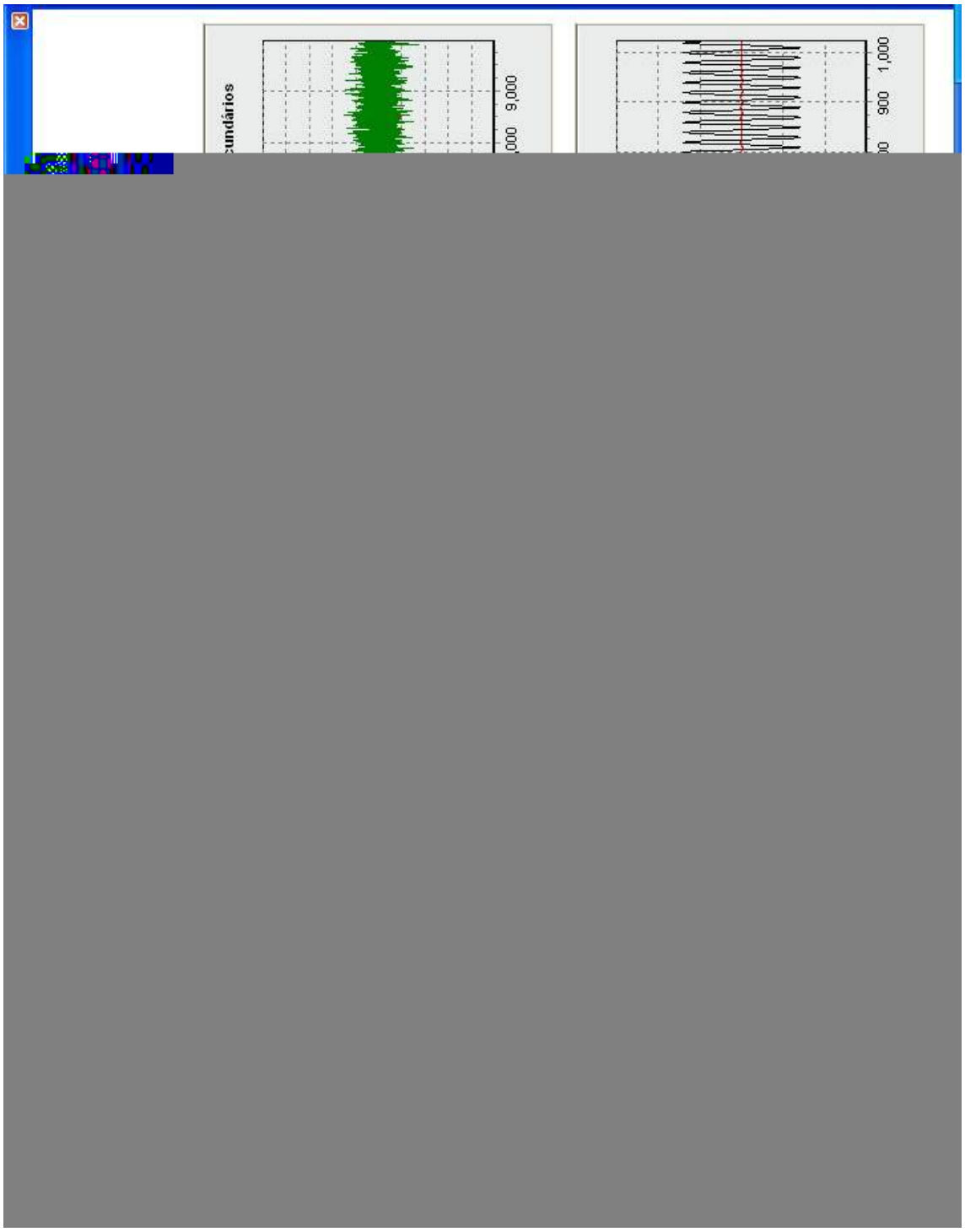


Fig. III.15 – Simulação CAR 6

Dos resultados das simulações apresentados nas figuras III.10 a III.15, concluí-se que:

- A ordem do filtro para estimar os coeficientes dos caminhos secundários influencia de maneira decisiva na qualidade do sinal de erro em seu estado estacionário;
- A variação do parâmetro *mu* nas simulações influenciou na velocidade de acomodação do sinal de erro, no entanto, apenas grandes variações deste parâmetro, na ordem de 10 vezes, são notadas no tempo de resposta do sistema na fase on-line;
- Não se levando em conta a qualidade do sinal de erro, a simulação numérica dos caminhos secundários se manteve estável com grandes variações dos parâmetros μ e da ordem do filtro, sem apresentar problemas de overflow ou instabilidade.

III.4.3 – Fase on-line

Na fase on-line, já conhecidos os parâmetros dos caminhos secundários calculados na fase off-line, desejam-se obter os seguintes sinais:

a) $d_1(n)$ pela filtragem de $x(n)$ por $P_1(z)$, utilizando-se um filtro IIR;

b) $y_1(n)$ pela filtragem de $x(n)$ por $W_1(z)$, utilizando-se um filtro FIR, segundo o algoritmo FXLMS;

c) da mesma forma são calculados $d_2(n)$ e $y_2(n)$.

Os gráficos das figuras III.16 a III.21 mostrados adiante, apresentam o resultado das simulações com combinações dos parâmetros: ordem do filtro e μ para sinais puramente senoidais. Para sinais com SNR=20dB os resultados estão apresentados nas figuras III.22 a III.25.

