CANCELAMENTO ADAPTATIVO DE ECO EM SISTEMAS EMBARCADOS

Andrew Buch Sampaio

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia Elétrica, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia Elétrica.

Orientador: Sergio Lima Netto

Rio de Janeiro
Junho de 2013

CANCELAMENTO ADAPTATIVO DE ECO EM SISTEMAS EMBARCADOS

Andrew Buch Sampaio

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA ELÉTRICA.

Examinada por:

_____
Prof. Sergio Lima Netto, Ph.D.

_____
Prof. Marcello Luiz Rodrigues de Campos, Ph.D.

_____
Prof. José Antonio Apolinário Junior, D.Sc.

RIO DE JANEIRO, RJ - BRASIL
JUNHO DE 2013

Sampaio, Andrew Buch

Cancelamento Adaptativo de Eco em Sistemas Embarcados/ Andrew Buch Sampaio. – Rio de Janeiro: UFRJ/COPPE, 2013.

XV, 106 p.: il.; 29,7 cm.

Orientador: Sergio Lima Netto

Dissertação – UFRJ/ COPPE/ Programa de Engenharia Elétrica, 2013.

Referências Bibliográficas: p. 104-106.

1. Cancelamento de Eco. 2. Cancelamento de Eco Elétrico. 3. Filtros Adaptativos. 4. Plataformas Embarcadas. I. Netto, Sergio Lima. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Elétrica. III. Título.

Agradecimentos

Agradeço e dedico este trabalho aos meus pais Carlos Frederico e Lilian Cristina, por todo o esforço realizado ao longo de suas vidas para me proporcionar uma educação digna, pelo inestimável amor e carinho, por sempre me apoiarem, incentivando a prosseguir com meus estudos. Dedico também aos meus irmãos Allan e Amanda, pelo companheirismo e exemplo de dedicação.

Dedico e agradeço à minha esposa Larissa, fonte de todas as minhas forças, pelo carinho, compreensão e dedicação em todos os momentos. Agradeço pelo exemplo de ser humano, que instiga a evolução daqueles que a rodeiam.

Agradeço ao meu orientador Prof. Sergio Lima Netto, por acreditar em meu potencial e dedicar incontáveis horas ao auxílio da produção deste trabalho. Agradeço pela amizade, pela paciência, pelos valiosos conselhos e pela experiência transmitida ao longo do desenvolvimento desta dissertação.

Agradeço aos amigos do trabalho que contribuíram direta ou indiretamente para a realização desta tese, bem como aos amigos do Laboratório de Processamento de Sinais, em especial Amaro Azevedo de Lima, por colaborar com valiosos dados técnicos.

Por fim, agradeço a Deus, por tornar possível esta realização.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M. Sc.)

CANCELAMENTO ADAPTATIVO DE ECO EM SISTEMAS EMBARCADOS

Andrew Buch Sampaio

Junho/2013

Orientador: Sergio Lima Netto

Programa: Engenharia Elétrica

O problema do cancelamento de eco é amplamente conhecido e suas possíveis soluções são discutidas das mais variadas formas na vasta literatura técnica especializada. Porém, a literatura normalmente se concentra nos aspectos teóricos do cancelamento de eco, deixando os aspectos práticos em segundo plano.

Esta dissertação tem como objetivo o desenvolvimento de canceladores de eco para plataformas embarcadas, onde sérias restrições numéricas e computacionais devem ser levadas em conta na implementação dos filtros adaptativos.

Desta forma, inicialmente os algoritmos da literatura são testados e comparados em ambiente livre de restrições, visando escolher o mais adequado para uma implementação real. Em seguida, o algoritmo escolhido é alterado para operar em ambientes restritivos, como processadores de 16 bits com aritmética de ponto-fixo, e seu desempenho é avaliado por meio de simulações em um computador pessoal. Então, o algoritmo operando em ponto-fixo é implementado em uma plataforma embarcada real, abordando todos os detalhes práticos necessários à correta operação do filtro adaptativo.

Assim, este trabalho propõe o algoritmo *Switched Normalized Least Mean Square* (S-NLMS), que possui uma etapa de pré-processamento de dados que realiza o alinhamento dinâmico das amostras e o protege contra perdas de quadros de áudio. Seu desempenho superior em relação ao NLMS tradicional é comprovado por meio de experimentos na plataforma real.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

ADAPTIVE ECHO CANCELLATION IN EMBEDDED SYSTEMS

Andrew Buch Sampaio

June/2013

Advisor: Sergio Lima Netto

Department: Electrical Engineering

The echo cancellation problem is widely known, and its solutions are vastly discussed on the specialized literature. However, the available literature concentrates mostly in the theoretical aspects of the echo cancellation, leaving the practical aspects behind.

This work focuses on developing echo cancellers for embedded platforms, in which several numerical and computational constraints are present and must be dealt with when implementing adaptive filters.

Thus, initially the literature algorithms are tested and compared in a restriction free environment, in order to choose the most suitable for implementation on a real system. After that, the chosen algorithm is changed in order to operate on restricted environments, such as fixed-point 16 bits processors, and its performance is assessed by simulations on a personal computer. Finally, the fixed-point algorithm is implemented on a real embedded platform, explaining all the practical enhancements required to the adaptative filter's correct operation.

This work proposes the *Switched Normalized Least Mean Square* (S-NLMS) algorithm, which applies an extra data pre-processing layer, in order to dynamically align the audio samples and protect the filter's coefficients against audio packet losses. Its superior performance compared to the traditional NLMS algorithm is proven by experiments conducted on the real platform.

Conteúdo

Índice de Figuras

Índice de Tabelas

Capítulo 1 – Introdução

1.1 - Motivação

O problema do cancelamento de eco é amplamente conhecido e suas possíveis soluções são discutidas das mais variadas formas na vasta literatura técnica especializada. Porém, a literatura normalmente se concentra nos aspectos teóricos do cancelamento de eco, deixando os aspectos práticos em segundo plano.

Esta dissertação tem como objetivo o desenvolvimento de canceladores de eco para plataformas embarcadas, onde sérias restrições numéricas e computacionais devem ser levadas em conta na implementação dos filtros adaptativos.

Em aplicações reais, problemas de natureza prática se fazem presentes e devem ser contornados, como, por exemplo, o sincronismo temporal exigido pelos algoritmos da literatura e falhas de transmissão dos pacotes de áudio entre os componentes do sistema. Em aplicações cujo processador é utilizado tanto para a interface homem-máquina quanto para o processamento de sinais, a natureza *multi-thread* e a latência entre os processos também passam a ser empecilhos. Tais problemas podem deteriorar o desempenho do cancelador de eco e, em casos extremos, inviabilizar a construção de um sistema de comunicação de qualidade.

Desta forma, este trabalho tem como objetivo a construção de um cancelador de eco em uma plataforma embarcada real, descrevendo todos os desafios e soluções encontrados ao longo do processo, bem como o ferramental necessário para seu desenvolvimento.

1.2 – Resultados Alcançados

Tendo em conta a natureza crítica do sistema embarcado real, esta dissertação propõe um novo algoritmo chamado S-NLMS (Switched *Normalized Least Mean Square*), que adiciona uma camada de pré-processamento dos dados antes da filtragem adaptativa responsável por proteger os coeficientes do filtro contra falhas de transmissão e assincronismos entre as amostras adquiridas e recebidas.

O desempenho do S-NLMS em relação à falha de transmissões foi testado no sistema embarcado real, onde, dependendo da taxa de perda de pacotes da rede, uma melhora de

53% a 700% na avaliação perceptual do áudio pôde ser experimentada em relação ao algoritmo NLMS .

A operação dos algoritmos S-NLMS e NLMS em redes sem perdas também foi testada. Empregando o alinhamento dinâmico das amostras de um mesmo quadro de áudio, o algoritmo S-NLMS proposto neste trabalho forneceu um desempenho aproximadamente 80% superior na avaliação perceptual em relação ao NLMS.

1.3 – Organização da Dissertação

O Capítulo 2 traz os conceitos básicos da filtragem adaptativa, descrevendo os principais algoritmos hoje utilizados, e o sistema no qual os algoritmos desenvolvidos nesta dissertação serão implementados.

O Capítulo 3 apresenta o desenvolvimento de canceladores de eco sem restrições, ou seja, o desenvolvimento de algoritmos que se valem de um ambiente composto por unidades lógicas somadoras e multipicadoras, alto poder de processamento e grandes quantidades de memória. Apresenta também um estudo de otimização de parâmetros, onde ambas as características qualitativas e quantitativas são levadas em conta, a fim de se realizar uma comparação justa entre os algoritmos desenvolvidos. Neste cenário, será escolhido o algoritmo que obtiver o melhor desempenho para que este seja modificado para operar em ambientes restritivos como o sistema objetivo.

O Capítulo 4 apresenta o desenvolvimento de um cancelador de eco com restrições, ou seja, um cancelador de eco para sistemas embarcados. Para isto, descreve as restrições impostas por este tipo de sistema e as ferramentas disponíveis para simulá-las. Ao fim do capítulo, o algoritmo escolhido no capítulo anterior é alterado para operar em ponto-fixo, e os resultados obtidos de simulações com amostras de voz são apresentados.

O Capítulo 5 apresenta em detalhes o cancelamento de eco em um sistema real. Explicitando as características do processador utilizado, as ferramentas necessárias para o desenvolvimento em um computador pessoal, bem como as dificuldades encontradas na implementação dos filtros em um sistema real. Ao final, um algoritmo para cancelamento de eco é desenvolvido e seu desempenho na plataforma alvo é apresentado.

O Capítulo 6 conclui esta dissertação resumindo os assuntos abordados e as diferentes soluções existentes, enfatizando os resultados obtidos nos desenvolvimentos do cancelador de eco para plataformas embarcadas.

Capítulo 2 - Conceitos Básicos de Cancelamento de Eco

2.1 - Introdução:

Eco é a repetição de uma forma de onda decorrente da reflexão gerada pela mudança da característica do meio no qual a onda se propaga [1].

Apesar de ser empregado de forma positiva em sistemas de sonares e radares, nas comunicações o eco pode degradar a qualidade do serviço. Por este motivo, os canceladores de eco são parte importante dos sistemas de comunicação.

Os efeitos percebidos do eco dependem basicamente do atraso entre as ondas incidente e refletida, o nível de sinal da onda refletida, e o número de caminhos pelos quais a onda é refletida.

Em particular, o atraso pode ter um impacto considerável na qualidade da conversação. Níveis elevados de atraso podem levar a problemas de interação entre os participantes.

Os efeitos do atraso são dependentes do modelo de conversação em curso [2]. Se o fluxo de conversação for predominante em apenas uma direção, então o efeito percebido pelo atraso pode ser desprezado. Se a conversação for altamente interativa, então mesmo baixos níveis de atraso podem ser problemáticos.

A Figura 2.1 mostra a relação entre o modelo de conversação e o efeito percebido pelo atraso (MOS – m*ean opinion score*) em três situações. Se a conversação for pouco interativa (fluxo em apenas uma direção) o atraso não causa nenhum impacto negativo. Para uma conversação moderadamente interativa existe um impacto considerável para atrasos superiores a 300 ms. Para uma conversação altamente interativa notam-se impactos significativos para atrasos já a partir de 200 ms.

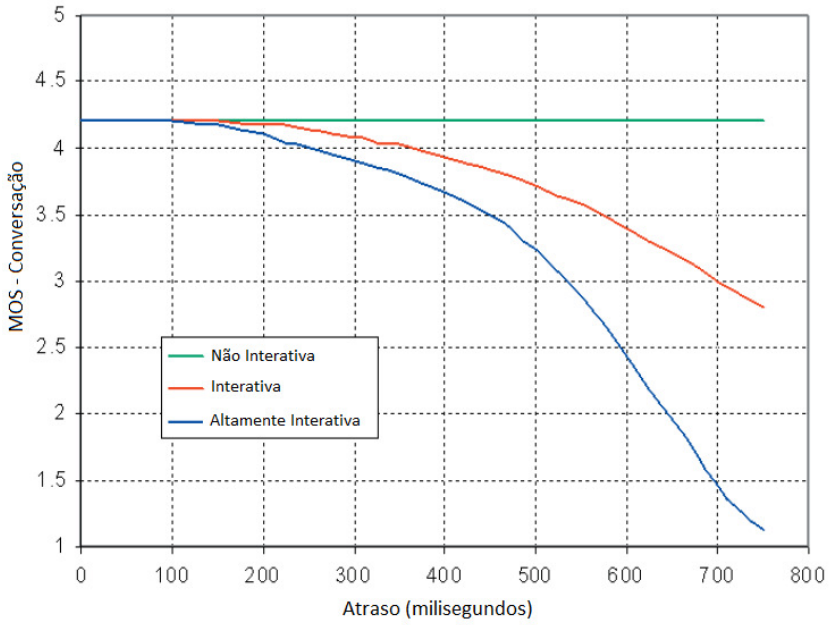

Figura 2.1- Relação entre o atraso e a qualidade da conversação para três modelos de interação [2].

Os efeitos do eco também são muito dependentes do atraso. Se o atraso do sistema for muito pequeno, um elevado nível de eco pode ser tolerado. A Figura abaixo mostra que com 10 ms de atraso, um elevado nível de eco (25 dB) pode ser tolerado enquanto em um sistema com atraso de 100 ms o nível de eco tolerável é de apenas 46 dB.

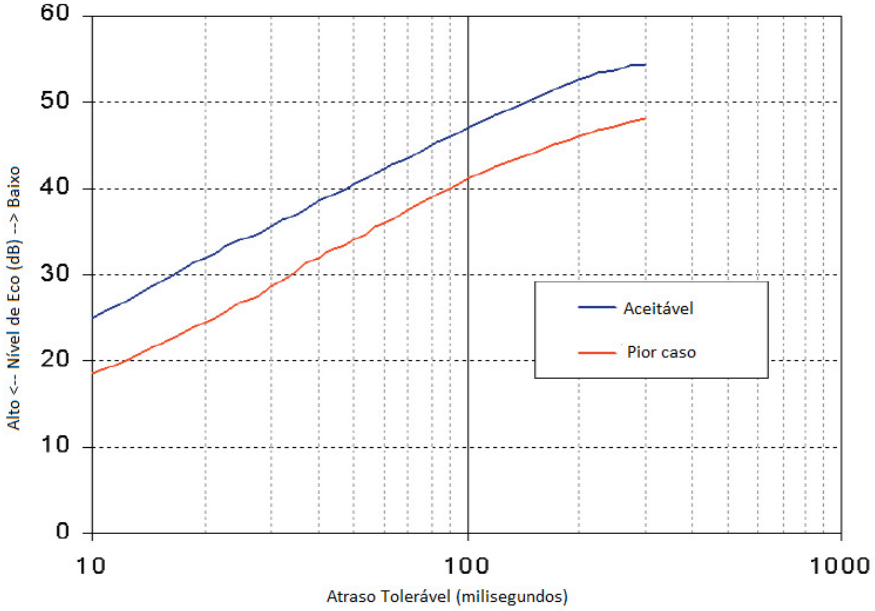

Figura 2.2 - Relação entre atraso e nível de eco (TELR) [2].

Desta forma, o presente capítulo tem como objetivo abordar as principais soluções existentes na literatura especializada para o problema do cancelamento de eco, explicando de forma detalhada a teoria envolvida em cada algoritmo utilizado.

Assim, as seções 2.2 e 2.3 do presente capítulo, abordam o problema do eco elétrico e acústico, respectivamente, explicando as suas naturezas e a forma como são criados nos sistemas de comunicações atuais. A Seção 2.4 aborda o problema do cancelamento de eco em sistemas de comunicação, apresentando as principais soluções empregadas. A Seção 2.5 apresenta as soluções numéricas para o problema do cancelamento de eco, descrevendo matematicamente as estratégias de cancelamento de eco mais utilizadas na atualidade. A Seção 2.6 descreve o sistema real onde os filtros desenvolvidos neste trabalho serão testados, apresentando suas particularidades de transmissão e operação. A Seção 2.7 conclui o capítulo, sumarizando os assuntos nele abordados.

2.2 - Eco de Linha (Elétrico)

O eco de linha é resultado de um descasamento de impedância nas interfaces híbridas, circuitos onde o par telefônico *full-duplex* do assinante é conectado às linhas de transmissão por meio de dois pares *half-duplex* (Tx e Rx).