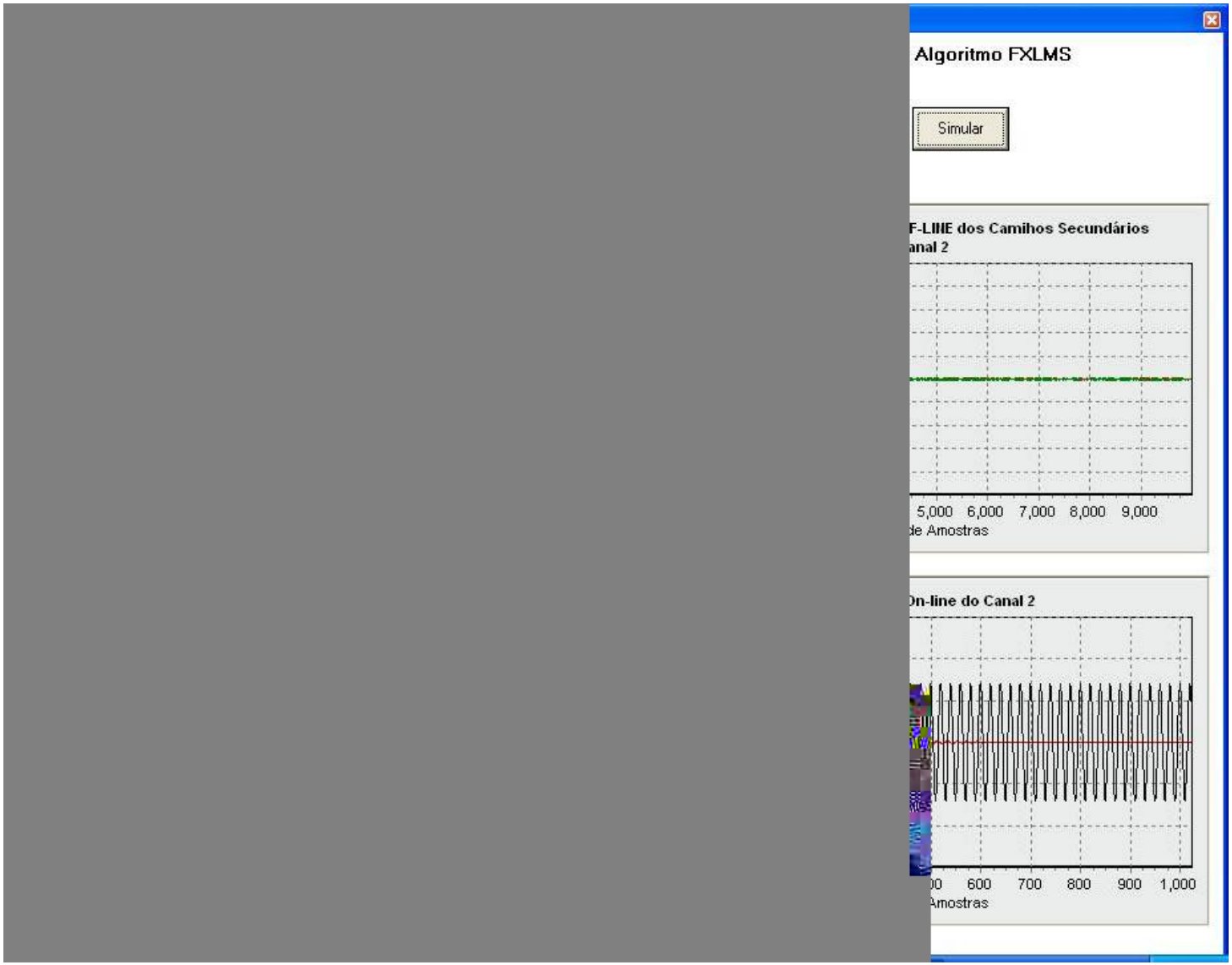


Fig. III.16 – Simulação CAR 7

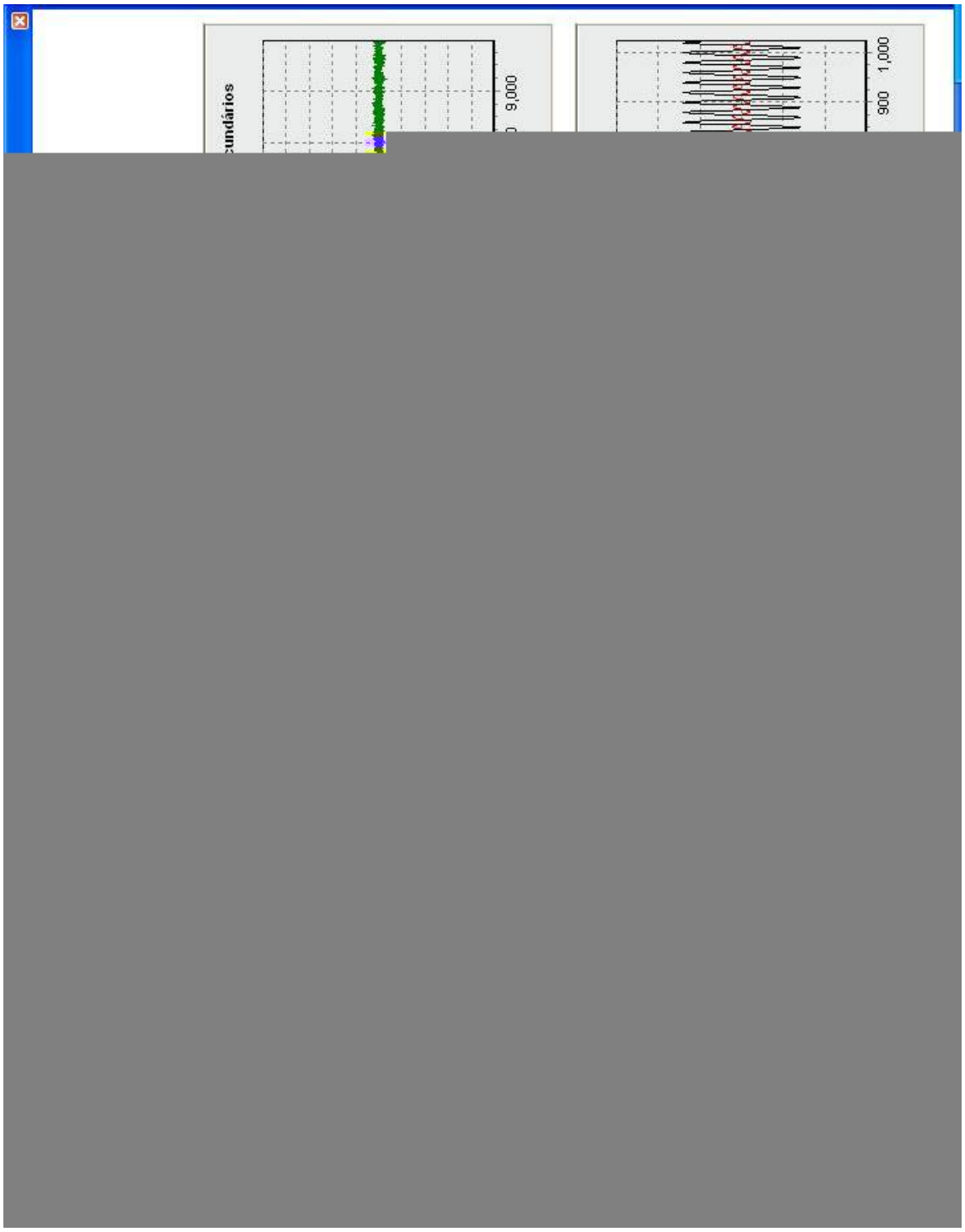


Fig. III.17 – Simulação CAR 8

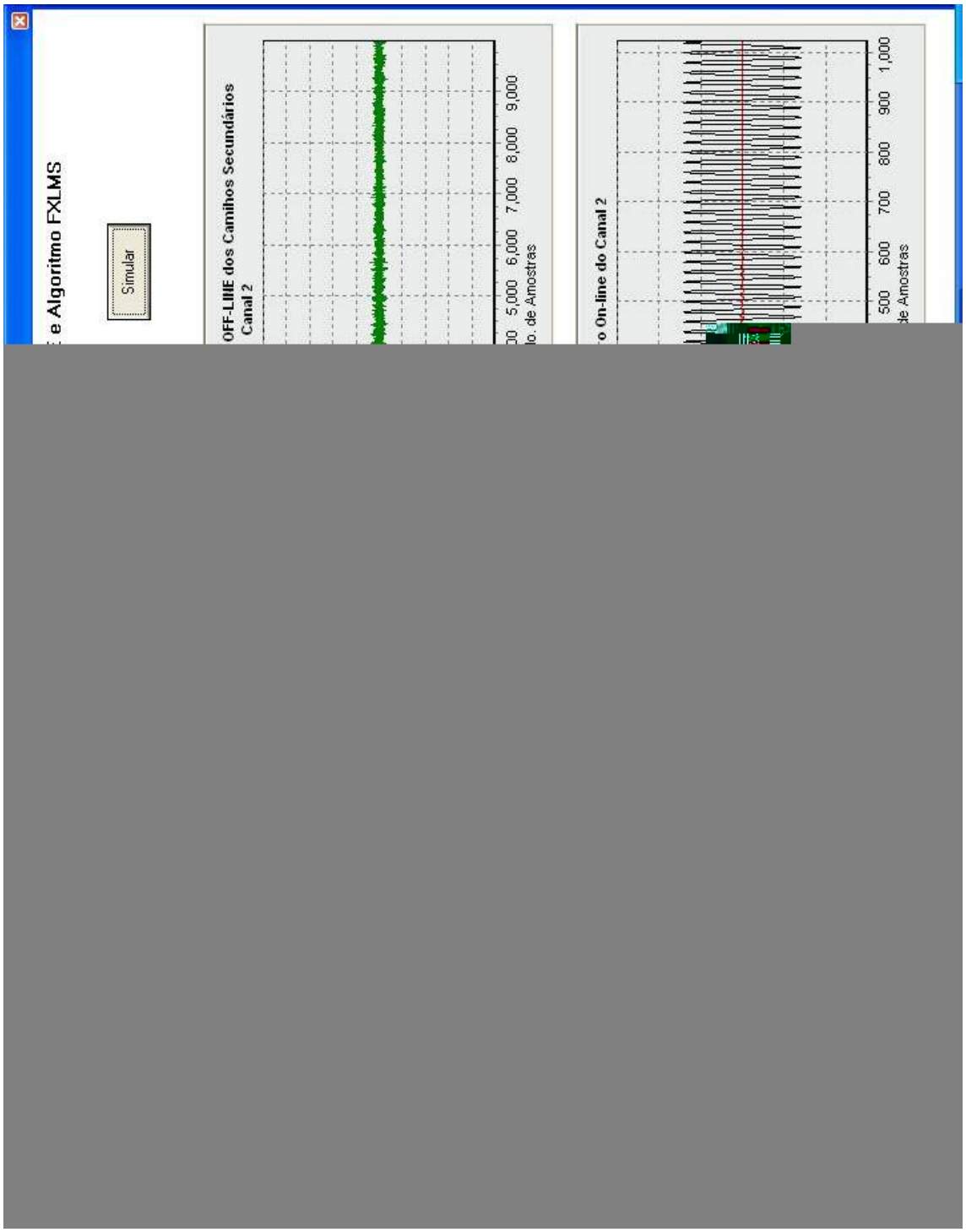


Fig. III.18 – Simulação CAR 9

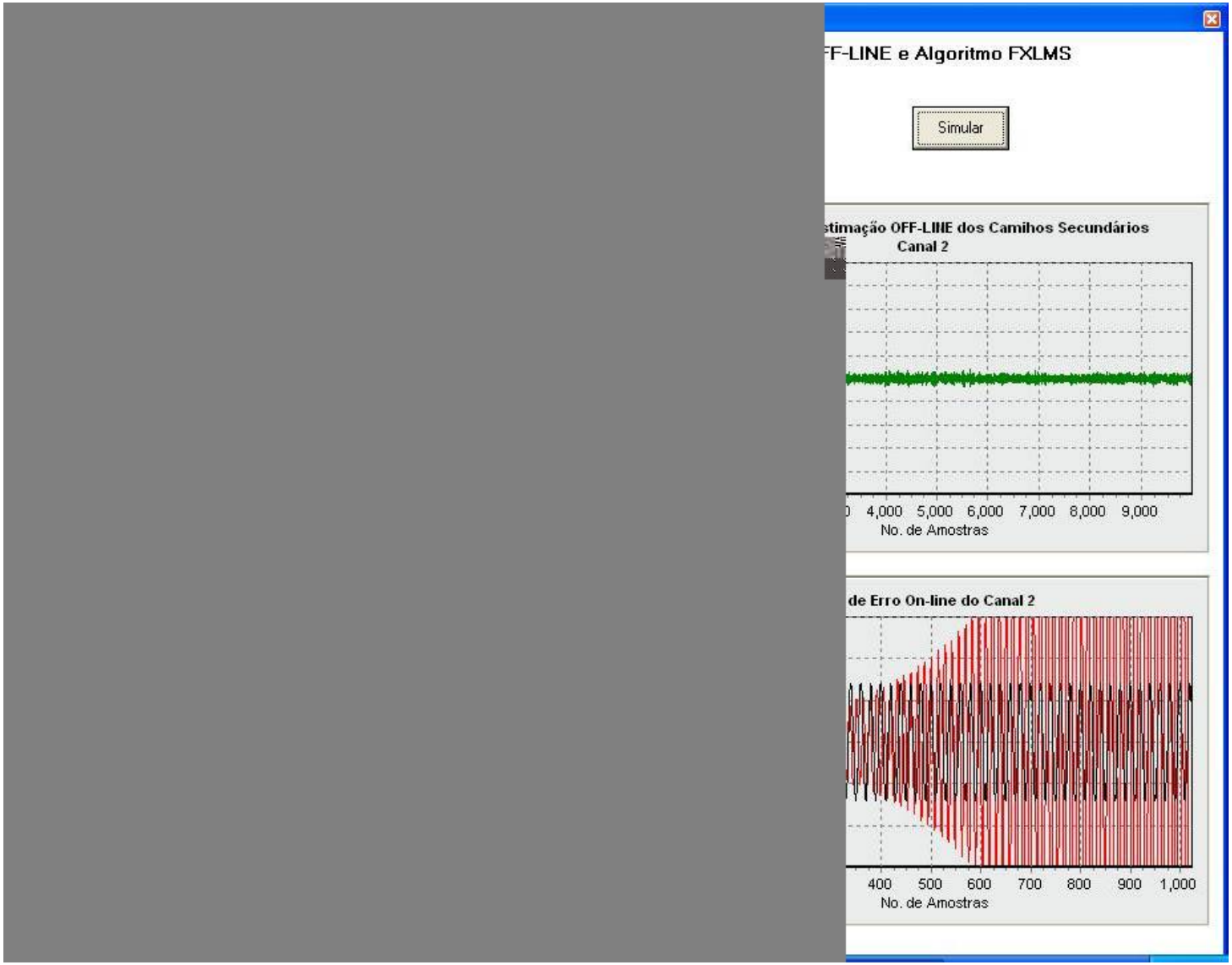


Fig. III.19 – Simulação CAR 10

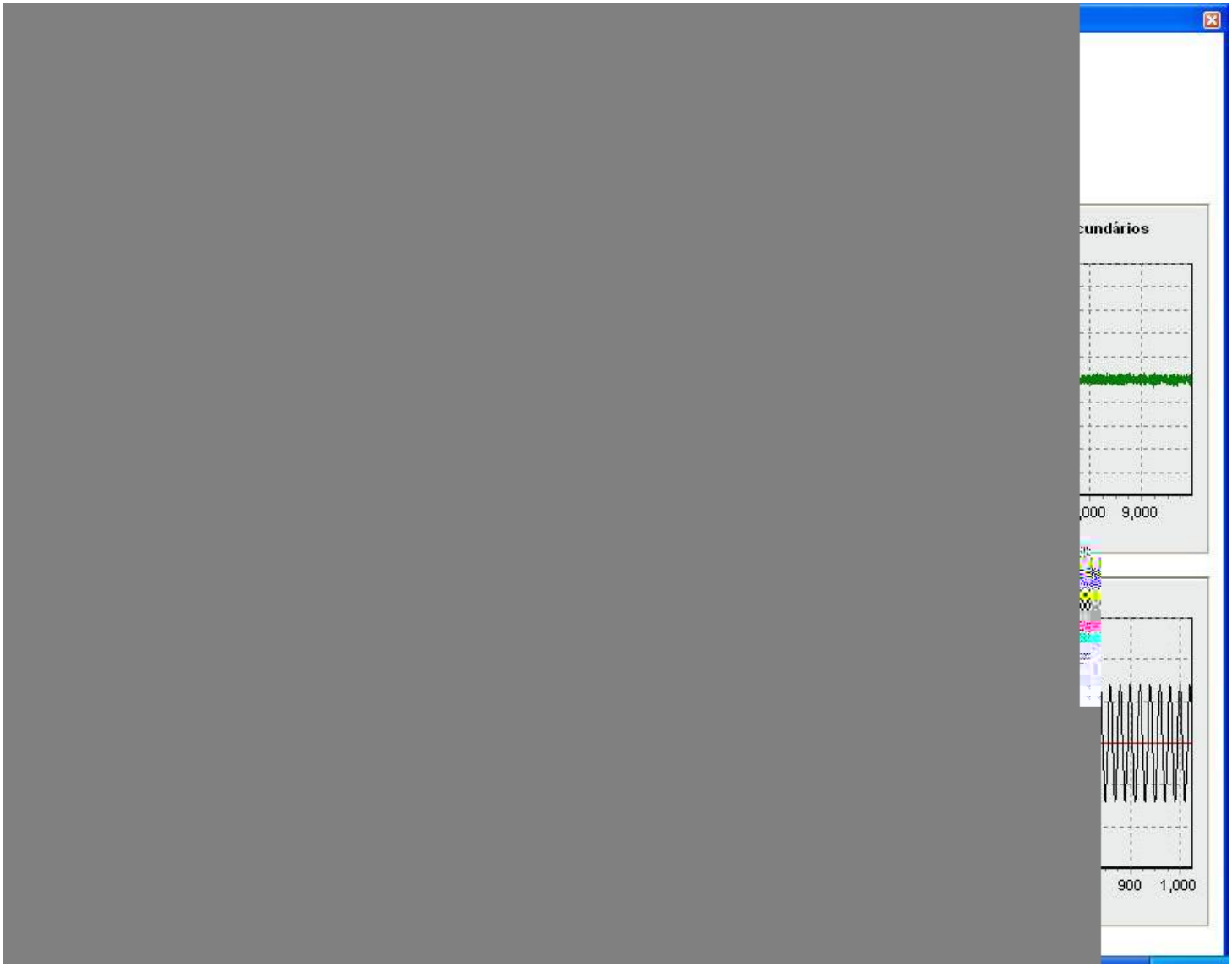


Fig. III.20 – Simulação CAR 11

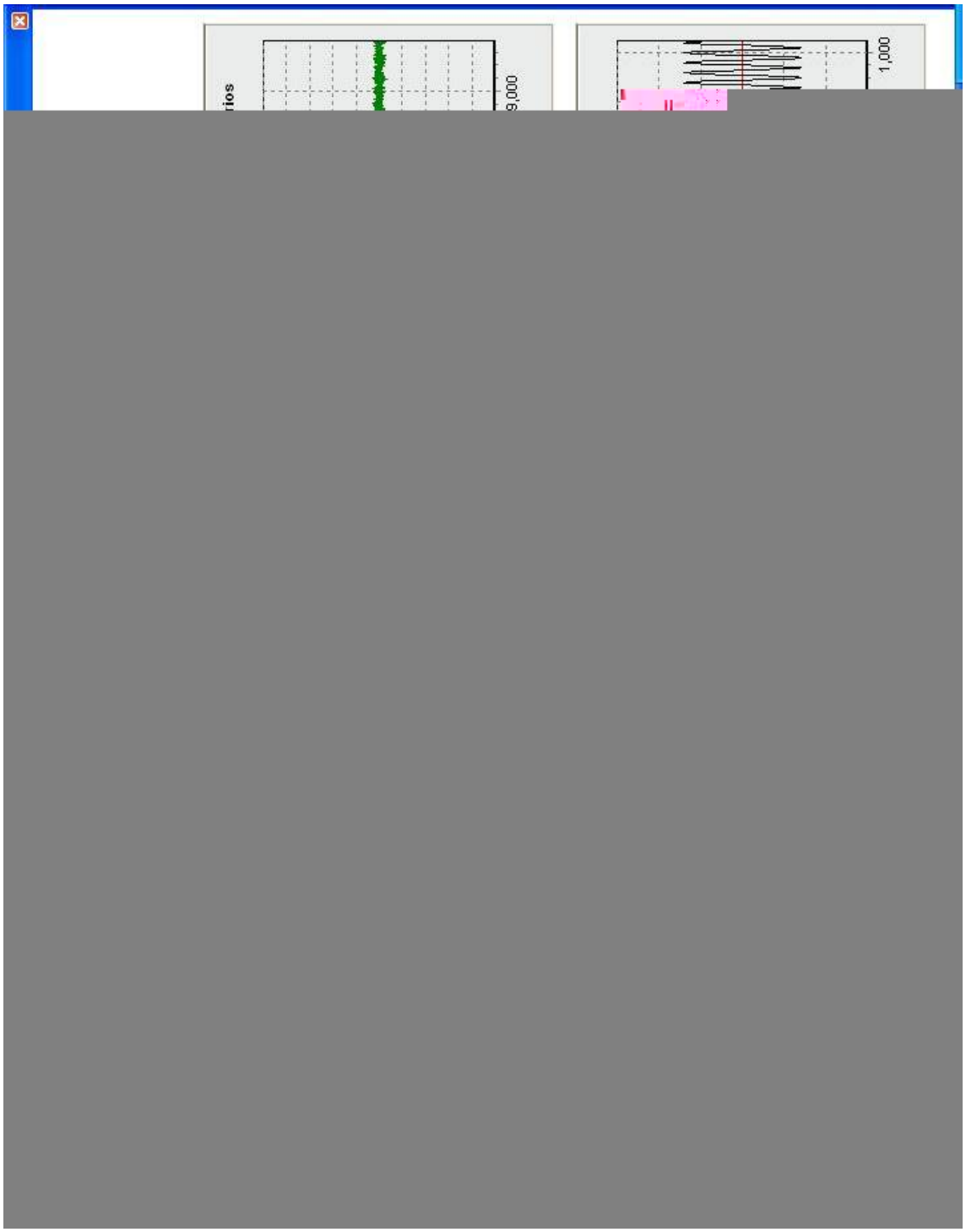


Fig. III.21 – Simulação CAR 12

Fig. III.22 – Simulação CAR 13

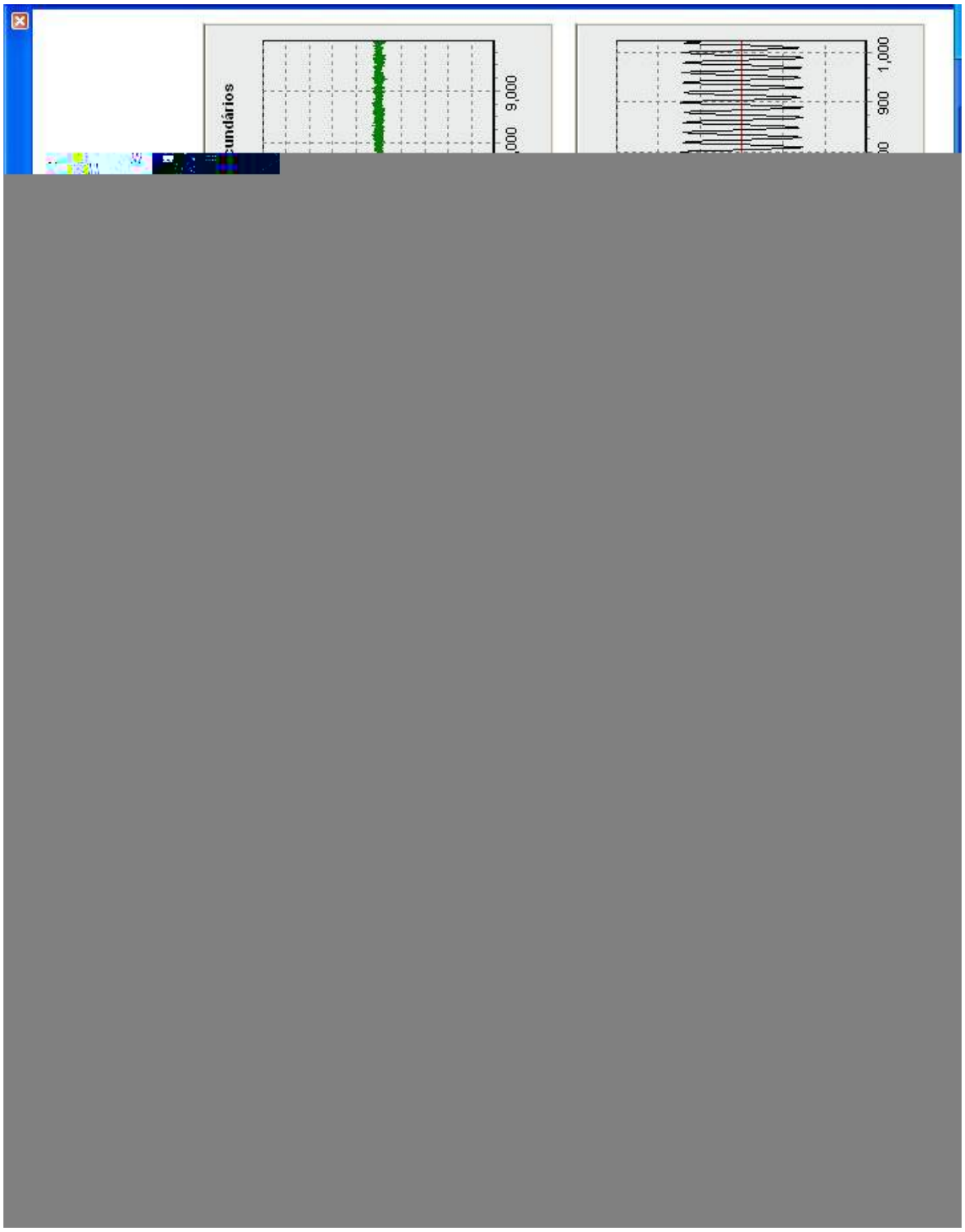


Fig. III.23 – Simulação CAR 14

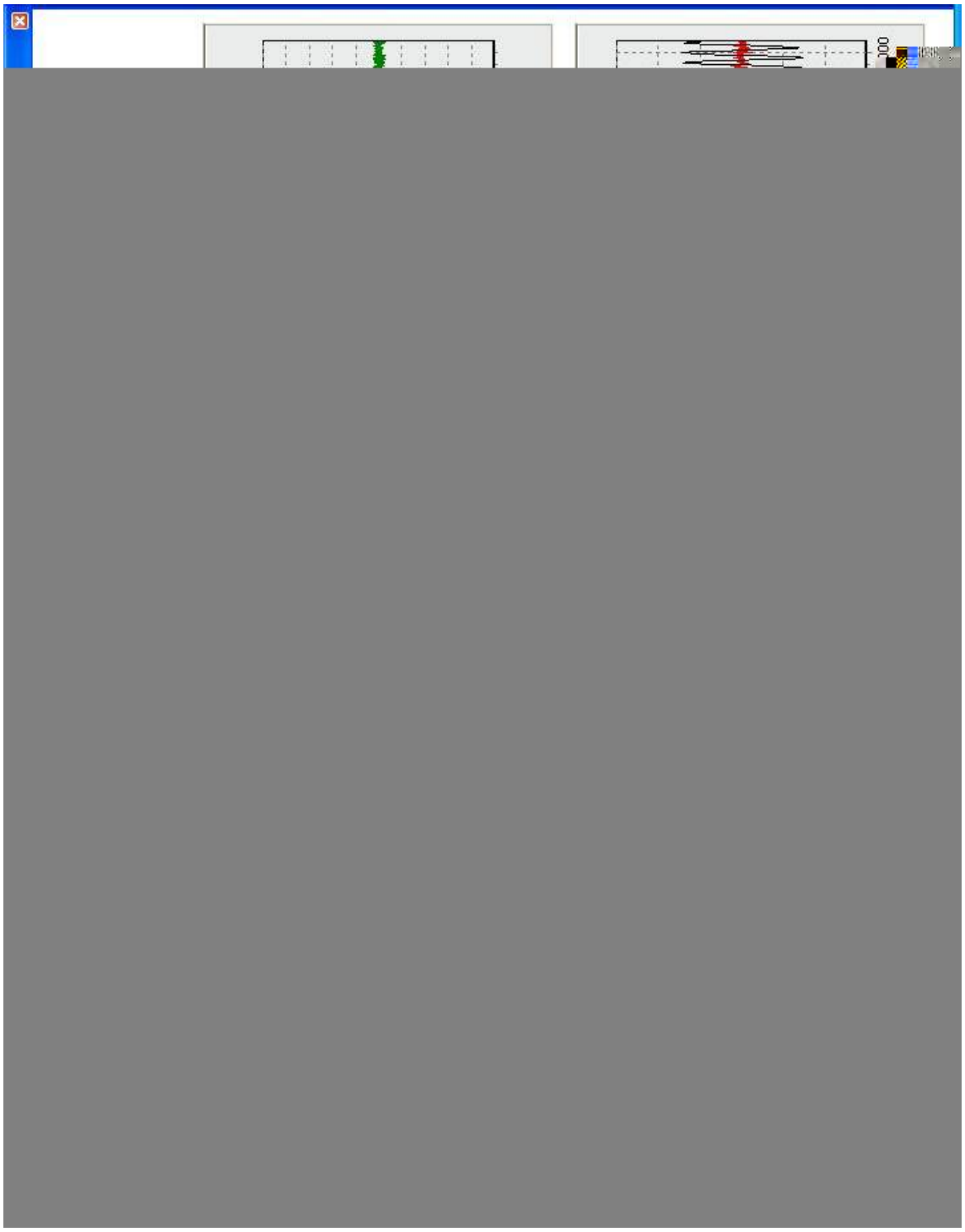


Fig. III.24 – Simulação CAR 15

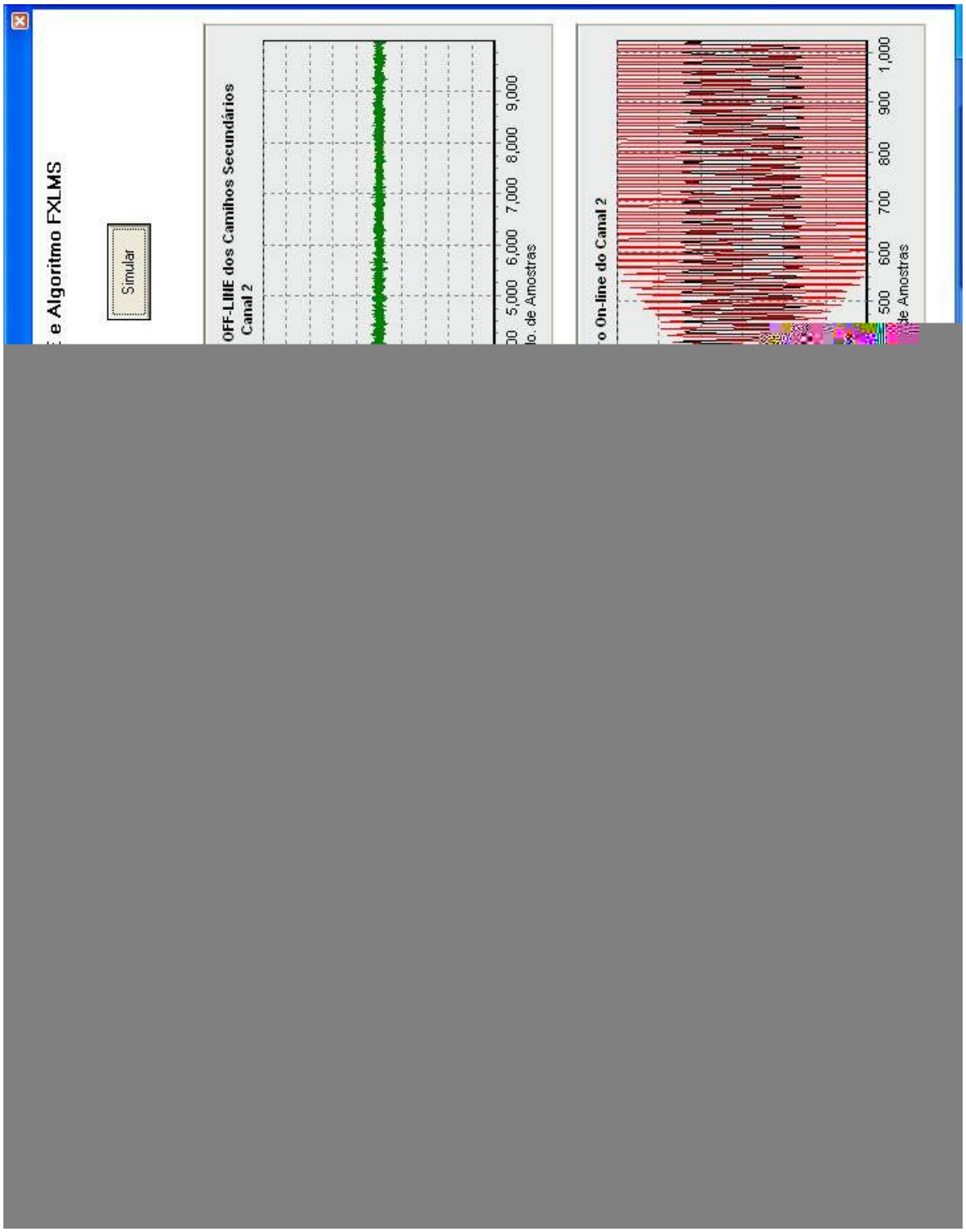


Fig. III.25 – Simulação CAR 16

Dos resultados das simulações apresentados nas figuras III.16 a III.25 concluí-se que:

Para sinais puramente senoidais (figuras III.16 a III.21):

- Pequena sensibilidade ao parâmetro μ. Valores da ordem de dez vezes não apresentaram variações no tempo de acomodação do sinal de erro;
- Sensibilidade à variação da ordem do filtro. Variações da ordem de quatro vezes determinam sensíveis diferenças no tempo de acomodação do sinal de erro, levando à instabilidade para ordem dos filtros muito grandes.