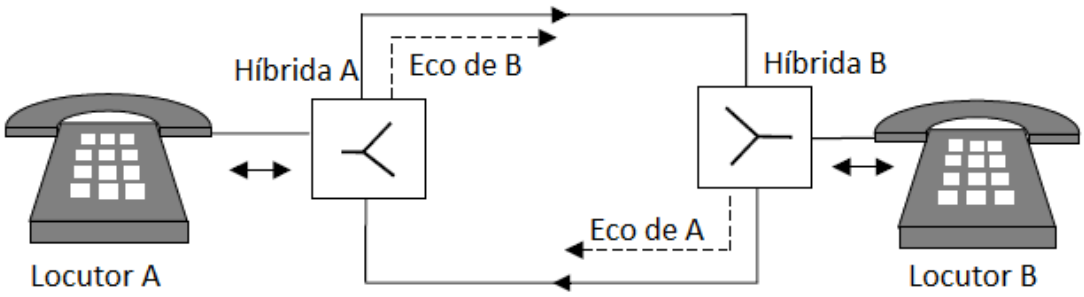

Figura 2.3 - Versão simplificada de um sistema de transmissão de longa distância, com os sinais de eco retornando de cada interface híbrida

Idealmente, no esquema representado pela Figura 2.3, a interface híbrida B receberia o sinal do locutor A pelo par *half-duplex* Rx sem permitir que qualquer energia deste retornasse ao par *half-duplex* Tx. Na prática não se consegue este objetivo, porque, no caso de interfaces passivas (Figura 2.4), a variação no comprimento do loop do assinante, as características individuais dos aparelhos e a impedância da linha causam o cancelamento imperfeito do sinal recebido. Como consequência, o locutor A recebe inevitavelmente uma cópia atrasada do seu próprio sinal de voz.

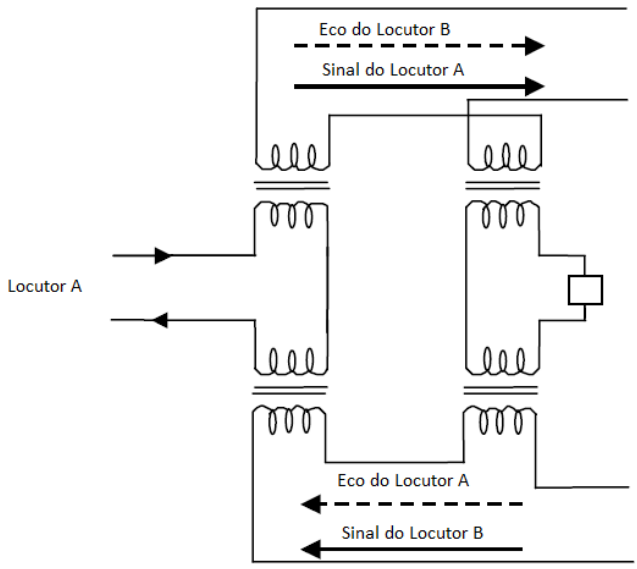

Figura 2.4 - Interface Híbrida Passiva: o eco é gerado devido ao acoplamento existente entre os transformadores.

No caso de interfaces ativas (Figura 2.5), a resposta em frequência dos componentes envolvidos faz com que o sinal recebido pelo par *half-duplex* Rx tenha um desvio de fase diferente de zero em relação ao sinal transmitido pelo par *full-duplex* do locutor, o que gera um descasamento entre as ondas e um consequente sinal de eco residual na linha de transmissão *half-duplex* Tx.

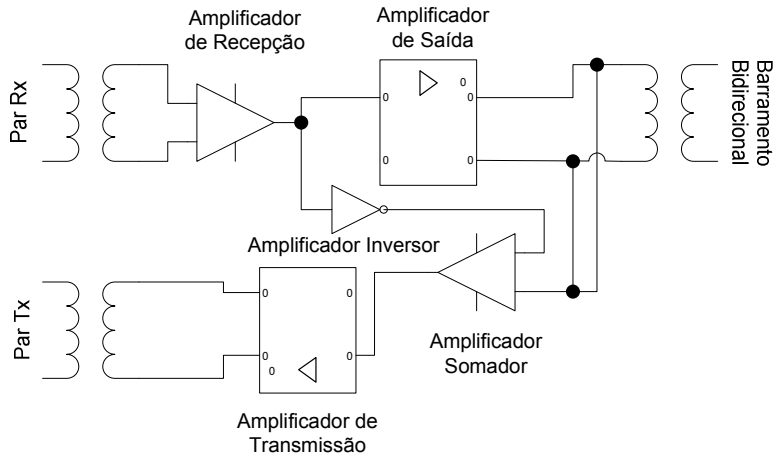

Figura 2.5 - Interface Híbrida Ativa: o eco é gerado devido aos desvios de fase sofridos pelos sinais no pares Rx e bidirecional, que fazem com que o cancelamento executado pelo amplificador somador não seja perfeito.

2.3 - Eco Acústico

Comumente encontrado em aparelhos celulares, sistemas de viva-voz e em sistemas de teleconferência, o eco acústico é resultado da existência de algum caminho de realimentação entre o alto falante e o microfone [1] (Figura 2.6).

Estes sistemas requerem ganhos muito superiores para manter os níveis de sinal enviados para ou recebidos da rede telefônica. O ganho adicional é utilizado para compensar as perdas associadas com o aumento da distância entre o usuário e alto falante e entre o usuário e o microfone. Este ganho adicional, associado ao fato de que a maioria dos fabricantes de sistemas de viva-voz posiciona o microfone e o alto falante na mesma carcaça, geralmente causa instabilidade eletroacústica.

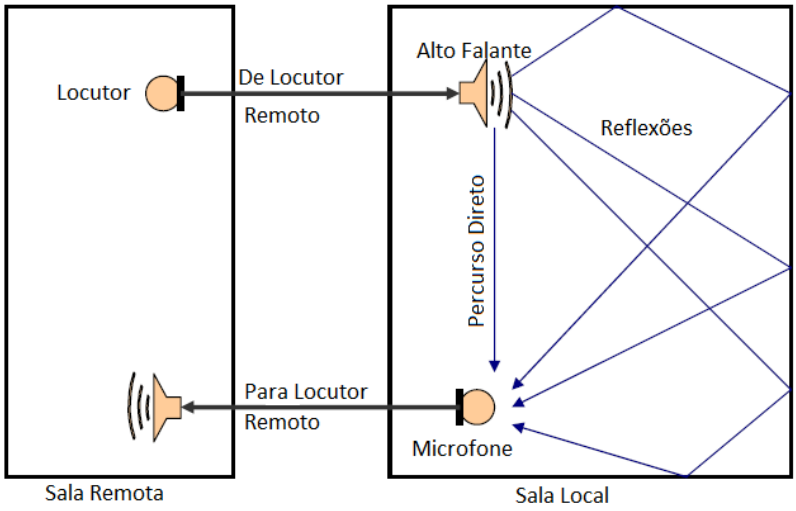

Figura 2.6 - Eco Acústico e sua natureza crítica: várias reflexões podem ser capturadas pelo microfone da sala local.

2.4 - Cancelamento de Eco

O cancelamento de eco foi desenvolvido no início da década de 1960 pela AT&T Bell Labs e mais tarde pela COMSAT TeleSystems [1]. Os primeiros sistemas canceladores de eco foram experimentalmente implementados em redes de comunicação via satélite com o objetivo de demonstrar o desempenho da rede para ligações de longa distância.

Atualmente, o cancelamento de eco baseia-se em sistema adaptativos, que consistem em gerar uma estimativa da resposta ao impulso do canal para sintetizar uma cópia do eco e subtraí-la do sinal retornado [3, 4].

O esquema básico de um sistema de cancelamento de eco de linha pode ser verificado na Figura 2.7 abaixo:

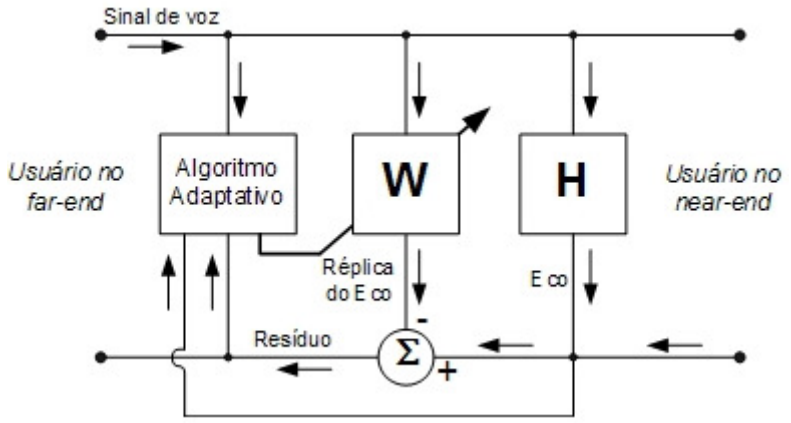

Figura 2.7 - Esquema básico de cancelamento de eco de linha: um filtro adaptativo é utilizado para modelar a resposta ao impulso do percurso de eco do sistema.

A ilustração leva em conta apenas o eco gerado pelo usuário remoto (*far-end)*. O bloco **H** representa a resposta ao impulso da interface híbrida do caminho de eco do sistema. Conforme explicitado anteriormente, o cancelador de eco, representado por um filtro digital **W** e associado a um algoritmo adaptativo, tem por objetivo alterar em tempo real a resposta do filtro digital, de modo a estimar o sistema **H**.

Na Figura 2.8, pode-se observar o esquema básico de um sistema de cancelamento de eco acústico:

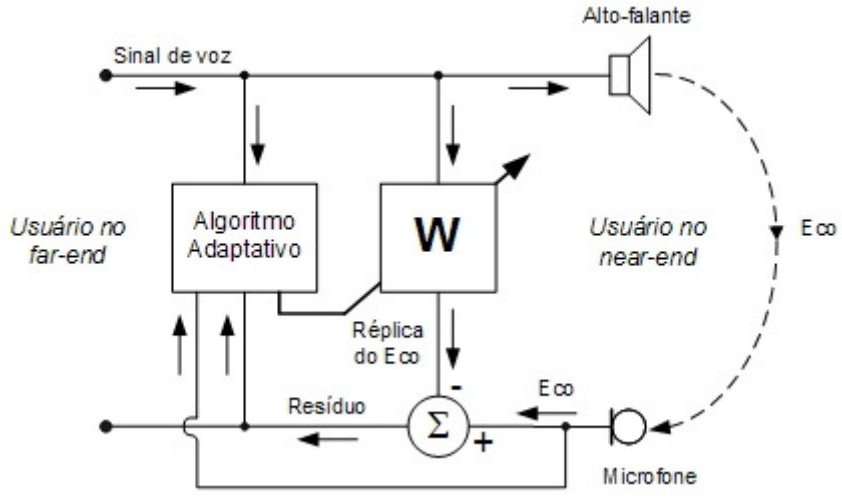

Figura 2.8 - Esquema básico de cancelamento de eco acústico: um filtro adaptativo é utilizado para modelar a resposta ao impulso do percurso de eco do sistema.

Neste caso, o sinal de voz do usuário remoto (*far-end)* é reproduzido pelo alto-faltante do telefone e percorre através do ar um (ou vários) caminho(s) até o microfone, onde é captado. Novamente, o algoritmo de filtragem adaptativa busca ajustar a resposta do filtro **W**, de modo a obter um modelo do caminho acústico percorrido pelo sinal de voz.

2.5 – Soluções Numéricas

2.5.1 - Predição Linear

A predição linear é um mecanismo simples e bastante eficaz para a estimação de um elemento de uma série temporal tomando como referência apenas suas N amostras passadas. Em outras palavras, dada a série temporal $u(n), u(n-1), u(n-2), \dots, u(n-N)$, a meta é computar $\hat{u}(n)$ por combinação linear dos demais N termos, ou equivalentemente, desejamos calcular $\hat{u}(n|u(n-1), u(n-2), \dots, u(n-N))$, uma estimativa de $u(n)$. Uma representação gráfica deste procedimento pode ser vista no diagrama abaixo:

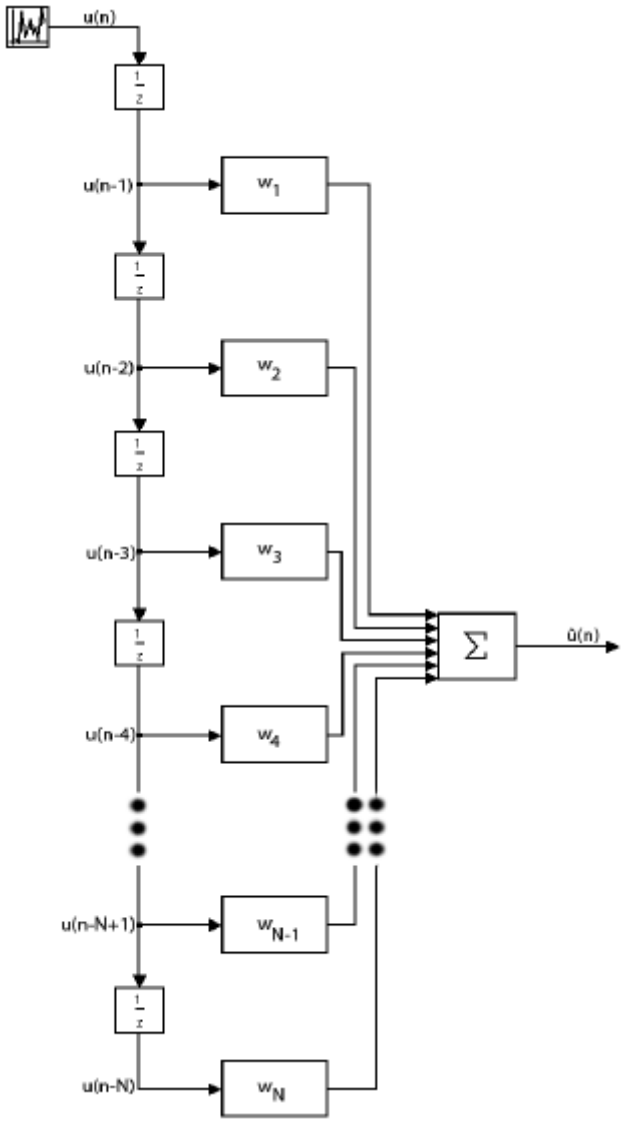

Figura 2.9 - Preditor Linear: combinação linear das amostras para diferentes atrasos visando prever o valor futuro.

Denotando-se o conjunto $u(n-1), u(n-2), \dots, u(n-N)$ por $\mho$, tem-se:

$$\hat{u}(n|\mathcal{U}) = \sum_{j=1}^{N} w_j \,.\, u(n-j). \tag{2.1}$$

O erro associado à predição é $e(n) = u(n) - \hat{u}(n|\mathcal{U})$ e, para minimizar este erro, utiliza-se usualmente o MSE (m*ean squared error*) como métrica:

$$\varepsilon = E[e^2(n)] = E\left[\left(u(n) - \hat{u}(n)\right)^2\right]. \tag{2.2}$$

Percebe-se que o MSE é uma função quadrática dos coeficientes do preditor linear, então, a fim de encontrar o vetor de coeficientes ótimos $\boldsymbol{w}^*$, faz-se:

$$\frac{\partial \varepsilon}{\partial w_i} = \frac{\partial E[e^2(n)]}{\partial w_i} = E\left[\frac{\partial e^2(n)}{\partial w_i}\right] = E\left[2e(n)\frac{\partial e(n)}{\partial w_i}\right] = -2E[e(n)u(n-i)]. \tag{2.3}$$

Substituindo $e(n) = u(n) - \sum_{i=1}^{N} w_i\, u(n-i)$ na equação acima, obtém-se

$$\frac{\partial \varepsilon}{\partial w_i} = -2E\left[\left(u(n) - \sum_{j=1}^{N} w_j \,.\, u(n-j)\right)u(n-j)\right] \tag{2.4}$$

$$= -2\left(E[u(n)u(n-i)] - \sum_{j=1}^{N} w_j \,.\, E[u(n-j)u(n-i)]\right) \tag{2.5}$$

$$= -2R_U(i) + 2\sum_{j=1}^{N} w_j . R_U(i-j)\,, \tag{2.6}$$

assumindo que o processo U é estacionário no sentido amplo (WSS), ou seja, sua média não depende do tempo e sua autocorrelação depende apenas do intervalo de observação, mas não do instante de início [5].