Para sinais com SNR=20dB (figuras III.22 a III.25):

- Grande sensibilidade ao parâmetro μ. Pequenas variações neste parâmetro levam o sistema a gerar problemas de overflow;
- Maior sensibilidade à variação da ordem do filtro. O sistema pode apresentar problemas de overflow e instabilidade.

Foram realizadas outras simulações com valores diferentes da relação sinal-ruído. Tais simulações não foram apresentadas no trabalho já que seu comportamento é semelhante ao da relação do sinal puramente senoidal e do sinal com SNR=20dB.

Os resultados obtidos estão de acordo com a teoria anteriormente apresentada. O valor do parâmetro μ é determinante para a velocidade e estabilidade do sistema. Valores fora da faixa especificada para os algoritmos em questão, LMS ou FXLMS, geram problemas de overflow, quando valores gerados ficam fora dos limites do tipo *float,* e de problemas de estabilidade quando o algoritmo tende a convergir e depois, rapidamente, diverge para valores de coeficientes não apropriados. A variação para valores maiores na ordem do filtro, de modo geral, levou a uma melhoria da qualidade do sinal de erro. No entanto, dependendo do algoritmo utilizado, valores muito altos ou não influenciaram na qualidade do sinal de erro no estado estacionário ou levaram o sistema a apresentar problemas de instabilidade.

Os resultados obtidos determinam que um sistema CARA do tipo proposto deva ser amplamente simulado para que sua implementação prática obtenha resultados satisfatórios dentro da faixa de freqüência desejada. Fatores como simulação da realimentação acústica e estimação dos caminhos secundários na forma on-line, devem melhorar a qualidade do sistema[22].

CAPÍTULO IV - Implementação em Sistema de Desenvolvimento DSP TMS320

Neste capítulo será descrito um modelo para implementação de um sistema CARA na plataforma de desenvolvimento TMS320C6713 da Texas Instruments (DSK6713). Serão mostradas as características básicas de hardware deste sistema DSP bem como a geração de código na linguagem C++ e sua utilização para controle do modelo.

IV.1 – O Sistema CARA Simplificado - Duto

O sistema proposto foi modelado por meio de simulação numérica e os resultados se apresentaram dentro dos valores esperados baseados nos cálculos teóricos abordados nos capítulos anteriores. No entanto, sua implementação física se torna inviável em função das dimensões e da falta de recursos existentes no momento da execução deste trabalho.

Para execução da parte prática deste trabalho foi então escolhido um sistema de cancelamento de ruído em duto. Este sistema é suficiente para comprovar os modelos de cancelamento e é baseado nele que o desenvolvimento da teoria do cancelamento ativo de ruído acústico está exemplificada [2][3][4][16][20].

IV.1.1 – Disposição Física do Sistema

O sistema será composto na parte acústica por um duto de material rígido, PVC, com as seguintes dimensões: comprimento L=1,55m e diâmetro d=0.11m. O dispositivo é mostrado nas figura IV.1 e IV.2.

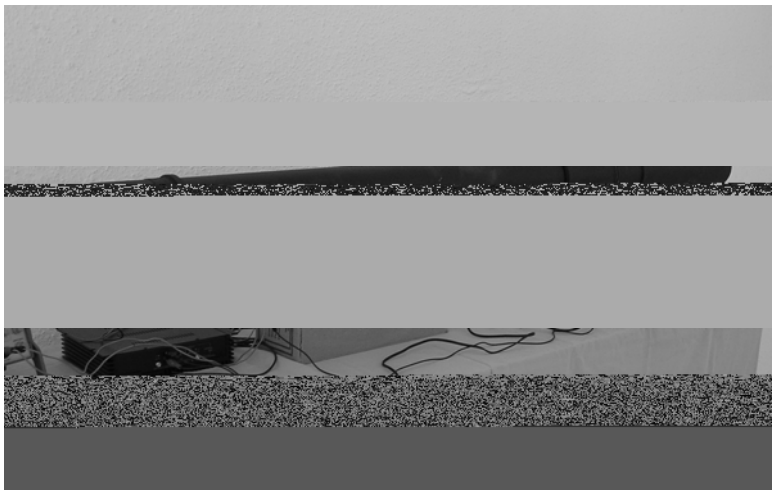


Fig. IV.1 – Sistema Duto

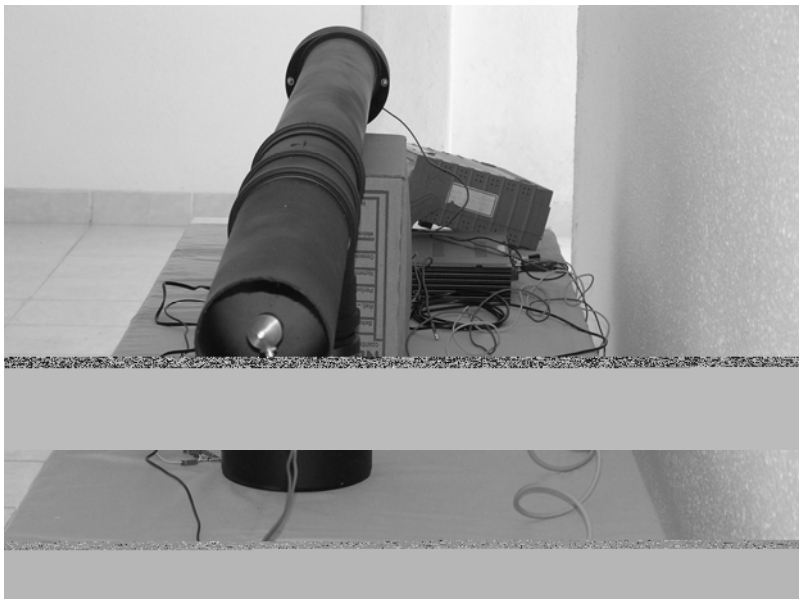


Fig. IV.2 – Detalhe da posição do microfone de erro

Serão utilizados como sensores dois microfones de eletreto, um para sensor de referência e outro para sensor de erro, com as seguintes características (do fabricante):

Modelo: EPE DG09767
Tipo: Eletreto
Sensibilidade: 56 ± 2dB (0dB=1V/ubar, 1 KkHz)

Os sensores e os atuadores podem ser vistos na figura IV.3.

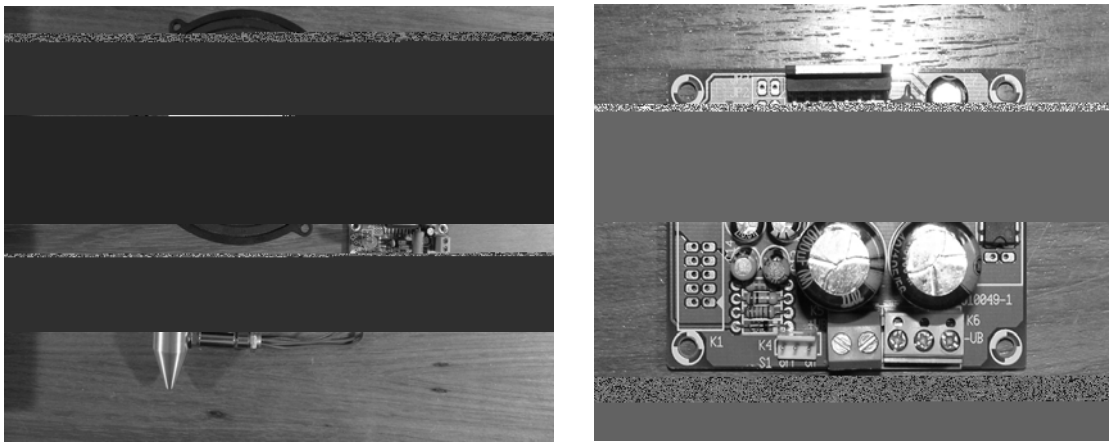


Fig. IV.3 – Alto-falante, microfone eletreto e amplificador de potência (em destaque)

Na parte eletrônica serão empregados os seguintes dispositivos:

- Dois pré-amplificadores (P1 e P2) para compatibilizar o sinal dos microfones de eletreto (MR e ME) com a entrada da placa de desenvolvimento DSK6713;
- Dois amplificadores de potência (A1 e A2) para amplificar o sinal de saída do DSK6713 e acionar os alto-falantes gerador (AFR) e de cancelamento (AFC);
- Gerador de sinal;
- Osciloscópio digital;
- Sistema DSK6713;
- Microcomputador PC;
- Material complementar: cabos, fontes, etc.

Os circuitos dos amplificadores e pré-amplificadores se encontram no Apêndice B. O esquema da placa de desenvolvimento está disponível no endereço do fabricante na Internet, http://www.ti.com. O sistema de desenvolvimento é mostrado na figura IV.4.

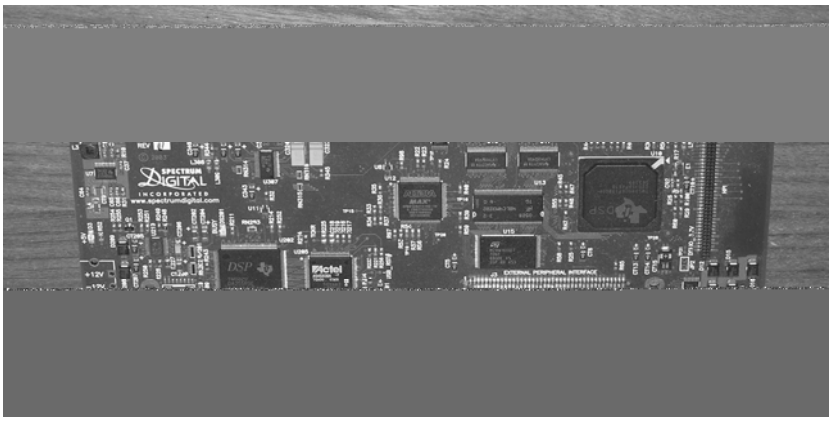


Fig. IV.4 – Sistema de desenvolvimento TMS320C6713 DSK

Testes realizados comprovam que tanto os microfones quanto os alto-falantes se comportam de forma linear para a faixa de freqüências de interesse para o modelo. O dispositivo de teste é mostrado na figura IV.5.

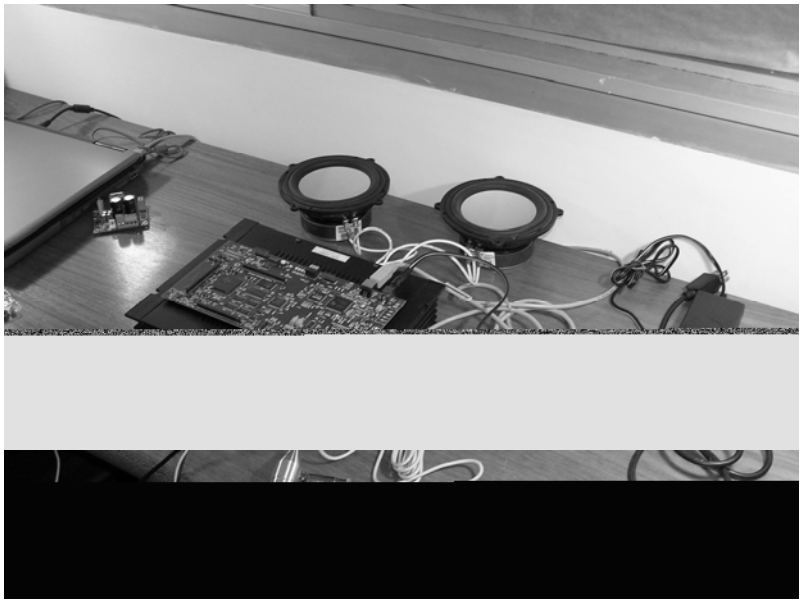


Fig. IV.5 – Testes preliminares dos sensores e atuadores

A figura IV.6 mostra, esquematicamente, a configuração do sistema a ser empregado.

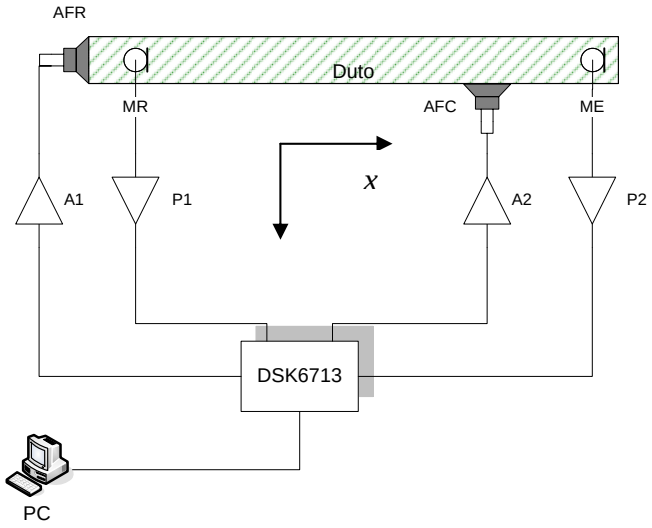


Fig. IV.6 – Dispositivo para simulação em tempo real

IV.2 – Introdução ao Sistema DSK6713

O sistema de desenvolvimento DSK6713 é um conjunto de ferramentas de hardware e software. Para o desenvolvimento do trabalho serão abordadas as características do processador e da placa. O sistema DSK6713 alcança alto desempenho pela combinação de altas taxas de clock e arquitetura específica para aplicação em DSP. Outras características

importantes são: cache de instruções, conjunto de instruções dedicado, barramento de dados múltiplo para acesso a várias palavras de dados em um único ciclo de clock além de baixo consumo. O processador DSP TMS320C6713 possui características específicas para a implementação de filtros e algoritmos:

- Unidades MAC (multiplicar e acumular) rápidas: a maioria das operações DSP se resume a multiplicação e adições para realizar funções de filtros, FFT, algoritmos, etc. As operações MAC são realizadas em um ciclo de instrução pelo processador;
- Acesso múltiplo à memória: o acesso a dados e instruções é realizado simultaneamente em função de múltiplos barramentos internos e bancos de memória independentes;
- Modos de endereçamento especiais: possui unidade de geração de endereços de memória em paralelo com a execução das instruções;
- Funções especiais de controle: o programador trata loops sem gastar ciclos de clock adicionais para testar contadores e desvios ao início do loop;
- Conjunto de instruções otimizado: funções dedicadas à implementação de algoritmos para DSP;
- Interface com periféricos otimizada: interface com A/D , D/A e I/O de alta velocidade.

Esse processador atinge um alto nível de paralelismo, executando instruções múltiplas simples em paralelo com altas taxas de clock. O 6713 utiliza a arquitetura VLIW (“Very Long Instruction Word”), instruções com palavras longas, permitindo que as oito unidades internas executem quatro ou oito instruções por ciclo de clock. Tal capacidade de processamento possibilita sua utilização em sistemas de radar, processamento de áudio e imagem, comunicação e aparelhagem médica.

Neste sistema de desenvolvimento está presente um codificador/decodificador (codec) de 32 bits e dois canais. Este codec será responsável pelas conversões A/D e D/A necessárias ao tratamento dos sinais analógicos do sistema CARA. O codec está baseado no dispositivo AIC23, trabalhando na freqüência de 12MHz e com freqüência de amostragem dos sinais de entrada que pode variar de 8kHz a 96kHz.

O processador TMS320C6713 trabalha a uma taxa de 225MHz, e a placa inclui 16MB de memória SDRAM, 256KB de memória flash, entrada para microfone, duas entradas analógicas, duas saídas analógicas e uma saída para fones de ouvido. Para facilidade operação e testes, a placa disponibiliza quatro chaves (“dip switches”) e quatro leds. O clock de 225MHz permite ao microcontrolador processar até 8 instruções de 32 bits em 4.44ns (1/225MHz). É um processador de ponto flutuante composto por 6 unidades lógicas e aritméticas (ALU), barramento de 32 bits e dois conjuntos de registradores de 32 bits para uso geral.

IV.2.1 – O Codec AIC23

Vale abordar uma descrição mais detalhada do codec AIC23 por ser responsável por todas as funções de entrada e saída (I/O) de sinais do sistema. O valor máximo do sinal de entrada deve estar na faixa de 6Vpp. As funções de filtragem e reconstituição dos sinais mostrados na figura I.7 são realizadas pelo próprio processador. Esse codec possui a capacidade de transferir palavras de 16, 20, 24 e 32 bits. Os conversores internos são baseados no modelo Sigma-Delta que atingem alta resolução com pequenas taxas de amostragem e estão na categoria na qual a taxa de amostragem pode ser bem maior do que a taxa de Nyquist[21]. A taxa de amostragem pode ser controlada por software, atribuindo valores a constante *k* da seguinte forma:

$$f_s = \frac{12MHz}{k} \Rightarrow k = \frac{12MHz}{f_s} \tag{4.00}$$

Para uma freqüência de amostragem igual a 8kHz, *k*=1500 e para 44.1 kHz, *k*=272. Esses valores são aplicados ao registrador *SAMPLERATE* do codec.

O sinal de saída é fornecido no formato complemento de dois que passa por um filtro passa-baixas para produzir o sinal de saída analógico. A figura IV.7 mostra o diagrama de blocos do codec AIC23.

(Em branco para melhor visualização da figura)

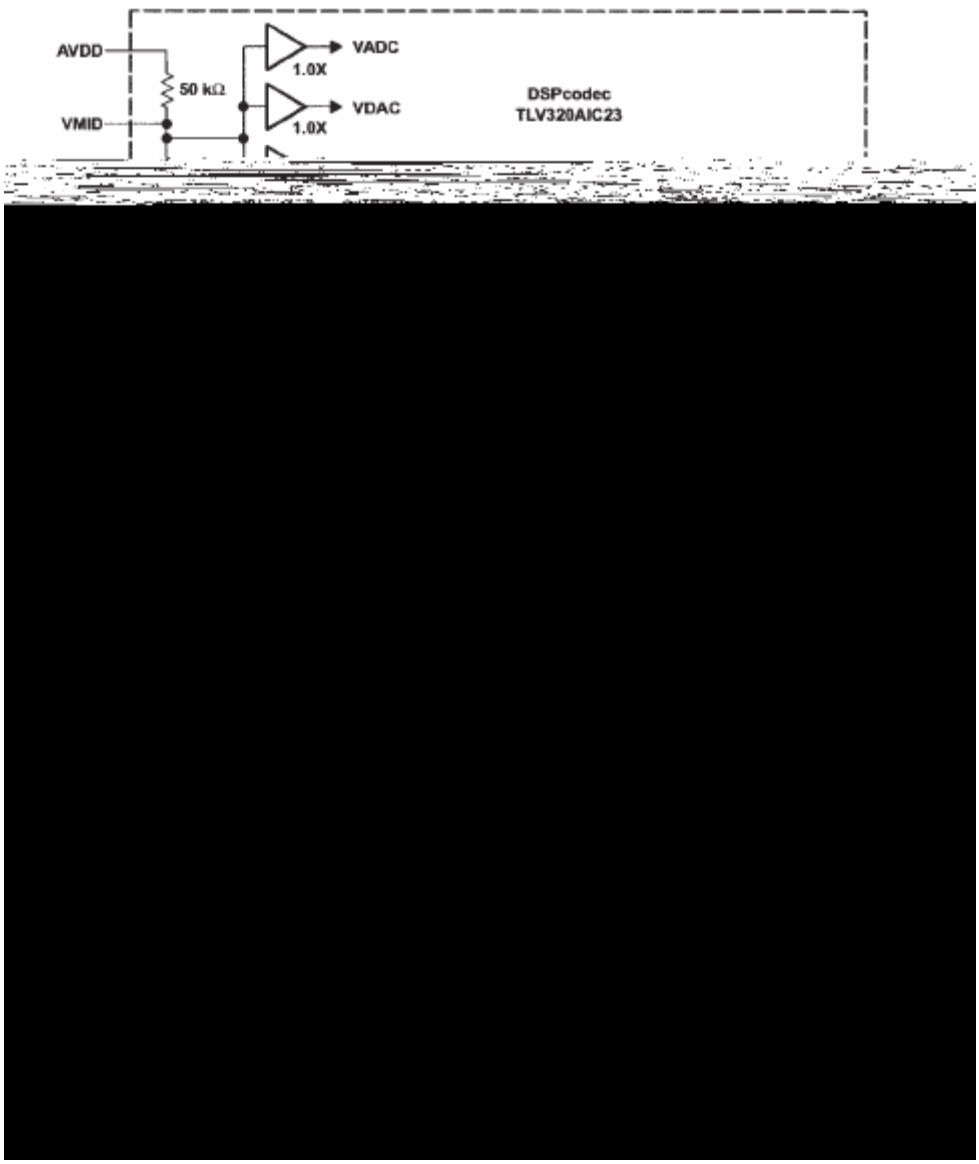


Fig. IV.7 – Diagrama de blocos do AIC23 (Texas Instruments)

IV.2.1.1 – Considerações Sobre as Linguagens de Programação

Basicamente são dois os tipos de linguagem de programação utilizados para programar o DSK6713 no desenvolvimento do sistema CARA. O primeiro é a linguagem de programação de “alto nível” C++, por meio de programas que acompanham o sistema ou por meio do MATLAB que possui bibliotecas que geram código compatível com o dispositivo. Quando questões de tempo de processamento não puderem ser resolvidas com código C++, será então utilizado o segundo tipo, a Linguagem Assembly.

Como o sistema CARA utiliza filtros digitais, a maioria das operações se resumem à adições e multiplicações, que podem ser implementadas da seguinte forma:

a. Adição

```
ADD   .L1     A3,A7,A7      ;A3+A7⇒A7   (soma e acumula)
```

L1 é opcional e indica qual unidade aritmética está sendo utilizada.

b. Multiplicação

```
 MPY   .M1   A7,B7,B6      ;LSB(A7)*LSB(B7) ⇒B6
||MPYH .M2   A7,B7,A6      ;MSB(A7)*MSB(B7) ⇒A6
```

O símbolo “||” indica que a operação está sendo executada em paralelo com a instrução anterior. Neste caso, duas operações MAC estão sendo executadas no mesmo ciclo de instrução.

IV.2.2 – Consideração Sobre Sistemas em Tempo Real

O emprego de processadores DSP em sistemas em tempo real é limitado pela freqüência de amostragem, f_s. A velocidade do processador determina a taxa máxima que um sinal analógico pode ser amostrado. Por exemplo, em um sistema de processamento do tipo amostra por amostra, uma amostra de saída é computada quando uma amostra é apresentada ao sistema. Então, o atraso entre a saída e a entrada é, no mínimo, de um intervalo de amostra igual a T segundos. Um sistema DSP para aplicação em tempo real demanda que o tempo de processamento t_p seja menor que o período de amostragem T para que o processamento seja completado antes da chegada da próxima amostra. Além disso, ainda devem ser levados em consideração os atrasos implícitos às operações de I/O, t_{io}.

$$t_p + t_{io} < T \tag{4.01}$$

Essa é uma limitação para o processamento em tempo real, e limita a maior freqüência do sinal a ser trabalhado pelo DSP.

A limitação de banda do sistema , t_{io} , é dada por (Kuo et.al, 2006)

$$f_M \leq f_s/2 < \frac{1}{2(t_p + t_{io})} \tag{4.02}$$

Fica claro que, quanto maior o tempo de processamento t_p , menor será a banda de trabalho do sistema. No caso de t_{io} , o problema pode ser diminuído usando, ao invés do processamento amostra por amostra, o processamento por blocos. As amostras são colocadas em banco de memória e gerenciadas por um controlador de acesso direto à memória (DMA). Esse controlador interrompe o processamento quando o bloco está cheio e esse bloco será, então, processado.

IV.2.3 – Desenvolvendo Algoritmos no Sistema DSP

Os algoritmos a serem implementados no trabalho serão desenvolvidos nas linguagens de alto nível MATLAB©, SIMULINK© e C++. Por meio destas ferramentas será simulada sua operação e no estágio seguinte irão rodar no sistema DSK6713 de forma autônoma. Os sinais de ruído para teste podem ser gerados internamente, lidos de arquivos previamente gravados ou por meio de equipamento eletrônico externo. Os resultados serão gravados em forma de arquivo ou analisados na forma de gráficos e de medidas por equipamentos eletrônicos.

Essa linha de ação pretende maximizar o tempo de desenvolvimento e diminuir o tempo de teste de falhas do sistema (debug):

- ✓ A utilização de linguagens de alto nível diminui o tempo de desenvolvimento dos algoritmos e mantém um alto grau de compatibilidade entre as diversas ferramentas utilizadas;
- ✓ Facilidade de edição e debug utilizando softwares integrados;
- ✓ As operações de I/O baseadas em gravação e leitura de arquivos são mais fáceis de implementar e de serem recuperadas para futura análise do sistema.

A figura IV.8 mostra, simplificadamente, o processo.

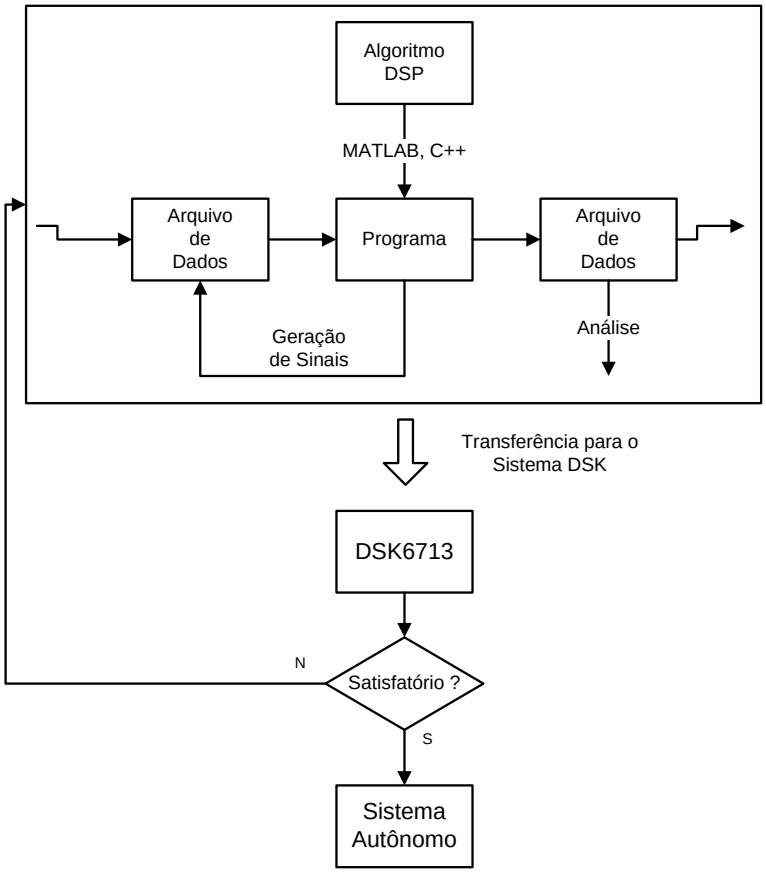


Fig. IV.8 – Procedimento para implantação dos algoritmos no sistema DSK6713

IV.3 – Características do Sistema Duto

O sistema físico mostrado na figura IV.1 de comprimento L, possui um alto-falante para gerar o ruído em $x = 0$ e é aberto na outra extremidade em $x = L$. O raio da seção circular do duto é $a = (d/2)$. O microfone de referência MR encontra-se em $x = x_R$, o alto-falante de cancelamento em $x = x_c$ e o microfone de erro ME em $x = L$.

O duto pode ser considerado como um guia de onda que força a propagação da onda acústica paralela a sua dimensão mais longa, no caso a direção x. Se o duto for excitado por uma perturbação, a pressão sonora gerada por AFR, e se o comprimento de onda for, no mínimo, duas vezes maior que a seção transversal do duto, apenas ondas planas se propagarão pelo duto[20]. Para o duto circular, a maior freqüência para a qual apenas ondas planas vão se propagar é dado por

$$f = \frac{1}{2\pi}\frac{1.84c}{a} \tag{4.03}$$

onde c é velocidade do som e a é o raio da seção circular do duto. Para o caso em estudo,

onde Q_o e $p(x)$ são valores rms. Substituindo em (4.06) chega-se a

$$\frac{\partial^2 p(x)}{\partial x^2} + \frac{\omega}{c^2} p(x) = \frac{\rho\omega}{V} Q_o(x) \tag{4.09}$$

O valor de pressão rms em estado estacionário em um ponto x é dado por [20]

$$p(x) = \sum_m p_m(x) = \sum_m \left| p_m(x) \right| e^{j\Theta_m} \quad N/m^2 \tag{4.10}$$

Cada termo m é a pressão acústica em ponto x com magnitude $\left|p(x)\right|$ e ângulo de fase Θ_m.

A montagem física do dispositivo de teste pode limitar o desempenho do sistema CARA em função de alguns aspectos: disposição dos sensores e atuadores, tamanho do filtro, estabilidade e causalidade.

Ambos os microfones de referência e de erro estão localizados no interior de um duto onde ocorre fluxo (onda acústica) e perceberão, além do ruído gerado, os ruídos gerados pelo próprio fluxo de ar. Então, fluxo de ruído e turbulência nos microfones podem degradar o limite de cancelamento. Para evitar tal problema a solução indicada por [4] é mostrada na figura IV.9:

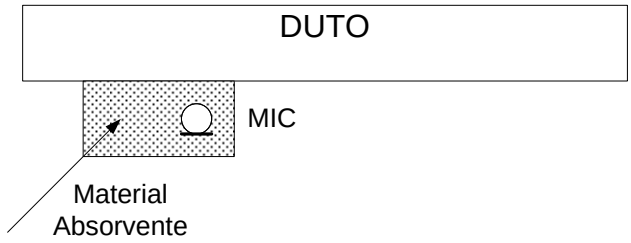


Fig. IV.9 – Solução para diminuir o efeito da turbulência no sensor

O microfone de referência fica embutido em um pequeno compartimento ao longo do tubo preenchido com material absorvente acústico. No dispositivo usado para teste, tal solução não é necessária já que as dimensões do microfone de eletreto e seu suporte, 8mm, são desprezíveis para as pressões sonoras envolvidas. Para o microfone de erro a melhor posição é na saída do duto evitando assim o efeito da turbulência[22].

IV.4 – Implementação do Software

O código será implementado em C++ e compilado no ambiente de desenvolvimento que acompanha o DSK6713. Basicamente devem-se seguir os seguintes passos:

- Inicializar o sistema de desenvolvimento acertando os parâmetros de compilação e link;

- Escre ódigo n
- Comp ransfi ri
- Execu apura o

Apesar de par um pr oc
destinado à i ment ç
intermediárias a im le
trabalho, por c derá- os
sistema.

IV.4.1 – Geração e sina s

Na primeii abor la
DSK6713. No ent nto, p ar
gerado por um ge ador (x
secundário, será gerad o
configuração da figura IV 1(

IV.4.2 – Inicialização e Funções Básicas de Entrada e Saída (I/O) do DSK6713

O programa que a ser executado no DSK6713 poderá seguir duas estratégias para aquisição de sinais: por interrupção ou por pooling, utilizando-se respectivamente as funções *comm_pool ()* e *comm_intr()*.

► Quando utilizado o método de interrupção, a interrupção INT11 é habilitada.

► Quando utilizado o método de interrupção, na configuração do sistema, deverá se fazer referência ao arquivo *vectors_intr.asm* e para o modo pooling ao arquivo *vectors_pool.asm.*

Escolhido o método de aquisição a leitura, a função utilizada para leitura de dados é a *input_sample()* e para saída de dados *output_sample()*.

► Se utilizado no modo mono, o canal esquerdo é escolhido por default.

O código listado abaixo na figura IV.11, mostra o "esqueleto" básico de um programa, escrito na Linguagem C, para aquisição de dados utilizando-se o DSK6713, no modo de interrupção:

```
01: #include "dsk6713_aic23.h" //  inicializa o CODEC AIC23
02: Uint32 fs=DSK6713_AIC23_FREQ_8KHZ; // taxa de amostragem
03: interrupt void c_int11() // rotina de interrupção (INT11 por default)
04: {
05: short sample_data;
06: sample_data = input_sample(); // Lê dado na entrada
07: output_sample(sample_data); // Envia dado para saída
08: return;
09: }
10: void main() // programa principal
11: {
12: comm_intr(); // inicializa DSK6713
13: while(1); // loop infinito
14: }
```

Fig. IV.11 – Código básico para leitura e escrita de dados no DSK6713(interrupção)

O código realiza a função de inicialização do sistema de desenvolvimento (01), inicialização do AIC23 com taxa de amostragem de 8KHz, cria rotina para tratamento da INT11 (03-09) e cria a rotina principal *main()* (10-14). A cada período de amostragem $T_s = 1/F_s = 1/8KHz = 0.125ms$ ocorrerá uma interrupção. Será então chamada a rotina *c_int11()* para tratar esta interrupção. No exemplo da figura IV.6 um dado é lido da entrada com conversor A/D e enviado à saída pelo conversor D/A. O programa permanece em loop contínuo em função do teste *while(1)*.

► A amplitude máxima do sinal de entrada deve ser de 6Vpp.

► No arquivo de inicialização c6713dskinit.c pode-se mudar o ganho dos canais direito e esquerdo, alterando-se o valor dos registros 1 e 0 respectivamente.

Para o modo pooling o código é mostrado na figura IV.12:

```
01: #include "dsk6713_aic23.h" //  inicializa o CODEC AIC23
02: Uint32 fs=DSK6713_AIC23_FREQ_8KHZ; // taxa de amostragem
03: void main() // programa principal
04: {
05:       short sample_data;
06:       comm_pool(); // inicializa no modo pooling
07        while(1); // loop infinito
08:       {
09:                 sample_data = input_sample(); // Lê dado na entrada
10:                 output_sample(sample_data); // Envia dado para saída
11:       }
12: }
```

Fig. IV.12 – Código básico para leitura e escrita de dados no DSK6713 (pooling)

Neste modo a entrada fica sendo constantemente lida por uma rotina que indica quando o dado está pronto para ser lido ou transmitido. É mais simples, mas menos eficiente que o método de interrupção. Deve-se ressaltar que esses são modelos básicos para utilizar o I/O do DSK6713, onde a entrada é copiada para a saída sem qualquer tipo de tratamento. As entradas e saídas são indicadas como LINE-IN e LINE-OUT na placa.

► Para utilizar a entrada de microfone deve-se alterar o valor do registro 5 de 0x0011 para 0x0015 do arquivo c6713dskinit.c.

IV.4.3 – Gerando Sinais no DSK6713

Para o projeto devem ser gerados dois tipos de sinais: senoidal e ruído branco. A geração de sinais senoidais se dá pelo método de geração de tabelas e é dependente da freqüência de amostragem utilizada. Gera-se uma freqüência f , calculando-se o número de pontos np da tabela da seguinte forma:

$$np = F_s / f \tag{4.11}$$

Por exemplo, para $f = 400Hz$ e $F_s = 8000$ devemos criar uma tabela com 20 pontos.

O código para gerar um sinal senoidal de 400Hz é o apresentado na figura IV.13:

```
#include "DSK6713_AIC23.h" // inicializa o CODEC AIC23
Uint32 fs=DSK6713_AIC23_FREQ_8KHZ; // taxa de amostragem
#include <math.h>
#define tam_tab (short)20 // tamanho da tabela (8000/400)
short sin_tab[tam_tab]; // tabela de senos
int i;
interrupt void c_int11() // rotina de interrupção (INT11 por default)
```

```
{
output_sample(sin_tab[i]); // envia cada valor de seno
if (i < tam_tab - 1) ++i; // incrementa o índice até o tamanho da tabela
else i = 0; //zera índice
return; //
}
void main()// principal
{
float pi=3.14159; // defini pi
for(i = 0; i < tam_tab; i++)// gera tabela de senos
sin_tab[i] = sin(2.0*pi*i/tam_tab); // scaled values
i = 0;
comm_intr(); // inicializa DSK6713
while(1); // loop infinito
}
```

Fig. IV.13 – Código básico para gerar um sinal senoidal no DSK6713

A geração de ruído é conseguida por meio de um registro de deslocamento “shift register” com realimentação, a partir de uma semente previamente estabelecida. Essa técnica gera uma seqüência pseudo-randômica, que será repetida após um intervalo de tempo finito, ou seja, não é randômica todo o tempo. O tamanho *m* do registro determina o comprimento da seqüência: $L=2^m-1$. A geração da seqüência pseudo-randômica esta baseada no modelo LFSR(“Leap Forward Linear Feedback Shift Registers”), figura IV.14, e pode ser encontrada em [4][22]. O código da figura IV.15 adaptado de [4][23] apresenta a implementação do shift register da figura IV.14 no DSK6713.