Na forma matricial, tem-se

$$\begin{bmatrix} R_U(0) & R_U(-1) & \cdots & R_U(1-N) \\ R_U(1) & R_U(0) & \ldots & R_U(2-N) \\ \vdots & \vdots & \ddots & \vdots \\ R_U(N-1) & R_U(N-2) & \cdots & R_U(0) \end{bmatrix} \begin{bmatrix} w_1^* \\ w_2^* \\ \vdots \\ w_N^* \end{bmatrix} = \begin{bmatrix} R_U(1) \\ R_U(2) \\ \vdots \\ R_U(N) \end{bmatrix}, \tag{2.7}$$

que é conhecida como equação de Wiener-Hopf. De forma sucinta:

$$\boldsymbol{R_U w^* = p_U}. \tag{2.8}$$

Onde $\boldsymbol{R_U}$ é a matriz de autocorrelação para o processo U e $\boldsymbol{p_U}$ é o vetor do lado direito da equação matricial anterior.

Assumindo que a matriz $\boldsymbol{R_U}$ possui inversa, o vetor de coeficientes ótimo pode ser facilmente calculado por:

$$\boldsymbol{w}^* = \boldsymbol{R_U}^{-1}\boldsymbol{p_U}. \tag{2.9}$$

A matriz hessiana do MSE pode ser facilmente calculada a partir da Equação (2.6). Para isto, basta derivar novamente em relação a cada w_i:

$$\frac{\partial^2 \varepsilon}{\partial w_i \partial w_k} = 2R_U(i-k). \tag{2.10}$$

De modo que matriz hessiana dada por

$$\boldsymbol{H} = \left[\frac{\partial^2 \varepsilon}{\partial w_i \partial w_k}\right]_{i,k} = [2R_U(i-k)]_{i,k} = 2\boldsymbol{R}_U, \tag{2.11}$$

que é geralmente positiva definida. Isto indica que o ponto estacionário $\boldsymbol{w}^*$está associado com um mínimo global.

Para o caso particular de predição por dois coeficientes, tem-se:

$$\begin{aligned}\varepsilon = E[u(n)^2 - 2w_1u(n)u(n-1) + w_1^2u^2(n-1) - 2w_2u(n)u(n-2) \\ + w_2^2u^2(n-2) + 2w_1w_2u(n-1)u(n-2)].\end{aligned} \tag{2.12}$$

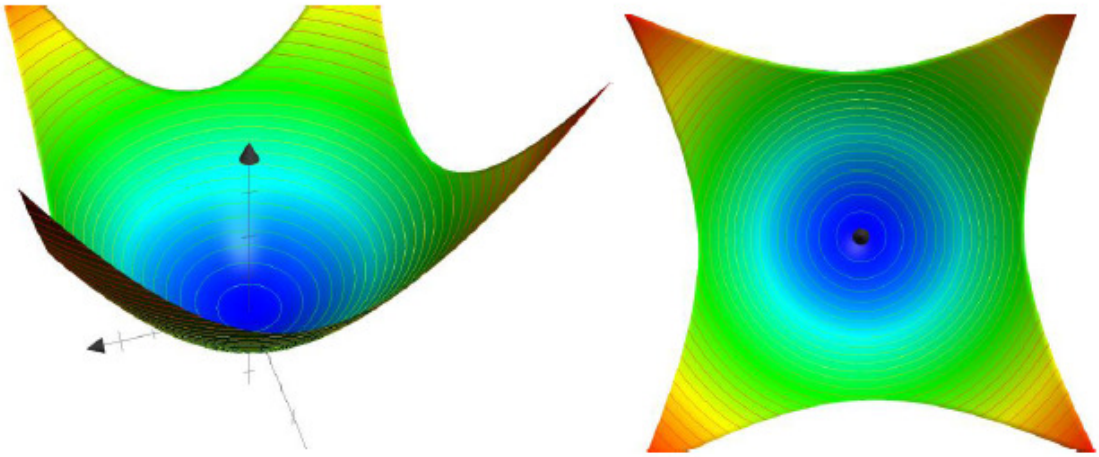

Figura 2.10 - Superfície de erro do preditor linear: natureza quadrática com mínimo global.

Como se pode inferir pela superfície de erro do preditor linear de dois coeficientes acima, a função erro apresenta apenas um mínimo.

2.5.2 - Steepest Descent

O método da descida mais íngreme (steepest descent) é um mecanismo iterativo fundamentado em análise de gradiente que nos permite criar um preditor linear que rastreie as variações estatísticas de uma entrada $u(n)$ sem a necessidade de calcular a solução da equação (2.9) a cada iteração. Desta forma, consegue-se a cada iteração reduzir o erro médio quadrático (MSE) do preditor linear sem que seja preciso recalcular diversas vezes a inversa da matriz de correlações do sinal de entrada.

Assim, dada uma função custo $C(w)$, o objetivo é minimizá-la a cada iteração do processo, ou seja, para cada iteração tem-se $C\big(w(n+1)\big) < C(w(n))$. Ao final do processo, caso o filtro convirja, obter-se-á $C(w_0) < C(w)$ para todo w.

Para alcançar esta meta, o método *steepest descent* ajusta iterativamente o sistema na direção da descida mais íngreme de $C(w)$, isto é, no sentido contrário àquele apontado por seu vetor gradiente $\nabla C(w)$. Matematicamente, tem-se:

$$w(n+1) = w(n) - \mu \nabla C(w), \tag{2.13}$$

onde μ é uma variável que controla o tamanho do passo do algoritmo.

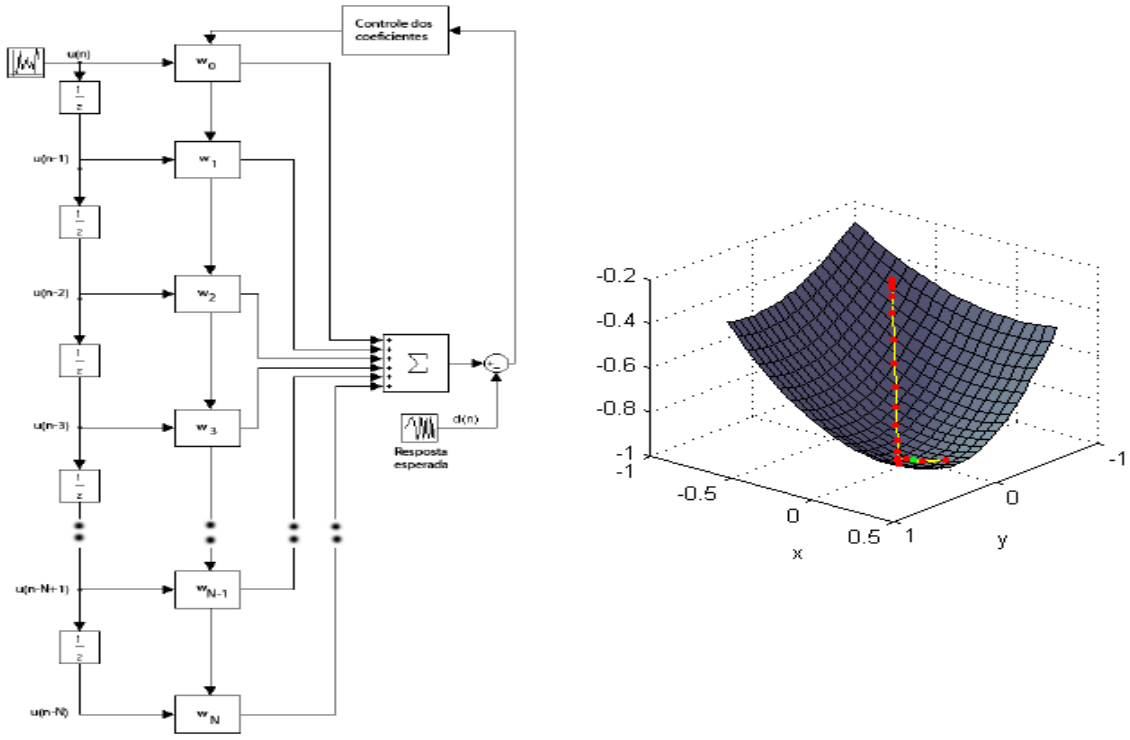

Figura 2.11 - Método Steepest Descent, que utiliza o preditor linear para prever a amostra futura e a compara com a amostra obtida de forma a realiamentar o filtro, visando percorrer o caminho de maior descida na superfície de erro.