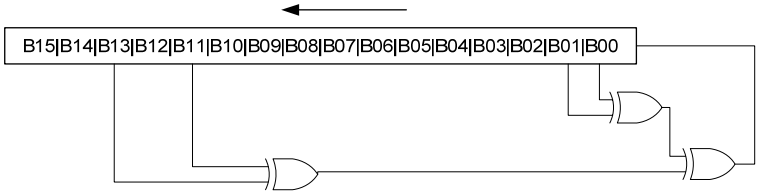


Fig. IV.14 – Geração de seqüência pseudo-randômica

Código:

```
#include “DSK6713_AIC23.h” // inicializa o CODEC AIC23
Uint32 fs=DSK6713_AIC23_FREQ_8KHZ; // taxa de amostragem
typedef struct BITVAL
{
 unsigned int b0:1, b1:1, b2:1, b3:1, b4:1, b5:1, b6:1;
 unsigned int b7:1, b8:1, b9:1, b10:1, b11:1, b12:1,b13:1;
 unsigned int dweebie:2; //bits 14-15
} bitval;

typedef union SHIFT_REG
{
 unsigned int regval;
 bitval bt;
} shift_reg;
```

```
short fb;
shift_reg sreg; //estrutura do shift register
interrupt void c_int11() //rotina de interrupção
{
short prnseq; //sequência pseudo-randômica
if(sreg.bt.b0) //sequence{1,-1}based on bit b0
prnseq = -8000; //escala de saída do valor de ruído
else
prnseq = 8000; // escala de saída do valor de ruído
fb =(sreg.bt.b0)^(sreg.bt.b1); //XOR bits 0,1
fb ^=(sreg.bt.b11)^(sreg.bt.b13);//combina bits 11,13 ->fb
sreg.regval<<=1; //shift register 1 bit à esquerda
sreg.bt.b0 = fb; //realiza a realimentação
output_sample(prnseq); //sequência de saída
return; //retorna da interrupção
}
void main()
{
sreg.regval = 0xFFFF; //inicializa o shift register
fb = 1; //valor inicial de realimentação
comm_intr(); //inicializa DSK
while (1); //loop
}
```

Fig. IV.15 – Geração da LFSR

IV.4.4 – Simulação de Sistema Exemplo: Redução de Ruído Sobre Sinal Senoidal

O trabalho implementa, agora, no DSK6713, o sistema exemplo apresentado em II.3.5 e representado na figura III.4. O sistema será executado em tempo real e não mais por simulação numérica em software. O modelo de redução de ruído senoidal usando filtro FIR e algoritmo LMS foi desenvolvido em C++, compilado e gravado no DSK6713.

O sinal desejado, d, de freqüência igual a 200Hz será aplicado à entrada do canal esquerdo do DSK6713 e uma interferência senoidal de freqüência igual a 500Hz, u, à entrada do canal direito. Os sinais são amostrados e seus valores armazenados em uma estrutura de 32 bits, 16 bits para cada canal. A freqüência de amostragem é de 8KHz, e o sinal de saída será enviado à saída do DSK. Como o sistema está funcionando em tempo real, o sinal de saída foi adquirido por um osciloscópio com capacidade de realizar FFT do sinal de entrada. O sistema está representado na figura IV.16.

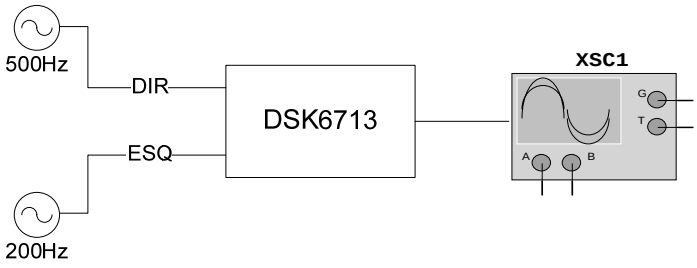


Fig. IV.16 – Sistema para redução em tempo real

O osciloscópio xsc1 é um osciloscópio digital com capacidade de realizar FFT em tempo real e com capacidade de armazenar dados e transferi-los ao computador.

O programa da figura IV.17 foi desenvolvido utilizando os códigos mostrados anteriormente e implementa o algoritmo LMS e filtro FIR.

```
#include "DSK6713_AIC23.h"                    // inicializa o CODEC AIC23
Uint32 fs=DSK6713_AIC23_FREQ_8KHZ;    // taxa de amostragem
#define Esq 0                                 //canal esquerdo
#define Dir 1                                 //canal direito
#define mu 1E-13                              //fator de convergência
#define N 30                                  //número de coeficientes
float c[N];                                   //coeficientes
float buf[N];                                 //buffer de entrada do filtro adaptativo
short saida;                                  //saída
short op = 1;                                 //controle do tipo de saída
volatile union{unsigned int uint; short channel[2];}AIC23_data;
interrupt void c_int11()                      //rotina de interrupção
{
short i;
float yn=0, E=0, sn=0, d=0, n=0;
AIC23_data.uint = input_sample();             //lê 32 bits (16 para cada canal)
d =(AIC23_data.channel[Esq]);                 //separa canal esquerdo
n = (AIC23_data.channel[Dir]);                //separa canal direito
sn = d + n;                                   //d+n soma sinal com ruído
buf[0] = n;                                   //valor de entrada para o filtro FIR: n
for (i = 0; i < N; i++)                       //calcula o filtro FIR
yn += (c[i] * buf[i]);                        //saída do filtro
E = (d + n) - yn;                             //sinal de erro: (d+n)-yn
for (i = N-1; i >= 0; i--)                    //atualiza coeficientes e buffer
{
c[i] = c[i] + mu*E*buf[i];                    //atualiza coeficientes segundo LMS
buf[i] = buf[i-1];                            //atualiza amostra
}
if(op == 1)                                   //testa o tipo de saída
saida=((short)E);                             //envia erro E para a saída
else if(op==2)                                //testa o tipo de saída
saida=((short)sn);                            //envia para a saída (d+n)
output_sample(saida);                         //gera saída
return;
}
void main()
{
short T=0;
for (T = 0; T < 40; T++)
{
c[T] = 0;                                     //inicializa coeficientes e buffer
buf[T] = 0;
}
comm_intr();                                  //inicializa DSK
while(1);                                     //loop
}
```

Fig. IV.17 – Filtro FIR e algoritmo LMS

Para facilitar a apresentação de forma gráfica, os dados foram transferidos ao MATLAB e aplicada FFT, gerando resultados na figura IV.18 a IV.21:

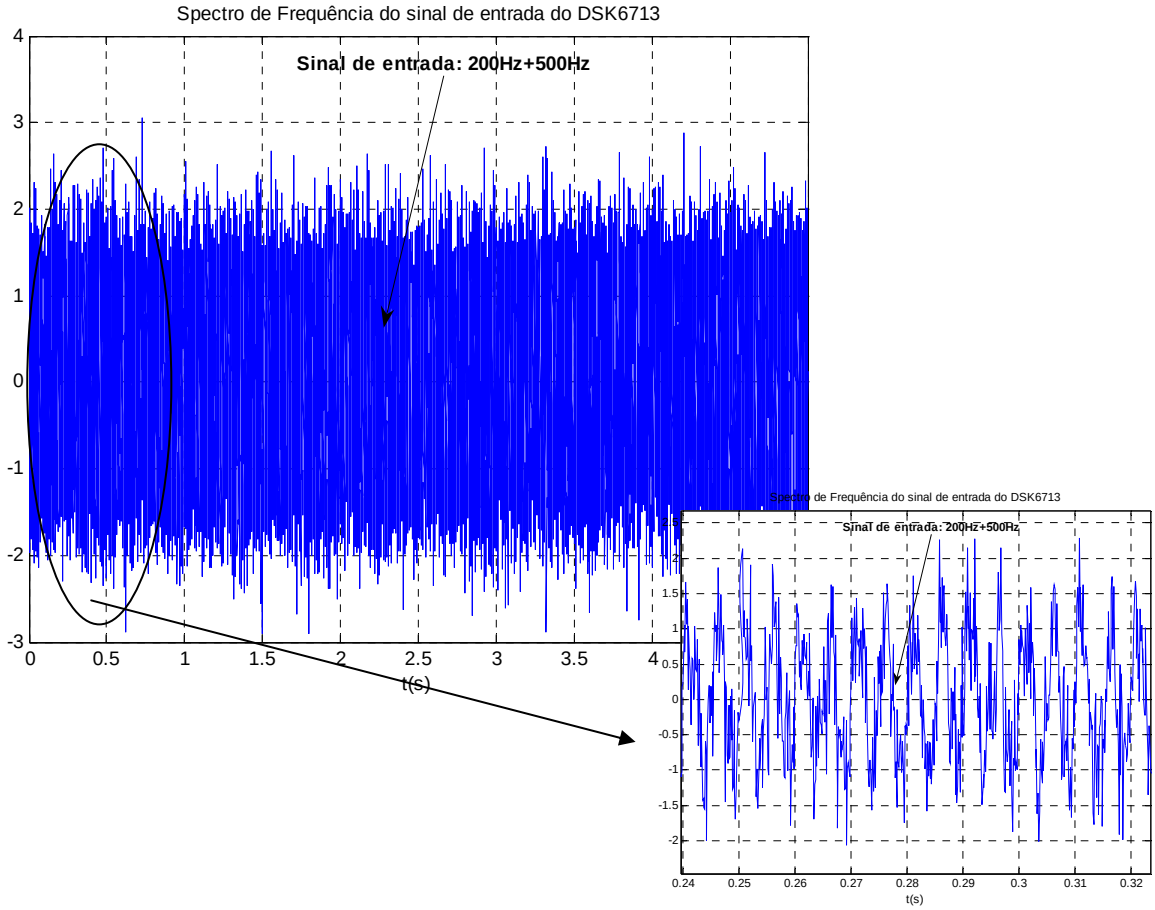


Fig. IV.18 – Sinal de entrada do DSK6713: 200Hz+500Hz(sinal + ruído)

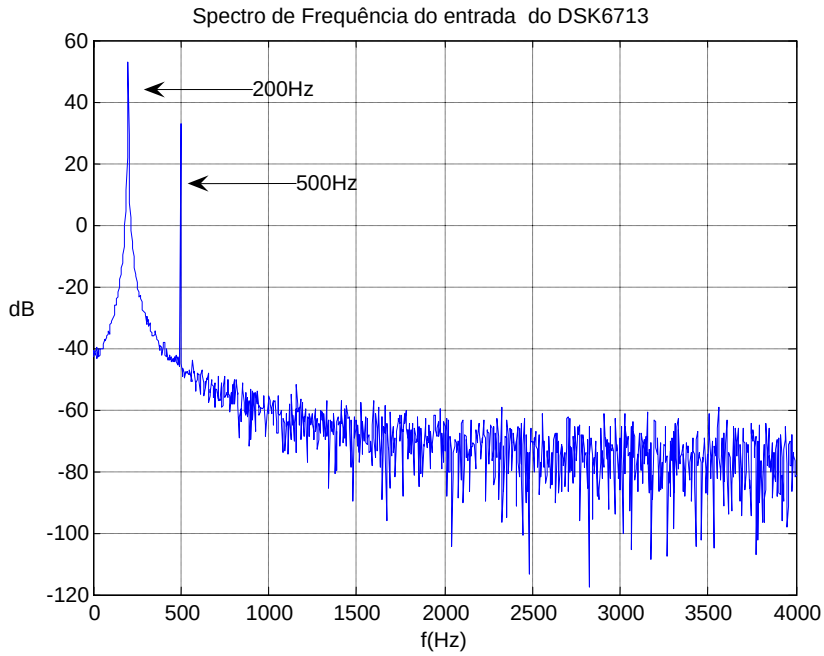
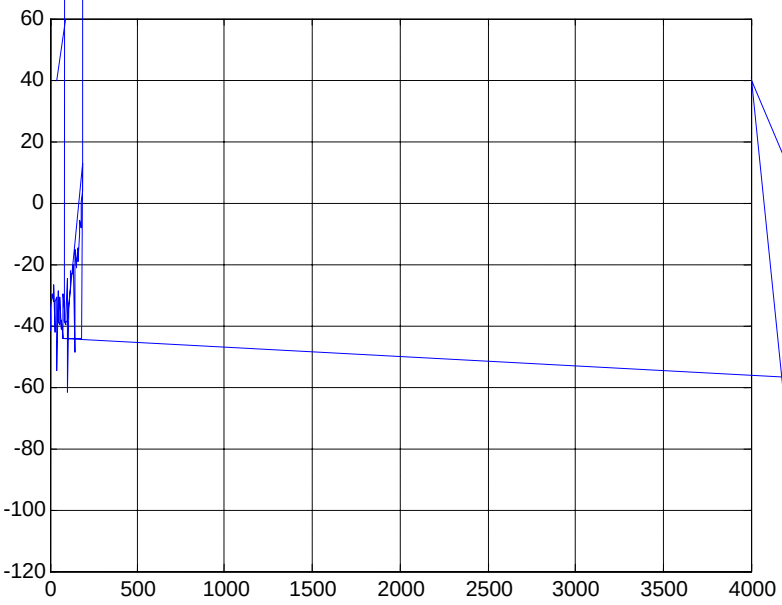


Fig. IV.19 – Espectro do sinal de entrada


60
40
20
0
-20
-40
-60
-80
-100
-120
0
500
1000
1500
2000
2500
3000
3500
4000

IV.5 – Duto: Levantamento dos Parâmetros do Caminho Secundário no Modo Off-line

O levantamento da função de transferência do caminho secundário foi realizada utilizando-se a técnica apresentada no item II.3.4.1. O ruído branco é gerado pelo sistema e é realizada uma aquisição com duração de 5 segundos a uma taxa de 8 kHz. O dispositivo pode ser revisto na figura II.13.

O ruído branco é gerado utilizando o código relativo à figura IV.15 e o sistema é implementado seguindo o pseudocódigo:

1. Gerar ruído branco $x(n)$: o ruído é enviado ao alto-falante de cancelamento e utilizado como referência de entrada;
2. Ler o sinal proveniente do microfone de erro $d(n)$;
3. Computar a resposta do filtro $y(n)$ utilizando o algoritmo LMS:
    a. Computar a saída do filtro adaptativo $\hat{S}(z)$

    $$y(n) = \sum_{i=0}^{M} w_i(n) x(n-i)$$

    b. Calcular o sinal de erro $e(n)$

    $$e(n) = d(n) - y(n)$$

    c. Atualizar os coeficientes do filtro

► A troca de dados entre o software de desenvolvimento do DSK6713 e o MATLAB é realizada por meio de uma linguagem *script* chamada GEL(“General Extension Language”). Ela permite que variáveis sejam gravadas no formato “.dat” e que, tais arquivos, possam ser abertos no MATLAB utilizando-se o comando “ccs_read_data(‘nome_arq.dat’);”.

Usando o procedimento acima, varias simulações foram realizadas para se alcançar um melhor desempenho computacional em relação à ordem do filtro a ser utilizado. A função de transferência do caminho secundário foi levantada usando-se um filtro adaptativo de ordem 40, cuja resposta e coeficientes variaram no tempo com valores mostrados na figura IV.22 a IV.24:

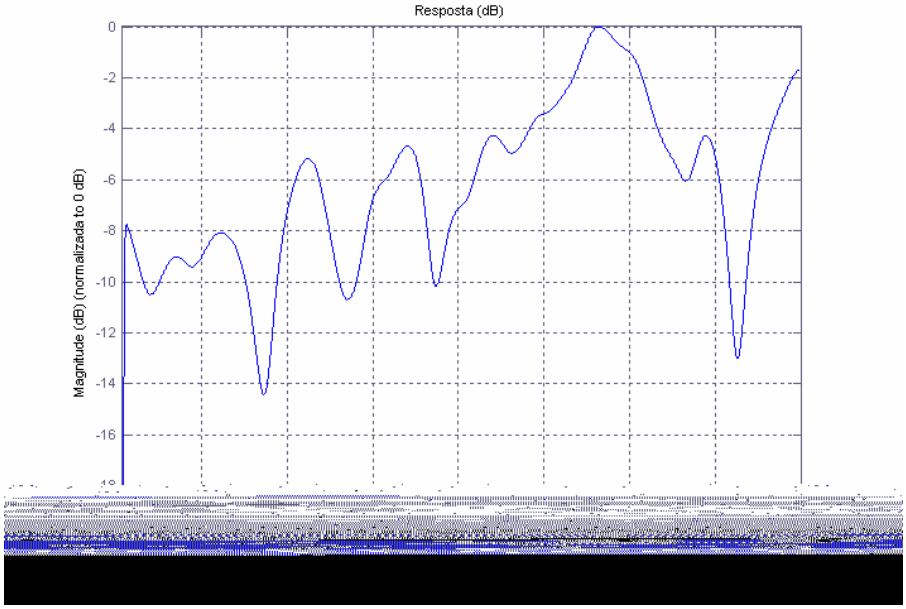


Fig. IV.22 – Resposta em freqüência do caminho secundário: magnitude

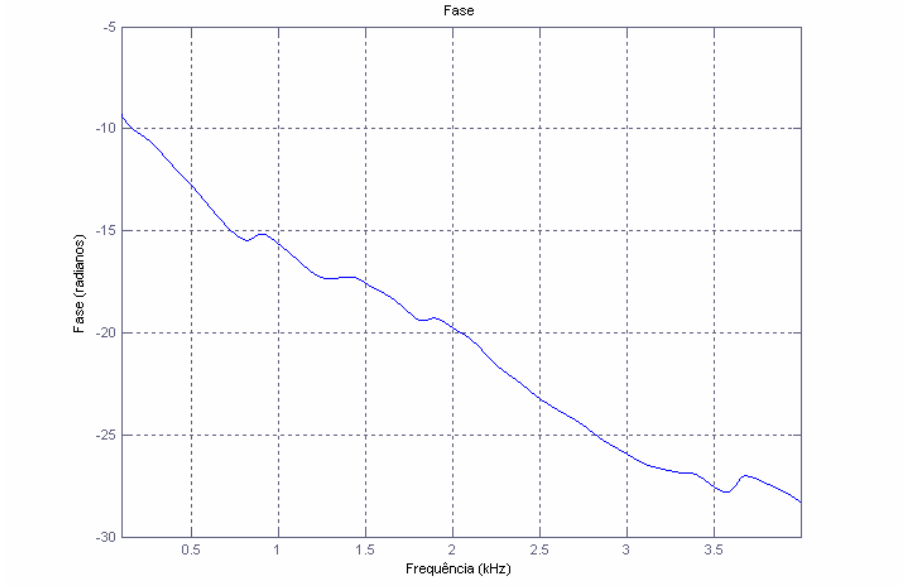


Fig. IV.23 – Resposta em freqüência do caminho secundário: fase

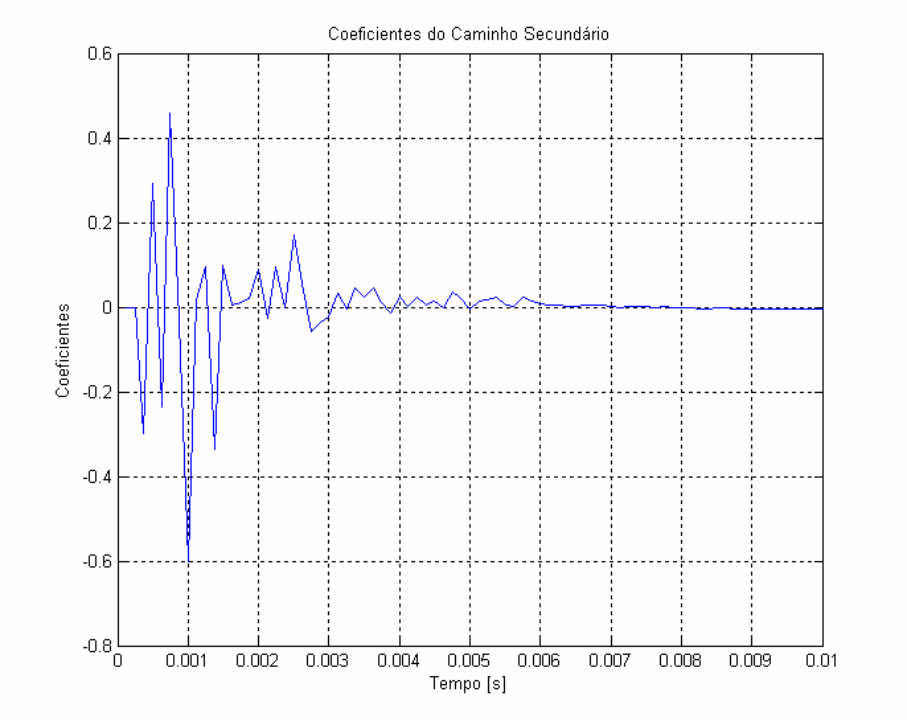


Fig. IV.24 – Comportamento dos coeficientes do filtro do caminho secundário

O comportamento dos sinais na identificação do caminho secundário podem ser observados, no tempo, na figura IV.25:

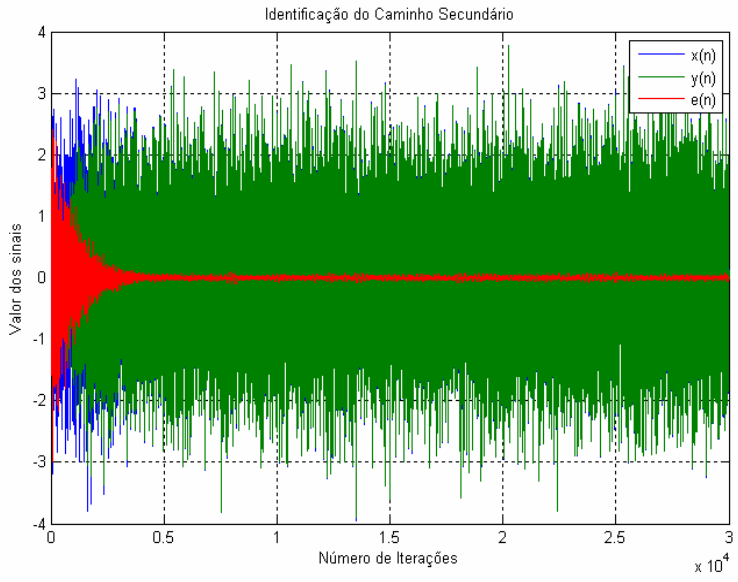


Fig. IV.25 – Comportamento dos sinais na identificação do caminho secundário

Os coeficientes do filtro que representam o caminho secundário serão então utilizados em seguida na fase on-line de cancelamento de ruído.

IV.5.1 – Duto: Ruído a Ser Cancelado

Para fins de simulação do sistema duto, que pode representar, por exemplo, um duto de sistema de ventilação e ar-condicionado, será gerado um ruído característico de um motor de ventilador. Esse ruído foi gravado e depois levantado suas características de freqüência, figura IV.26:

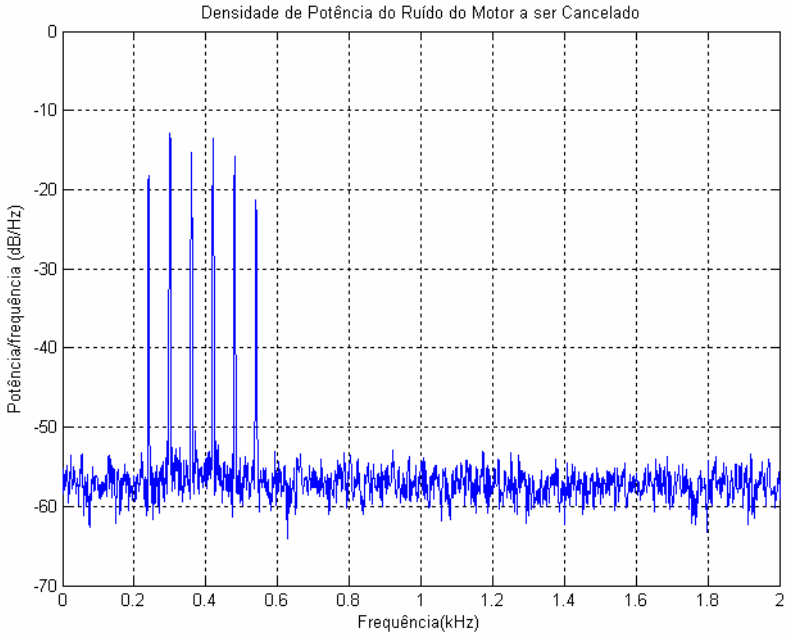


Fig. IV.26 – Espectro do sinal de ruído a ser cancelado

IV.5.2 – Cancelamento On-line

De posse do modelo do caminho secundário e do sinal de ruído a ser cancelado, o sistema deve agora realizar o cancelamento on-line do ruído do motor. Esse processo foi mostrado na figura II.12 e deve seguir o algoritmo apresentado no pseudocódigo da figura III.9.

Várias simulações foram realizadas para se chegar a valores de μ e da ordem do filtro que fossem satisfatórias à solução do problema. Novamente problemas como overflow para filtros de ordem muito alta se apresentaram. Resultados inesperados como travamento e não convergência do algoritmo foram gerados em função do problema da causalidade[23][24][25].

Da mesma forma que na fase off-line, os sinais foram armazenados em memória e depois transferidos ao MATLAB para análise.

Os resultados obtidos estão apresentados nas figuras IV.27 a IV.29.

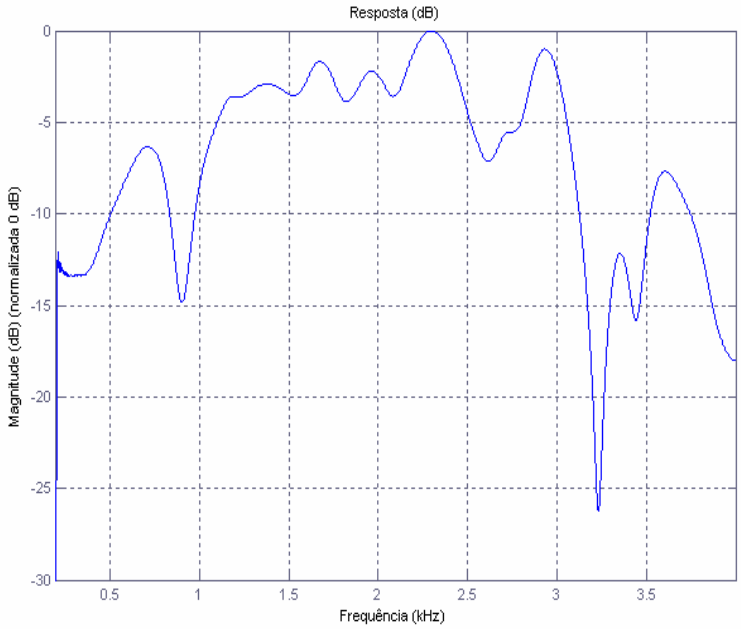


Fig. IV.27 – Resposta em Freqüência do caminho primário: magnitude

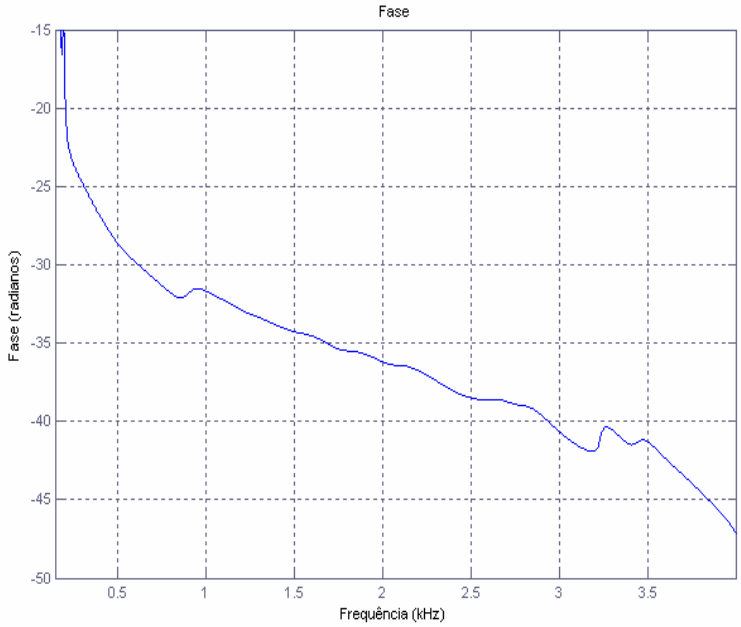


Fig. IV.28 – Resposta em freqüência do caminho primário: fase

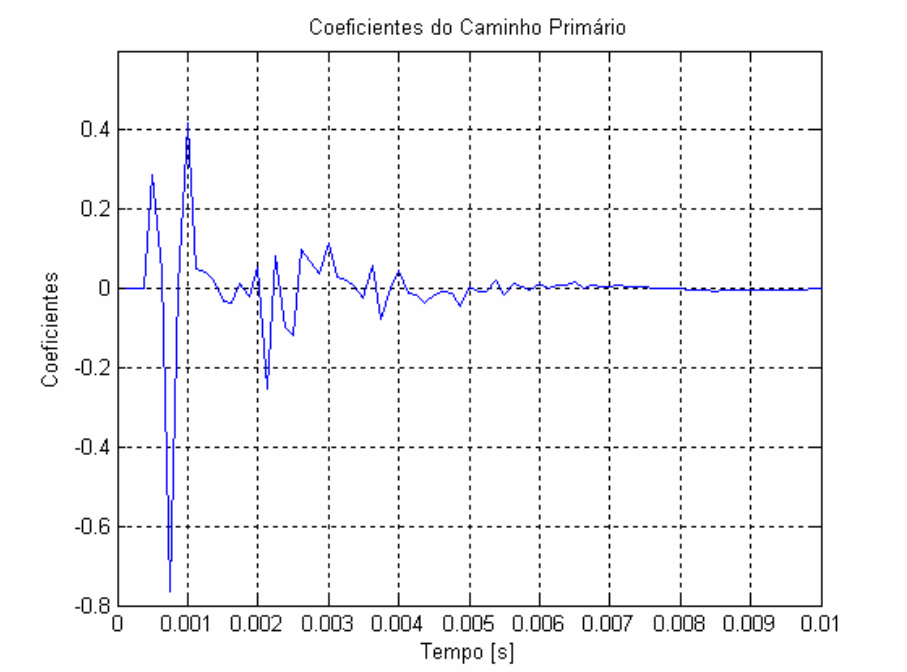


Fig. IV.29 – Comportamento dos coeficientes do filtro do caminho primário

Os sinais de ruído, cancelamento e erro foram gravados por 7.5 segundos, 60.000 amostragens com $Fs = 8000$. Esse tempo ficou limitado pelo tamanho de memória disponível no DSK6713. A figura IV.30 mostra o comportamento dos sinais para um filtro de ordem 50.

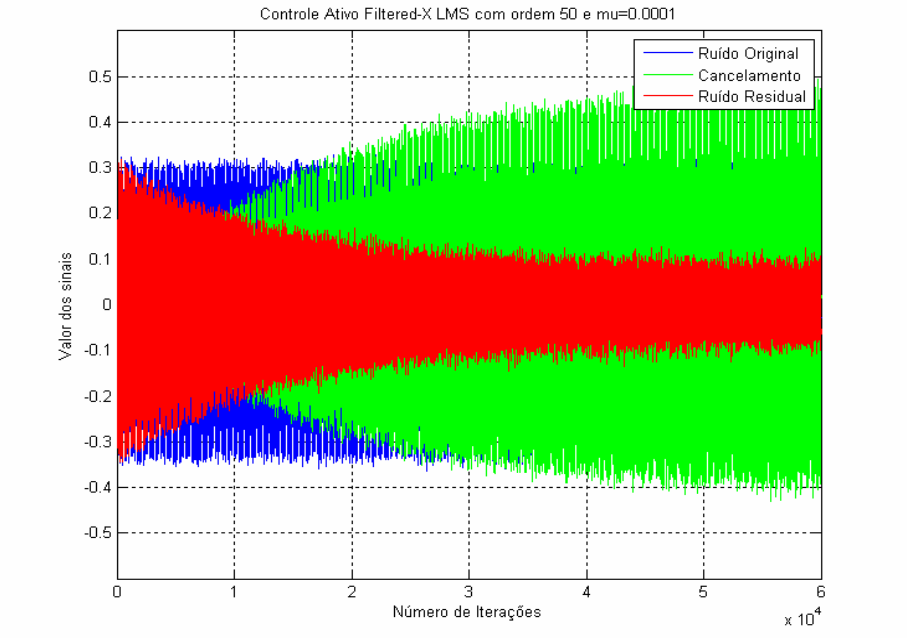


Fig. IV.30 – Comportamento sinais no cancelamento de ruído com ordem de filtro 50

Pela figura anterior, nota-se que em torno de 5 segundos, o sinal de erro já se encontra estável, mas em um nível não satisfatório.

Na figura IV.31 fica mais evidente o cancelamento, da ordem de 6dB em média.

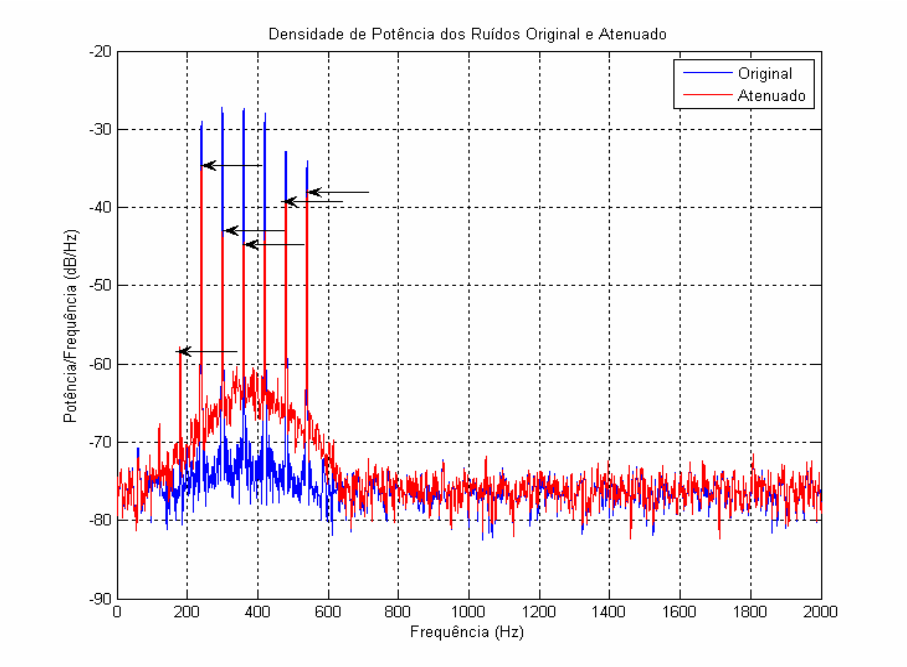


Fig. IV.31 – Cancelamento do ruído do motor em função da freqüência

Fica também evidente na figura IV.31 que o sistema gerou na saída ruídos que, apesar de níveis quase desprezíveis, não estavam presentes no sinal original. Realizando outras simulações, tal efeito foi diminuído aumentando-se a ordem do filtro adaptativo. Valores de ordem do filtro em torno de 250 se mostraram adequados. Valores mais altos não melhoraram o desempenho do sistema e levaram o algoritmo à instabilidade. A figura IV.32 mostra o comportamento dos sinais com filtro de ordem 250.

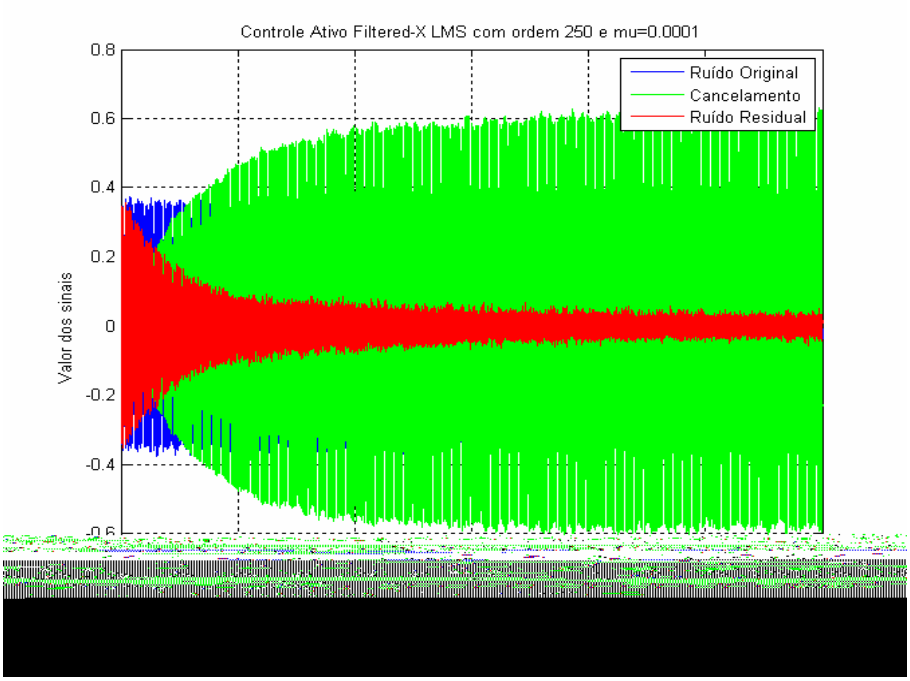


Fig. IV.32 – Comportamento sinais no cancelamento de ruído com ordem de filtro 250

Com esse valor de ordem do filtro, o sistema entra em regime estacionário em torno 3.5 segundos. A figura IV.33 mostra que a atenuação chegou perto dos 20 dB em média e que o sistema não introduziu novos ruídos indesejados no sinal.

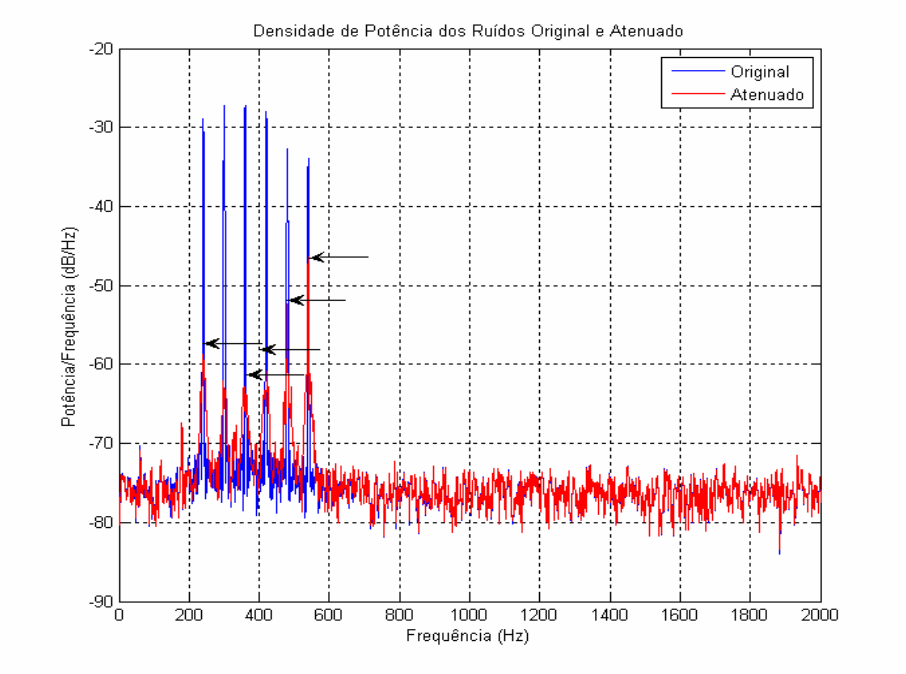


Fig. IV.33 – Cancelamento do ruído do motor em função da freqüência

Alguns aspectos dos testes devem ser ressaltados:

- sensibilidade ao posicionamento dos microfones de erro e referência;
- sensibilidade ao posicionamento do sistema completo: os resultados são sensíveis à obstáculos colocados perto da extremidade aberta do duto.