2.5.3 - Filtro de Wiener

O problema da predição linear pode ser generalizado para o caso onde se deseja caracterizar a relação estocástica entre dois processos representados pelas realizações $y(n)$ e $x(n)$. O modelo linear resultante é comumente chamado de Filtro de Wiener [5]. Neste contexto, assume-se que uma estimativa $\hat{y}(n)$ de $y(n)$ pode ser determinada a partir da combinação linear das amostras de $x(n)$. Ou seja:

$$\hat{y}(n) = \sum_{j=0}^{M} w_i \,.\, x(n-i), \tag{2.14}$$

onde w_i para $i = 0,1,2,\ldots,M$são os coeficientes do filtro de Wiener e M é a ordem do filtro. O erro neste caso é dado por $e(n) = y(n) - \hat{y}(n)$ e o MSE associado é dado por $\varepsilon = E[e^2(n)]$.

De forma similar ao caso da predição linear, tem-se:

$$\frac{\partial \varepsilon}{\partial w_i} = \frac{\partial E[e^2(n)]}{\partial w_i} = E\left[\frac{\partial e^2(n)}{\partial w_i}\right] = E\left[2e(n)\frac{\partial e(n)}{\partial w_i}\right] = -2E[e(n)x(n-i)] \tag{2.15}$$

$$\frac{\partial \varepsilon}{\partial w_i} = -2E\left[\left(y(n) - \sum_{j=0}^{M} w_j \,.\, x(n-j) \right) x(n-i)\right] \tag{2.16}$$

$$= -2\left(E[y(n)x(n-i)] - \sum_{j=1}^{M} w_j \,.\, E[x(n-j)x(n-i)] \right) \tag{2.17}$$

$$= -2p_{XY}(i) + 2\sum_{j=0}^{M} w_j . R_X(i-j), \tag{2.18}$$

onde $p_{XY}(i)$ é a correlação cruzada entre $y(n)$ e $x(n-i)$, assumindo que os processos são conjuntamente WSS.

Equivalentemente ao caso do preditor linear, a solução de Wiener poder ser calculada a partir da seguinte expressão:

$$\begin{bmatrix} R_X(0) & R_X(-1) & \cdots & R_X(-M) \\ R_X(1) & R_X(0) & \ldots & R_X(-M+1) \\ \vdots & \vdots & \ddots & \vdots \\ R_X(M) & R_X(M-1) & \cdots & R_X(0) \end{bmatrix} \begin{bmatrix} w_0^* \\ w_1^* \\ \vdots \\ w_M^* \end{bmatrix} = \begin{bmatrix} p_{XY}(0) \\ p_{XY}(1) \\ \vdots \\ p_{XY}(M) \end{bmatrix} \tag{2.19}$$

$$\boldsymbol{w}^* = \boldsymbol{R_X}^{-1}\boldsymbol{p}_{YX}. \tag{2.20}$$

O filtro de Wiener é inadequado para situações em que não exista estacionaridade ou quando o conhecimento a priori da estatística dos sinais $x(n)$ e $e(n)$ a serem processados é inviável. Nestes casos, o filtro de Wiener poderá ser não realizável ou não ter desempenho ótimo. Uma solução para tal situação pode ser realizar primeiramente uma estimativa da estatística do sinal a ser processado e, em seguida, aplicá-la no cálculo do filtro. Este procedimento, porém, não atende as necessidades de aplicações em tempo real [6].

2.5.4 - Filtros Adaptativos

Em ambientes onde não se tem conhecimento a priori das características estatísticas dos sinais de entrada, os filtros adaptativos se apresentam como uma excelente solução. Estes se baseiam em algum algoritmo de otimização para ajustar seus coeficientes dinamicamente com o intuito de minimizar em tempo real uma dada função custo.

O processo evolui da seguinte forma:

1. Inicializam-se os coeficientes do filtro com valores apropriados (zero, caso haja desconhecimento total do ambiente);
2. Se o ambiente for estacionário, os coeficientes do filtro convergirão iterativamente para os coeficientes do filtro de Wiener;
3. Caso contrário, o algoritmo de otimização possibilita ao filtro acompanhar as variações estatísticas ao longo do tempo, desde que essas variações sejam suficientemente lentas.

2.5.4.1 - Algoritmo LMS (Least Mean Square)

Baseado no método do gradiente descendente explicitado anteriormente, o algoritmo LMS utiliza o valor instantâneo de $e^2(n)$ como estimativa da função custo calculada por $E[e^2(n)]$, ou seja,

$$\nabla_{\mathrm{w}}\mathrm{E}[\mathrm{e}^2(n)] \approx \nabla_{\mathrm{w}}\mathrm{e}^2(n) = 2e(n)\frac{\partial e(n)}{\partial \boldsymbol{w}}. \tag{2.21}$$

Sendo $e(n) = d(n) - y(n)$ e $y(n) = \boldsymbol{w}^T\boldsymbol{x}(n)$, então

$$\frac{\partial e(n)}{\partial \boldsymbol{w}} = \frac{\partial[d(n) - \boldsymbol{w}^T\boldsymbol{x}(n)]}{\partial \boldsymbol{w}} = -\boldsymbol{x}(n). \tag{2.22}$$

Logo,

$$\nabla_{\mathrm{w}}\mathrm{e}^2(n) = 2e(n)\frac{\partial e(n)}{\partial \boldsymbol{w}} = -2e(n)\boldsymbol{x}(n). \tag{2.23}$$

Assim, a equação de atualização dos coeficientes explicitada pelo método *steepest descent* se torna:

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

$$= w(n) - \mu\big(-2e(n)\boldsymbol{x}(n)\big) \tag{2.25}$$

$$= w(n) + \bar{\mu}e(n)\boldsymbol{x}(n), \tag{2.26}$$

onde a constante 2 foi incorporada ao fator $\bar{\mu}$ que controla o tamanho do passo do algoritmo.

2.5.4.2 - Algoritmo NLMS (Normalized LMS)

O algoritmo NLMS é uma implementação do algoritmo LMS que leva em conta as variações do sinal de entrada do filtro. Isto é conseguido utilizando um tamanho de passo variável normalizado pela energia do sinal de entrada do filtro, o que garante que o erro seja minimizado em magnitude. Sua lei de formação é

$$\mu(n) = \frac{\mu}{\boldsymbol{x^T}(n)\boldsymbol{x}(n) + \gamma} \tag{2.27}$$

$$e(n) = d(n) - \boldsymbol{w}^T(n+1)\boldsymbol{x}(n) \tag{2.28}$$

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \mu(n)e(n)\boldsymbol{x}(n) \tag{2.29}$$

onde o parâmetro de regularização γ foi introduzido para evitar valores muito elevados para o passo de adaptação quando a energia do sinal de entrada é pequena.

2.5.4.3 - Algoritmo PNLMS (Proportionate NLMS)

O algoritmo PNLMS aplica um tamanho de passo adaptativo e individual para cada coeficiente do filtro [7]. Os tamanhos dos passos são calculados a partir da última estimativa dos coeficientes do filtro de forma a garantir que os coeficientes grandes recebam elevados tamanhos de passo. Desta maneira, os coeficientes ativos são ajustados mais rápido do que os não ativos. O algoritmo PNLMS adapta os coeficientes segundo

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \frac{\mu}{\boldsymbol{x^T}(n)\boldsymbol{G}(n)\boldsymbol{x}(n) + \gamma}\boldsymbol{G}(n)\boldsymbol{x}(n)e(n), \tag{2.30}$$

onde:

$$\boldsymbol{G}(n) = diag\{g_0(n)g_1(n) \dots g_{N-1}(n)\}. \tag{2.31}$$

$$g_l(n) = \frac{\gamma_l}{\frac{1}{N}\sum_{i=0}^{N-1}\gamma_i(n-1)}, l = 0,1, \dots, N-1. \tag{2.32}$$

$$\gamma_l(n) = max\{\gamma_{min}, |w_l(n)|\}, l = 0,1, \dots, N-1. \tag{2.33}$$

$$\gamma_{min} = \varphi \max\{\delta, |w_0(n)|, \dots, |w_{N-1}(n)|\}. \tag{2.34}$$

Os parâmetros δ e φ são números positivos com valores típicos $\delta = 0.01$ e $\varphi = \frac{5}{N}$. O termo γ_{min} protege $w_l(n)$ de não ser adaptado quando este é muito menor que o maior coeficiente e δ regula a atualização quando todos os coeficientes são inicializados com zeros [6].

2.5.4.4 - Algoritmo NDR-LMS (Normalized Data-Reusing LMS)

Existem situações onde é possível reutilizar dados antigos para melhorar a convergência dos filtros adaptativos. Em ambientes onde o sinal de entrada é correlacionado, os algoritmos da família data-reusing são considerados uma boa alternativa para aumentar a velocidade de convergência. O preço a ser pago é o aumento do desajuste do algoritmo, onde a relação entre o aumento na velocidade de convergência e o aumento no desajuste pode ser controlada pelo tamanho do passo do algoritmo [6].

Sendo *L* o número de dados antigo a ser utilizado, o NDR-LMS atualiza os coeficientes do filtro como [8]

$$\boldsymbol{w_{i+1}}(n) = \boldsymbol{w}_i(n) + \frac{\mu}{\boldsymbol{x^T}(n-i)\boldsymbol{x}(n-i) + \gamma} e_i(n)\boldsymbol{x}(n-i), \tag{2.35}$$

onde $i = 0,1, \dots, L$ e

$$\boldsymbol{w}_0(n) = \boldsymbol{w}_L(n-1), \tag{2.36}$$

$$e(n) = d(n) - \boldsymbol{w}_i^T(n)\boldsymbol{x}(n), \tag{2.37}$$

$$\boldsymbol{w}_0(n+1) = \boldsymbol{w}_L(n). \tag{2.38}$$

2.5.4.5 - Algoritmo BNDR-LMS (Binormalized Data Reusing LMS)

Para obter uma convergência mais rápida comparada aos outros algoritmos LMS, o BNDR-LMS faz reuso de um par de dados antigo, além de combinar projeções ortogonais de duas direções de gradientes consecutivos e normalização [9]. A regra de atualização de coeficientes é

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \mu\left[\frac{\beta_1}{2}\boldsymbol{x}(n) + \frac{\beta_2}{2}\boldsymbol{x}(n-1)\right], \tag{2.39}$$

onde μ é o tamanho do passo e

$$\frac{\beta_1}{2} = \frac{e(n)\|\boldsymbol{x}(n-1)\|^2 - e(n-1)\boldsymbol{x}^T(n-1)\boldsymbol{x}(n)}{\|\boldsymbol{x}(n)\|^2\|\boldsymbol{x}(n-1)\|^2 - [\boldsymbol{x}^T(n)\boldsymbol{x}(n-1)]^2}, \tag{2.40}$$

$$\frac{\beta_2}{2} = \frac{e(n-1)\|\boldsymbol{x}(n)\|^2 - e(n)\boldsymbol{x}^T(n-1)\boldsymbol{x}(n)}{\|\boldsymbol{x}(n)\|^2\|\boldsymbol{x}(n-1)\|^2 - [\boldsymbol{x}^T(n)\boldsymbol{x}(n-1)]^2}, \tag{2.41}$$

e $e(n-1)$ é o erro a posteriori na iteração $(n-1)$:

$$e(n-1) = d(n-1) - \boldsymbol{w}^T(n)\boldsymbol{x}(n-1). \tag{2.42}$$

O algoritmo BNDR-LMS, assim como o algoritmo NLMS, faz parte da família *affine-projection* (APA), que generaliza a ideia de reutilização de L pares de dados para aumentar a convergência dos algoritmos adaptativos em situações em que o sinal de entrada é correlacionado. Enquanto o algoritmo NLMS é o caso particular do APA para L = 1, o BNDR-LMS é uma forma fechada do APA para L =2 [8].

2.5.4.6 - Algoritmo P-BNDR-LMS (Proportionate BNDR-LMS)

De forma similar ao explicitado no algoritmo PNLMS, o algoritmo P-BNDR-LMS combina as ideias de reutilização de dados antigos e de tamanhos de passo individuais para cada coeficiente do filtro adaptativo [6]. Sua regra de atualização é

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \mu\boldsymbol{G}(n)\left[\frac{\beta_1}{2}\boldsymbol{x}(n) + \frac{\beta_2}{2}\boldsymbol{x}(n-1)\right], \tag{2.43}$$

onde μ é o tamanho do passo, $\boldsymbol{G}(n)$ é definido como no algoritmo PNLMS e

$$\frac{\beta_1}{2} = \frac{e(n)\boldsymbol{x}^T(n-1)\boldsymbol{G}(n)\boldsymbol{x}(n-1) - e(n-1)\boldsymbol{x}^T(n-1)\boldsymbol{G}(n)\boldsymbol{x}(n)}{\boldsymbol{x}^T(n)\boldsymbol{G}(n)\boldsymbol{x}(n)\boldsymbol{x}^T(n-1)\boldsymbol{G}(n)\boldsymbol{x}(n-1) - [\boldsymbol{x}^T(n)\boldsymbol{G}(n)\boldsymbol{x}(n-1)]^2}. \tag{2.44}$$

$$\frac{\beta_2}{2} = \frac{e(n-1)\boldsymbol{x}^T(n)\boldsymbol{G}(n)\boldsymbol{x}(n) - e(n)\boldsymbol{x}^T(n-1)\boldsymbol{G}(n)\boldsymbol{x}(n)}{\boldsymbol{x}^T(n)\boldsymbol{G}(n)\boldsymbol{x}(n)\boldsymbol{x}^T(n-1)\boldsymbol{G}(n)\boldsymbol{x}(n-1) - [\boldsymbol{x}^T(n)\boldsymbol{G}(n)\boldsymbol{x}(n-1)]^2}. \tag{2.45}$$

2.6 - O Sistema Alvo

O propósito deste trabalho é desenvolver um filtro adaptativo para cancelamento de eco elétrico em um sistema de intercomunicação comercial. Devido à tecnologia empregada no sistema, o filtro deverá ser implementado em ponto-fixo em uma plataforma embarcada para o cancelamento do eco elétrico causado por descasamentos de fase dos sinais em uma interface híbrida ativa. No sistema em questão, os sinais são enviados via

rede *ethernet*, e os atrasos inerentes a esta tecnologia catalisam o efeito incômodo do eco.

Originalmente desenvolvido com circuitos analógicos, o sistema Setha 7000 é um sistema de intercomunicação e sinalização de emergência industrial centralizado não seletivo onde todas as estações de conversação estão em um barramento de áudio comum. Existem dois tipos de estação de conversação: a console de mesa (ECC Setha 7030), localizada na sala de controle (CIC), e a console de área (ECC Setha 7011), localizada na área de processo. Dependendo do tipo de área de processo a ser instalado, o sistema Setha 7000 está disponível em versões à prova de explosões, recomendada para áreas classificadas Ex (zonas 1 e 2, grupamento de gases IIB [10]), e à prova de tempo, para operação sob intempéries de clima, como, por exemplo, temperatura e umidade. Por ser um sistema dedicado, caso a instalação possua mais de uma área classificada de interesse, deve existir obrigatoriamente uma console de mesa para cada área a ser controlada.

O subsistema alvo deste trabalho é composto por três novos circuitos desenvolvidos a fim de atualizar o sistema Setha 7000 de forma a torná-lo seletivo. Basicamente, este subsistema transforma o sistema Setha 7000 em um sistema digital, onde a voz (sinal analógico) é capturada nos circuitos de borda, e, em seguida, é codificada e transmitida via IP (ethernet). A seguir, uma breve descrição de cada circuito do subsistema é apresentada:

- Console Setha 7030IP: Estação de Conversação e Chamada (ECC) digital projetada para áreas internas, tais como salas de comando. A voz trafega em IP (VoIP) codificada e trasmitida seguindo o padrão G711 (Sinal de voz amostrado 8kHz, comprimido utilizando A-Law ou μ-Law e transmitido em pacotes de 160 Bytes) [11].
- Interface Digital Setha 7058: Interface de conversão multicanal de sinais digitais (VoIP) para sinais analógicos não balanceados half-duplex (Tx e Rx). A exemplo da Console Setha 7030IP, o sinal de voz digital é convertido em sinal analógico e vice-versa seguindo o padrão G711.
- Interface Bidirecional Setha 7059: Interface Híbrida ativa reponsável pela conversão multicanal de sinais analógicos half-duplex não balanceados para sinais analógicos full-duplex balanceados com 1,5 V_{RMS} .

A Figura 2.12 apresenta uma topologia típica de aplicação do sistema atual elucidando os principais componentes do sistema. Nela, pode-se ver as Consoles Setha 7030IP conectadas a um *switch,* que, por sua vez, se comunica com um *switch* remoto por meio de fibra óptica. No *switch* remoto está conectada a Interface Digital Setha 7058, que, por sua vez, está conectada à Interface Bidirecional Setha 7059, responsável por inserir o áudio nos canais analógicos ligados aos amplificadores de potência e às unidades de conversação e chamada (UCCs) Setha 7011Ex. As interfaces e amplificadores estão fisicamente alocados em armários, que ficam localizados na sala de controle de cada área de processo. As UCCs Setha 7011Ex e as cornetas, por sua vez, estão localizadas as áeras classificadas, ou seja, em áreas de processo.

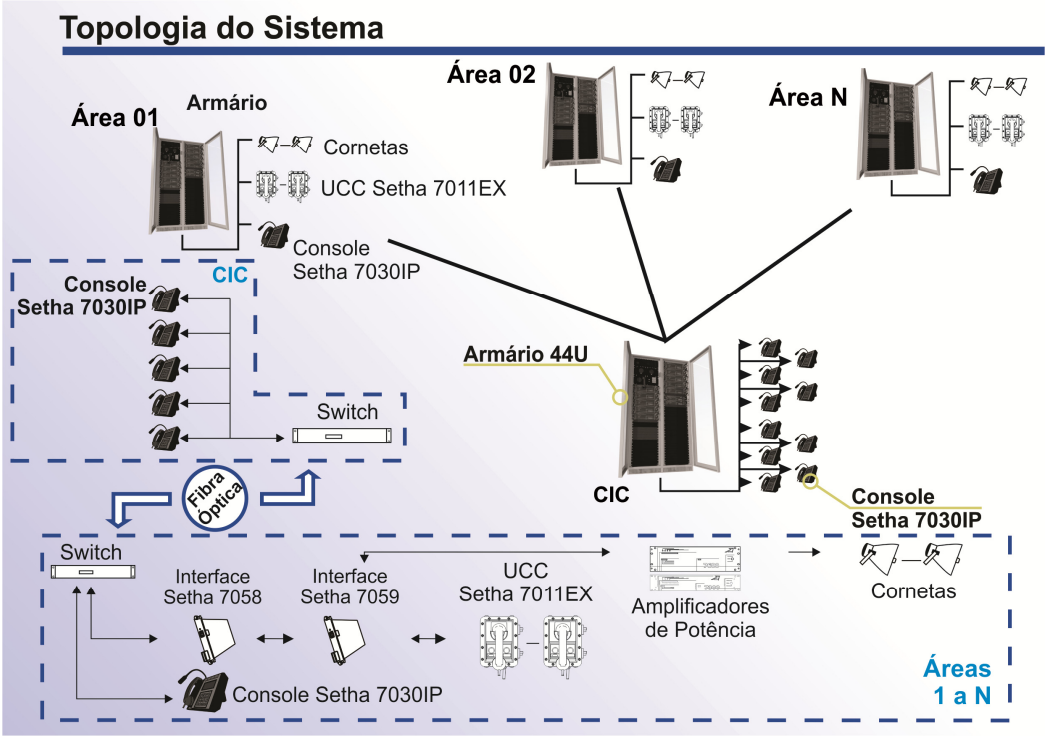

Figura 2.12 - Sistema Objetivo: as consoles Setha 7030IP localizadas no centro integrado de controle se comunicam com as interfaces Setha 7058 e Setha 7059 localizadas em armários de 44U de altura por meio de fibra óptica.

A seguir, será abordada a descrição funcional do sistema, explicando como são realizadas as chamadas de conversação em baixa e alta voz entre o Centro Integrado de Controle (CIC) e as áreas de processo.

2.6.1 - O fluxo de áudio

Conforme se verifica na Figura 2.12, a Console Setha 7030IP é conectada ao *switch* para transmissão dos sinais de áudio, controle e gerenciamento. Os sinais são enviados à Interface Digital 7058, que por sua vez também está conectada a um switch. Esta

converte o sinal de áudio digital em analógico e o retransmite à Interface Bidirecional 7059 via dois canais de áudio não balanceado (Tx e Rx), ou seja, dois canais em que a transmissão elétrica dos sinais de áudio é feita em relação ao terra (um fio com sinal e outro com terra). Empregando o esquema de interface híbrida ativa, abordado na Figura 2.5, a interface Bidirecional transforma os dois sinais não baleados em um sinal balanceado bidirecional, ou seja, em um sinal cuja transmissão é feita em dois fios de modo a evitar ruídos de linha, e o transmite ao canal correspondente (Linha K1, Linha K2, Chamada Interna ou Chamada Externa).

2.6.1.1 - Fluxo de acionamento de chamada em alta voz (CIC -> Área)

A chamada em alta voz é utilizada sempre que um operador quer enviar alguma mensagem para um grupo de pessoas que estão localizadas em áreas de processo ruidosas. Nela, o sinal de voz é direcionado a todas as cornetas presentes na área classificada. A Figura 2.13 apresenta seu modo de operação:

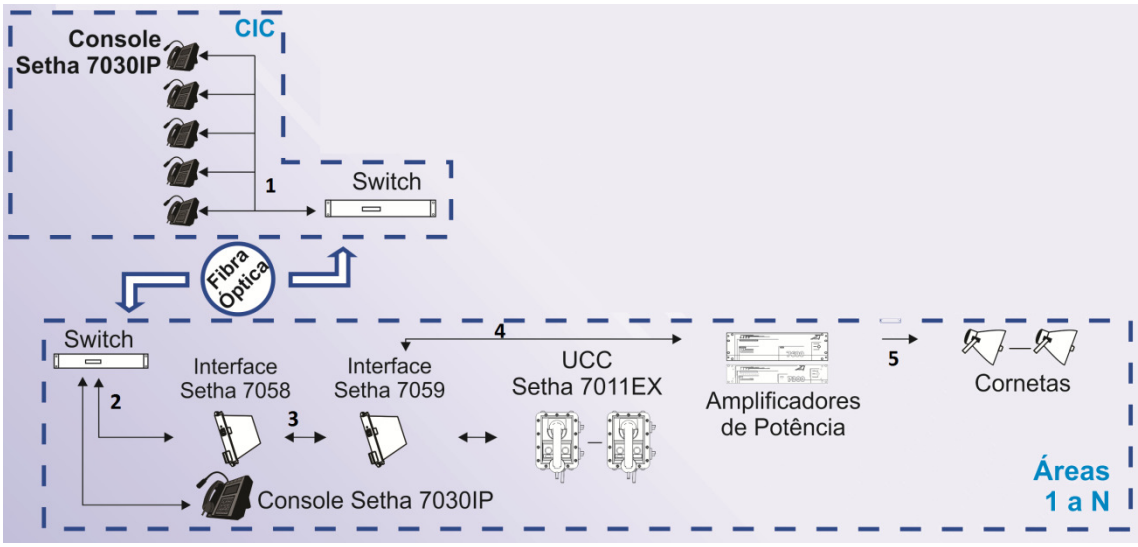

Figura 2.13 - Sistema Objetivo: processo de chamada em alta voz.

1-A console Setha 7030IP envia o sinal de voz digital à rede por meio do *switch*.

2-A Interface 7058 recebe e converte o sinal de áudio digital em dois sinais analógicos não balanceados (Tx e Rx) e os transmite à Interface Bidirecional 7059.

3-A Interface Bidirecional 7059 transforma os dois sinais não balanceados em um sinal balanceado bidirecional e o transmite ao Amplificador de Potência Setha 7300.

4-O amplificador de potência envia o sinal para a linha de transmissão de destino (Cornetas).

2.6.1.2 - Fluxo de conversação em baixa voz (CIC -> Área)

A chamada em baixa voz é utilizada sempre que o operador deseja conversar com uma ou mais pessoas em