Essas características foram detectadas no decorrer dos testes, já que, os mesmos testes realizados em condições idênticas de setup eletrônico apresentavam resultados incoerentes. Detectadas estas interferências, os resultados passaram a apresentar repetibilidade esperada para o sistema[24]. Outra dificuldade encontrada foi o controle de ganho dos microfones. O controle de ganho foi realizado manualmente. O controle automático de ganho deve ser implementado em trabalhos futuros como sugerido na figura IV.34[25].

Contornados os obstáculos de desenvolvimento do software para o sistema DSP e da configuração física da montagem do hardware, os resultados obtidos se mostraram compatíveis com a teoria aplicada.

A figura seguinte mostra a inclusão, a ser implementada em trabalhos futuros, dos controles automáticos de ganho CAG1 e CAG2.

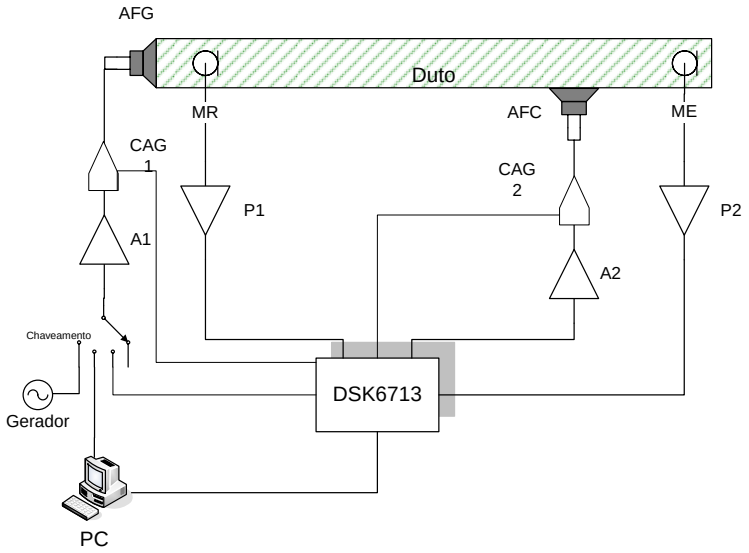


Fig. IV.34– Inclusão de controle de ganho no sistema

CONCLUSÃO E TRABALHOS FUTUROS

Esse trabalho empregou técnicas do processamento digital de sinais no cancelamento ativo de ruído acústico. Para tanto, técnicas para minimizar problemas de causalidade do controlador e das dimensões das zonas de cancelamento de ruído foram apresentadas.

No Capítulo I, foram apresentados conceitos teóricos sobre as várias topologias para controle ativo de ruído com ênfase no controle “feedforward” e sistemas do tipo SISO.

No Capítulo II foram estudados algoritmos da família LMS e foram realizadas simulações para demonstrar sua capacidade de seguimento, convergência e estabilidade. Foram também apresentados os filtros FIR e IIR. O objetivo deste estudo foi, baseado na solução de Wiener-Hopf, obter o conjunto de coeficientes ótimos para o controlador do sistema, de forma a minimizar a função custo *J*, ou, em outras palavras, fazer a melhor estimação do sinal desejado *d(n)* em termos do critério do MSE.

No Capítulo III, foi proposto um sistema multicanal para cancelamento de ruídos em pequenos ambientes confinados. Para a modelagem dos caminhos de cancelamento foi empregado o algoritmo FXLMS. Para tanto, foi desenvolvido um programa na linguagem C para simulação dos filtros e da variação de seus parâmetros, com apresentação dos resultados de forma gráfica. Os resultados obtidos comprovaram que o sistema se comportou dentro dos valores teóricos esperados e mostrou a sensibilidade à variação dos parâmetros de velocidade e ordem dos filtros. Os aspectos da implementação e da influência da representação numérica dos coeficientes na resposta dos filtros foram abordadas de modo a evitar problemas de overflow e convergência dos algoritmos.

No Capítulo IV o modelo teórico foi experimentalmente testado por meio de um sistema de duto com microfones como sensores e alto-falantes como atuadores. A implementação, em tempo-real, foi realizada em um sistema de desenvolvimento DSP DSK6713. Por meio deste arranjo verificou-se que o modelo prático se mostrou estável e com resultados compatíveis com o modelo teórico, levando em consideração as questões restritivas e de linearidade. Resultados práticos com atenuações da ordem de 20/30 dB foram alcançados, mas o sistema ficou limitado a freqüência de amostragem de 8kHz. A otimização do código empregado, utilizando a Linguagem Assembler, pode ser um viés para a melhoria do desempenho global do sistema.

O estudo foi baseado no conceito de ondas planas e ruído uniformemente distribuído. Um estudo mais profundo das características acústicas do ambiente, da não linearidade do meio e dos transdutores deve servir como base para novas pesquisas na área do cancelamento ativo de ruídos e, como conseqüência, o desempenho e a robustez do controlador devem ser investigados para ambientes mais reverberantes e ruidosos.

Embora o foco do trabalho tenha sido o cancelamento de ruído acústico, os resultados e as técnicas desenvolvidas podem ser aplicadas a outras classes problemas como, por exemplo, o controle de vibrações e processamento de imagens.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] SCOTT D. Snyder, *Active Noise Primer*. Springer, Adelaide. Austrália. 2000.

[2] NELSON, P.A.; ELLIOT, S.J., *Active Control of Sound*, Academic Press, London, 1992.

[3] ELLIOT, S. J., *Down With Noise*, IEEE Spectrum, June, 1999.

[4] KUO, Sen M.; MORGAN, Dennis R., *Active Noise Control Systems*. Wiley, NY, USA. 1996.

[5] WHEELER, P. D. and D Smeatham, *On Spatial Variability in the Attenuation Performance of Active Hearing Protectors*, Applied Acoustics, Elsevier Science Publishers Ltd., England, 1992.

[6] HANSEN Colin H., *Understanding Active Noise Cnacellation*. Spon Press, London, 2001.

[7] ZHANG Ming, LAN Hui, Wee Ser. *Cross-updated active noise control system with online secondary path modeling.* Speech and Audio Processing, IEEE Transactions on. Publication: Jul 2001.Volume: 9, Issue: 5. Page(s): 598-602.

[8] MECKLENBRAUKER W., CLASSEN. *Comparison of the convergence of two algorithms for adaptive FIR digital filters*. *IEEE Transactions on Circuits and Systems*, vol. 28, no. 6, pp. 510–518, 1981.

[9] MANOLAKIS, Dimitris G. , INGLE,Vinay K. I, KOGON, Stephen M. *Statistical and Adaptive Signal Processing: Spectral Estimation, Signal Modeling, Adaptive Filtering and Array Processing.* Artech House Publishers, 2005.

[10] PROAKIS, John G.; MANOLAKIS, Dimitris G., *Digital Signal Processing, Principles, Algorithms and Applications*, Prentice Hall, New Jersey, USA, 1996.

[11] FIELDER,L. D., *"Human Auditory Capabilities and Their Consequences in Digital-Audio Converter Design"*, Audio in Digital Times, Proc. Audio Eng. Soc. 7th Inter. Conf., Toronto, Ont., Canada, May 14th-17th 1989, pp. 45-62.

[12] LJUNG S., LJUNG L. *Error propagation properties of recursive least squares adaptation algorithms.* A bridge between control science and technology. Volume 2 (A86-33143 14- 63). Oxford and New York, Pergamon Press, 1985, p. 677-681.

[13] HAYKIN, Simon. *Adaptive Filter Theory*, 3rd Edition, Prentice-Hall, Upper Side River, NJ, 1996.

[14] MORGAN, D. R., "*An analysis of multiple correlation cancellation loops with a filter in the auxiliary path*," IEEE Transactions on Acoustics, Speech, and Signal Processing, August 1980, pp. 454-467.

[15] BURGESS, J.C., *Active Adaptive Sound Control in a Duct: A Computer Simulation. J. Acoust*. Soc. A.m., Vol. 70, Sept. 1981, pp715-726.

[16] L. Eriksson; M. Allie, "*Use of random noise for on-line transducer modeling in an adaptive active attenuation system*," Journal of the Acoustic Society of America, vol. 85, pp. 797-802, Feb. 1989.

[17] XIAO, Lin; Shau-Shiun Jan, "*Adaptive Noise Canceling in an Acoustic Duct, Project Report for "Adaptive Signal Processing*", Stanford University.

[18] TLV320AIC23 Stereo Audio Codec, 8- to 96-kHz, with Integrated Headphone Amplifier Data Manual, SLWS106G, Texas Instruments, Dallas, TX, 2003.

[19] WIDROW B., STEARNS D. *Adaptive Signal Processing.* Englewood Cliffs, Prentice-Hall, 1985.

[20] KINSLER, Frey; Coppens; Sanders. *Fundamentals of Acoustics*, 3rd edition, Wiley&Sons, 1982.

[21] OPPENHEIN, A.V.; SCHAFRE, R.W. *Digital Signal Processing*. USA .Prentice-Hall.1989.

[22] LIU, S., J. Yuan; K.-Y. Fung. *Robust active control of broadband noise in finite ducts*. Journal of the Acoustic Society of America, Vol. 111, No. 6, June 2002.

[23] HONG J.; S. Park, J. Yoon; J. Koh, D. Kim, *Design of Real Random Number Generator.* Proceeding (393) Communication Systems and Networks, 2003.

[24] ELLIOTT S. J., BOUCHER C. C., NELSON P. A. *The effects of acoustic. coupling on the stability and convergence of independent controllers*. The Journal of the Acoustical Society of America -- June 1997 -- Volume 101, Issue 6, pp. 3498-3516**.**

[25] BOUCHARD, M.Sch. *Multichannel affine and fast affine projection algorithms for active noise control and acoustic equalization systems*. Inf. Technol. & Eng., Univ. of Ottawa, Ont., Canada; This paper appears in: Speech and Audio Processing, IEEE Transactions on Publication Date: Jan 2003Volume: 11, Issue: 1On page(s): 54- 60.

Apêndice A

Listagem dos códigos em linguagem C e MATLAB® utilizados nas simulações numéricas e no sistema de desenvolvimento DSK6713.

A.1 – Código para teste do algoritmo LMS (MATLAB)

```
%
% Resposta do Algoritmo LMS a entradas com ruído branco e colorido
%

% Condições iniciais do sistema P(z)
p(1,:) = [1 0.5 -1 0.3 -0.4 0.5];

%Ordem do filtro adaptativo = 5
M = 6;

%Taxa de convergência (mu)
mu = 0.01;

%Inicializações e geração de sinais

%Inicializa vetors de erro
err_branco = zeros(1,2000);
err_cor = zeros(1,2000);

%Ruído branco
r_bco = randn(1,2000);

%Ruído cor: ruído branco pssado por filtro passa baixa
[Num,Den]=butter(10,0.5);
r_cor = filter(Num,Den,r_bco);
r_cor = r_cor/std(r_cor); % normaliza

%Gera a entrada desejada
v = sqrt(0.001)*randn(1,2000);
tic %inicia a contagem de tempo
%Parametros do filtro LMS
LMS_coef = zeros(M,length(r_bco)+1);
LMS_coef_cor = LMS_coef;
vet = zeros(M,1);
vetc = zeros(M,1);

for i = 1:100

   for j = 1:length(r_bco) %Atualiza coeficientes

      p(j+1,:)=p(j,:)+sqrt(0.001)*randn(1,M);

      vet(1) = r_bco(j);
      vetc(1) = r_cor(j);

      d(j) = p(j+1,:)*vet + v(j);
      d_color(j)=p(j+1,:)*vetc + v(j);

      e(j) = d(j) - LMS_coef(:,j)' * vet;
      e_color(j) = d_color(j) - LMS_coef_cor(:,j)' * vetc;

      LMS_coef(:,j+1) = LMS_coef(:,j) + mu * vet * e(j);
      LMS_coef_cor(:,j+1) = LMS_coef_cor(:,j) + mu * vetc * e_color(j);

      for k = M:-1:2
         vet(k) = vet(k-1);
         vetc(k)=vetc(k-1);
```

```
      end;
   end;

   e_time_LMS(i) = toc;

%atualiza vetores de erro
   err_branco = err_branco + e.^2;
   err_cor = err_cor + e_color.^2;

end;

figure(1)
plot(1:2001,p(:,1),1:2001,LMS_coef(1,:))
legend('Sistema H(z), mu=0.05','LMS')
title('Coeficiente #1;Ruído Branco');
figure(2)
plot(1:2001,p(:,5),1:2001,LMS_coef(5,:))
legend('Sistema H(z), mu=0.05','LMS')
title('Coeficiente #5;Ruído Branco');
figure(3)
plot(1:2001,p(:,1),1:2001,LMS_coef_cor(1,:))
legend('Sistema H(z), mu=0.05','LMS')
title('Coeficiente #1;Ruído Colorido');
figure(4)
plot(1:2001,p(:,5),1:2001,LMS_coef(5,:))
legend('Sistema H(z), mu=0.05','LMS')
title('Coeficiente #5;Ruído Colorido');
```

A.2 - Código para teste do algoritmo RLS (MATLAB)

```
%
% Resposta do Algoritmo RLS a entradas com ruído branco e colorido
%

% Condições iniciais do sistema P(z)
p(1,:) = [1 0.5 -1 0.3 -0.4 0.5];

%Ordem do filtro adaptativo = 6
M = 6;

%Fator de esquecimento
lam = 0.8;
delta=0.0001;

%Inicializações e geração de sinais

%Inicializa vetors de erro
err_branco = zeros(1,2000);
err_cor = zeros(1,2000);

%Ruído branco
r_bco = randn(1,2000);

%Ruído cor: ruído branco pssado por filtro passa baixa
[Num,Den]=butter(10,0.5);
r_cor = filter(Num,Den,r_bco);
r_cor = r_cor/std(r_cor); % normaliza

%Gera a entrada desejada
v = sqrt(0.001)*randn(1,2000);
tic %inicia a contagem de tempo

%Parametros do filtro RLS
RLS_coef = zeros(M,length(r_bco)+1);
RLS_coef_cor = RLS_coef;
vet = zeros(M,1);
vetc = zeros(M,1);
Prb=1/delta*eye(M);
```

```
Pcl=1/delta*eye(M);

for i = 1:100

   for j = 1:length(r_bco) %Atualiza coeficientes

      p(j+1,:)=p(j,:)+sqrt(0.001)*randn(1,M);

      vet(1) = r_bco(j);
      vetc(1) = r_cor(j);

      d(j) = p(j+1,:)*vet + v(j);
      d_color(j)=p(j+1,:)*vetc + v(j);

      pi_vet=Prb*vet;
      pi_vetc=Pcl*vetc;

      k_vec=pi_vet/(lam+vet'*pi_vet);
      k_vecc=pi_vetc/(lam+vetc'*pi_vetc);

      xi_rb(j)=d(j)-RLS_coef(:,j)'*vet;
      xi_cl(j)=d_color(j)-RLS_coef_cor(:,j)'*vetc;

      RLS_coef(:,j+1) = RLS_coef(:,j) + k_vec*xi_rb(j);
      RLS_coef_cor(:,j+1) = RLS_coef_cor(:,j) + k_vecc*xi_cl(j);

      Prb=1/lam*(Prb-k_vec*vet'*Prb);
      Pcl=1/lam*(Pcl-k_vecc*vetc'*Pcl);

      for k = M:-1:2
         vet(k) = vet(k-1);
         vetc(k)=vetc(k-1);
      end;
   end;

   e_time_RLS(i) = toc;

%atualiza vetores de erro
   err_branco = err_branco + xi_rb.^2;
   err_cor = err_cor + xi_cl.^2;

end;

figure(1)
plot(1:2001,p(:,1),1:2001,RLS_coef(1,:))
legend('Sistema H(z), lam=0.95','RLS')
title('Coeficiente #1 - Ruído Branco');
figure(2)
plot(1:2001,p(:,5),1:2001,RLS_coef(5,:))
legend('Sistema H(z), lam=0.95','RLS')
title('Coeficiente #5 - Ruído Branco');
figure(3)
plot(1:2001,p(:,1),1:2001,RLS_coef_cor(1,:))
legend('Sistema H(z), lam=0.95','RLS')
title('Coeficiente #1 - Ruído Colorido');
figure(4)
plot(1:2001,p(:,5),1:2001,RLS_coef(5,:))
legend('Sistema H(z), lam=0.95','RLS')
title('Coeficiente #5 - Ruído Colorido');
```

A.3 – Código C++ para simulação do Sistema Proposto CARA – (Borland C++ 5.0)

Interface gráfica:

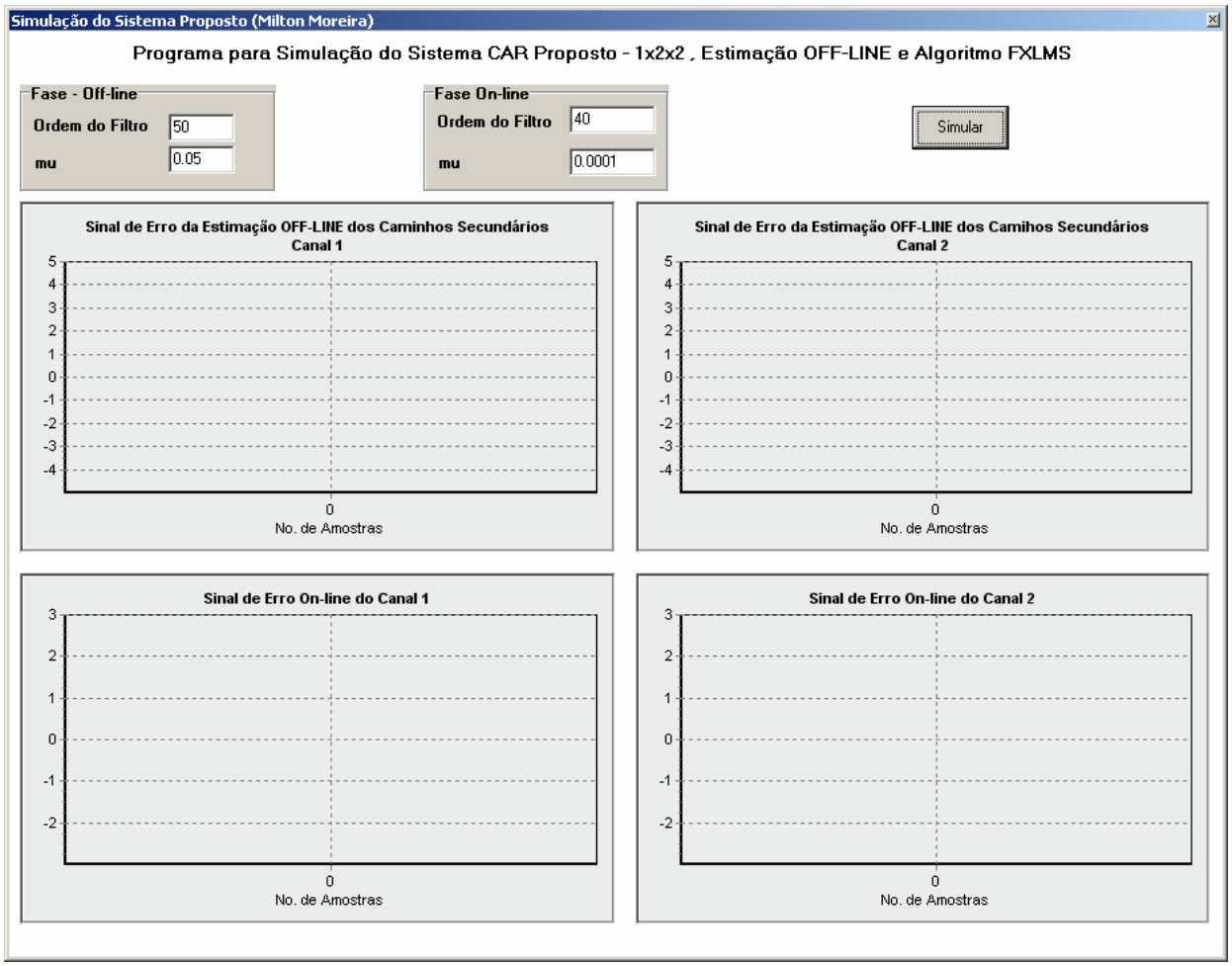


Fig. A3.1 – Interface gráfica para simulação em C++

Código:

```
#include <vcl.h>
#pragma hdrstop

#include "Unit1.h"

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <f_fir.h>
#include <f_iir.h>
#include <ruido.h>
#include <atualiza.h>
#include <coef_filtro.h>

#define MAX  512            // ordem dos arrays

float w1[MAX];              // pesos W1(z)
float w2[MAX];              // pesos W2(z)
float sh11[MAX];            // pesos S^11(z)
float sh12[MAX];            // pesos S^12(z)
float sh21[MAX];            // pesos S^21(z)
```

```
float sh22[MAX];              // pesos S^22(z)

/*  variáveis utilizadas na modelagem off-line  */

float u[MAX];                  // x(n)
float d1offline[MAX];          // d1(n)
float d2offline[MAX];          // d2(n)

/*  variáveis utilizadas na modelagem on-line  */

float x[MAX];                 // x(n)
float y1[MAX];                // y1(n)
float y2[MAX];                // y2(n)
float yp11[MAX];              // y'11(n)
float yp12[MAX];              // y'12(n)
float yp21[MAX];              // y'21(n)
float yp22[MAX];              // y'22(n)
float d1[MAX];                // d1(n)
float d2[MAX];                // d2(n)
float xp11[MAX];              // x'11(n)
float xp12[MAX];              // x'12(n)
float xp21[MAX];              // x'21(n)
float xp22[MAX];              // x'22(n)

//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int j,k,i,nv;                 // indices
  int Ls=0;                     // ordem de S^mk(z)
  int Lw=0;                     // ordem de Wk(z)
  float mus=0;                 // passo p/atualizar S^mk(z)
  float muw1=0;                // passo p/atualizar W1(z)
  float muw2=0;                // passo p/atualizar W2(z)

/*  zera variáveis utilizadas na modelagem off-line */

  float uu=0;                   // novos dados x(n)
  float y1offline_n=0;          // novos dados y1(n)
  float y2offline_n=0;          // novos dados y2(n)
  float d1offline_n=0;          // novos dados d1(n)
  float d2offline_n=0;          // novos dados d2(n)
  float ee1=0;                  // novos dados e1(n)
  float ee2=0;                  // novos dados e2(n)

/*  zera variáveis utilizadas na modelagem on-line  */

  float xx=0;                   // novos dados x(n)
  float y1n=0;                  // novos dados y1(n)
```

```
  float xp21n=0;              // novos dados x'21(n)
  float xp22n=0;              // novos dados x'22(n)

// ponteiros para arquivos

  FILE *fpin;                 // ponteiro para arq. x(n) no modo on-line
  FILE *fpe1;                 // ponteiro para arq. e1(n) no modo on-line
  FILE *fpe2;                 // ponteiro para arq. e2(n) no modo on-line

  FILE *fpee1;
  FILE *fpee2;
  FILE *fpee3;
  FILE *fpee4;

  fpe1 = fopen("a1.dat","wb");
  fpe2 = fopen("a2.dat","wb");
  fpee1=fopen("ee1.dat","wb");
  fpee2=fopen("ee2.dat","wb");
  fpee3=fopen("ee3.dat","wb");
  fpee4=fopen("ee4.dat","wb");

  Ls = StrToFloat(o_offline->Text);
  mus = StrToFloat(u_offline->Text);
  Lw = StrToFloat(o_online->Text);
  muw1 = StrToFloat(u1_online->Text);
  muw2 = StrToFloat(u2_online->Text);
  muw1=muw2;

// zera arrayas

  for (k=0; k<=MAX-1; ++k)
  {
    w1[k] = (float) 0.0;
    w2[k] = (float) 0.0;
    sh11[k] = (float) 0.0;
    sh12[k] = (float) 0.0;
    sh21[k] = (float) 0.0;
    sh22[k] = (float) 0.0;

    u[k] = (float) 0.0;
    d1offline[k] = (float) 0.0;
    d2offline[k] = (float) 0.0;

    x[k] = (float) 0.0;
    y1[k] = (float) 0.0;
    y2[k] = (float) 0.0;
    yp11[k] = (float) 0.0;
    yp12[k] = (float) 0.0;
    yp21[k] = (float) 0.0;
    yp22[k] = (float) 0.0;
    d1[k] = (float) 0.0;
    d2[k] = (float) 0.0;
    xp11[k] = (float) 0.0;
    xp12[k] = (float) 0.0;
    xp21[k] = (float) 0.0;
    xp22[k] = (float) 0.0;
  }

//   Programa principal
//   Modelagem do caminho secundário off-line S11(z), S21(z)

  for (j=0; j<10000; j++)
  {
    uu = ruido() - 0.5;                 // gera ruído branco x(n)
    atualiza(u, MAX, uu);               // atualiza x(n)
    d1offline_n = f_iir(u, zeros, z_cs11, d1offline, polos, p_cs11, MAX, MAX);
                                        // filtro f_iir de x(n) por S11(z) para obter d1(n)
    d2offline_n = f_iir(u, zeros, z_cs21, d2offline, polos, p_cs21, MAX, MAX);
                                        // filtro f_iir de x(n) por S21(z) para obter d2(n)
```

```
   atualiza(d1offline, MAX, d1offline_n);                  // atualiza d1(n)
   atualiza(d2offline, MAX, d2offline_n);                  // atualiza d2(n)
   y1offline_n = f_fir(u, MAX, sh11, Ls);                  // filtro f_fir de x(n) por S^11(z) p/obter y1(n)
   y2offline_n = f_fir(u, MAX, sh21, Ls);                  // filtro f_fir de x(n) por S^21(z) p/obter y2(n)
   ee1 = d1offline_n - y1offline_n;                        // e1(n) = d1(n) - y1(n)
   ee2 = d2offline_n - y2offline_n;                        // e2(n) = d2(n) - y2(n)

   fwrite(&ee1, sizeof(float), 1, fpee1);
   fwrite(&ee2, sizeof(float), 1, fpee2);
   Chart1->Series[0]->AddXY(j,ee1,"",clGreen);
   Chart1->Series[1]->AddXY(j,ee2,"",clRed);
   for (k=0; k<=Ls-1; ++k)
   {
     sh11[k] = sh11[k] + mus * ee1 * u[MAX-1-k];
     sh21[k] = sh21[k] + mus * ee2 * u[MAX-1-k];       // atualiza coefs. de S^11(z) e S^21(z) usando LMS
   }
 }

 for (k=0; k<MAX-1; k++)
 {
   u[k] = (float) 0.0;       // zera x(n)
 }

// Modelagem do caminho secundário off-line S12(z), S22(z)

 for (j=0; j<10000; j++)
 {
   uu = ruido() - 0.5;                                   // gera ruído branco x(n)
   atualiza(u, MAX, uu);                                 // atualiza x(n)
   d1offline_n = f_iir(u, zeros, z_cs12, d1offline, polos, p_cs12, MAX, MAX);
                                                         // filtro f_iir de x(n) por S12(z) para obter d1(n)
   d2offline_n = f_iir(u, zeros, z_cs22, d2offline, polos, p_cs22, MAX, MAX);
                                                         // filtro f_iir de x(n) por S22(z) para obter d2(n)
   atualiza(d1offline, MAX, d1offline_n);                // atualiza d1(n)
   atualiza(d2offline, MAX, d2offline_n);                // atualiza d2(n)
   y1offline_n = f_fir(u, MAX, sh12, Ls);                // filtro f_fir de x(n) por S^12(z) p/obter y1(n)
   y2offline_n = f_fir(u, MAX, sh22, Ls);                // filtro f_fir de x(n) por S^22(z) p/obter y2(n)
   ee1 = d1offline_n - y1offline_n;                      // e1(n) = d1(n) - y1(n)
   ee2 = d2offline_n - y2offline_n;                      // e2(n) = d2(n) - y2(n)

   fwrite(&ee1, sizeof(float), 1, fpee3);
   fwrite(&ee2, sizeof(float), 1, fpee4);
   Chart2->Series[0]->AddXY(j,ee1,"",clTeeColor);
   Chart2->Series[1]->AddXY(j,ee2,"",clGreen);
   for (k=0; k<=Ls-1; ++k)
   {
     sh12[k] = sh12[k] + mus * ee1 * u[MAX-1-k] ;
     sh22[k] = sh22[k] + mus * ee2 * u[MAX-1-k] ;      // atualiza coefs. de S^12(z) e S^22(z) usando LMS
   }
 }

//   Fase On-line

 fpin=fopen("S100.dat","rb");                            // abre arquivo de sinal

 i=0;
 float v=0;

 while((fread(&xx,sizeof(float), 1, fpin)) == 1)         // dados do arquivo->x(n)
 {
   v=xx;
   atualiza(x, MAX, xx);                                 // atualiza x(n)
   d1n = f_iir(x, zeros, z_p1, d1, polos, p_p1, MAX, MAX);      // f_iir x(n) , P1(z) -> d1(n)
   atualiza(d1, MAX, d1n);                                      // atualiza d1(n)
   d2n = f_iir(x, zeros, z_p2, d2, polos, p_p2, MAX, MAX);      // f_iir x(n) , P2(z) -> d2(n)
   atualiza(d2, MAX, d2n);                                      // atualiza d2(n)
   y1n = f_fir(x, MAX, w1, Lw);                                 // f_fir x(n) , W1(z) -> y1(n)
   atualiza(y1, MAX, y1n);                                       // atualiza y1(n)
   y2n = f_fir(x, MAX, w2, Lw);                                  // f_fir  x(n) , W2(z) -> y2(n)
```

```
atualiza(y2, MAX, y2n);                                          // atualiza y2(n)
yp11n = f_iir(y1, zeros, z_cs11, yp11, polos, p_cs11, MAX, MAX);     // f_iir  y1(n) , S11(z) -> y'11(n)
atualiza(yp11, MAX, yp11n);                                      // atualiza y'11(n)
yp12n = f_iir(y2, zeros, z_cs12, yp12, polos, p_cs12, MAX, MAX);     // f_iir  y2(n) , S12(z) -> y'12(n)
atualiza(yp12, MAX, yp12n);                                      // atualiza y'12(n)
yp21n = f_iir(y1, zeros, z_cs21, yp21, polos, p_cs21, MAX, MAX);     // f_iir  y1(n) , S21(z) -> y'21(n)
atualiza(yp21, MAX, yp21n);                                      // atualiza y'21(n)
yp22n = f_iir(y2, zeros, z_cs22, yp22, polos, p_cs22, MAX, MAX);     // f_iir  y2(n) , S22(z) -> y'22(n)
atualiza(yp22, MAX, yp22n);                                      // atualiza y'22(n)
e1 = d1n - yp11n - yp12n;                                        // e1(n) = d1(n) - y'11(n) - y'12(n)
fwrite(&e1, sizeof(float), 1, fpe1);
e2 = d2n - yp21n - yp22n;                                        // e2(n) = d2(n) - y'21(n) - y'22(n)
fwrite(&e2, sizeof(float), 1, fpe2);

Chart3->Series[0]->AddXY(i,e1,"",clBlue);
Chart4->Series[0]->AddXY(i,e2,"",clRed);

Chart3->Series[1]->AddXY(i,v,"",clBlack);
Chart4->Series[1]->AddXY(i,v,"",clBlack);

i++;

xp11n = f_fir(x, MAX, sh11, Ls);                                 // f_fir  x(n) , S^11(z) -> x'11(n)
atualiza(xp11, MAX, xp11n);                                      // atualiza x'11(n)
xp12n = f_fir(x, MAX, sh21, Ls);                                 // f_fir  x(n) , S^21(z) -> x'12(n)
atualiza(xp12, MAX, xp12n);                                      // atualiza x'12(n)
xp21n = f_fir(x, MAX, sh12, Ls);                                 // f_fir  x(n) , S^12(z) -> x'21(n)
atualiza(xp21, MAX, xp21n);                                      // atualiza x'21(n)
xp22n = f_fir(x, MAX, sh22, Ls);                                 // f_fir  x(n) , S^22(z) -> x'22(n)
atualiza(xp22, MAX, xp22n);                                      // atualiza x'22(n)

for (j=0; j<=Lw-1; ++j)
```

```
 unsigned int dweebie:2;                    // Fills the 2 bit hole - bits 14-15
} bitval;

typedef union SHIFT_REG
{
 unsigned int regval;
 bitval bt;
} shift_reg;
short fb;
shift_reg sreg;                                    //estrutura do shift register
short prnseq;                                      //seqüência pseudo-randômica

interrupt void c_int11()                           //rotina de interrupção
void main()
{
comm_intr();                                       //inicializa DSK
                                                   //loop

    float c[taps], y[taps], x1[taps], x[taps], w[taps], yn, rn, en, e_n, j, val;
    int i, k, l=0, shift, value=0;
    sreg.regval = 0xFFFF;                          //inicializa o shift register
    fb = 1;                                        //valor inicial de realimentação
    comm_intr();
    if(sreg.bt.b0)                                 //sequence{1,-1}based on bit b0
            prnseq = -8000;                        //escala de saída do valor de ruído
   else
            prnseq = 8000;                         // escala de saída do valor de ruído
   fb =(sreg.bt.b0)^(sreg.bt.b1);                  //XOR bits 0,1
   fb ^=(sreg.bt.b11)^(sreg.bt.b13);               //com bits 11,13 ->fb
   sreg.regval<<=1;                                //shift register 1 bit à esquerda
   sreg.bt.b0 = fb;                                //realiza a realimentação

   for( i=0; i<taps; i++ )                         // zera variáveis
        x1[i] = w[i] = x[i] = c[i] = y[i] = 0;

   for( i=0; i<30000; i++ )                        // Modelagem Off-Line
    {
        yn = prnseq;
        value = 0;
        en = input_sample();                                // Lê entrada mic erro
        for( k=taps-1; k>0; k-- )                           // Calcula rn
            y[k] = y[k-1];
        y[0] = yn;
        rn = 0;
        for( k=0; k<taps; k++ )
            rn += c[k] * y[k];

        e_n = en - rn;                                      // Calcula erro

        for( k=0; k<taps; k++ )                             // Atualiza coeficientes
            c[k] = c[k] + ( mu * e_n * y[k] );
    }

while(1)
{
// Fase On-line:

for( i=0; i<60000; i++ )                             // Controle Aivo
    {
        AIC23_data.uint = input_sample();           // lê 32 bits (16 para cada canal)
        refe =(AIC23_data.channel[Esq]);            // separa canal esquerdo
        erro = (AIC23_data.channel[Dir]);           // separa canal direito
        for( k=taps-1; k>0; k-- )                   //
            x[k] = x[k-1];
        x[0] = refe;
        en = erro;
        for( k=taps-1; k>0; k-- )
            x1[k] = x1[k-1];
```

```
        x1[0] = 0;
        for( k=0; k<taps; k++ )
            x1[0] += c[k] * x[k];
        yn = 0;
        for( k=0; k<taps; k++ )
            yn += w[k] * x[k];
        output_sample(yn);
        e_n = en + rn;
        for( k=0; k<taps; k++ )
            w[k] = w[k] - ( mu * e_n * x1[k] );

    }

return;
}
```

Apêndice B

B.1 – Esquema do amplificador para os microfones de referência e erro

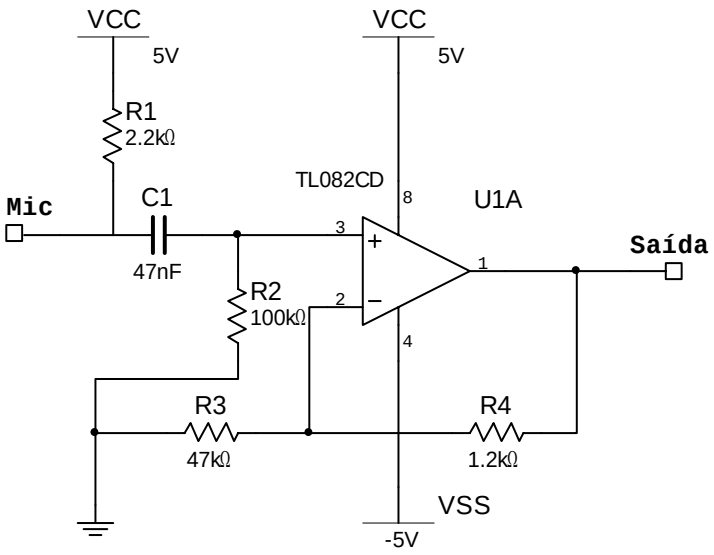


Fig. B.1 – Amplificador para microfones de eletreto (G≈40 , 1+[47/1.2])

B.2 – Esquema do amplificador de potência para os alto-falantes de ruído e cancelamento

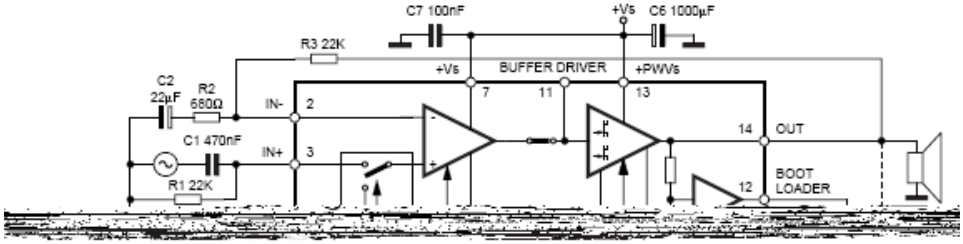


Fig. B.2 – Amplificador de potência básico utilizando o TDA 7293

(Copyright da STMicroelectronics)

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo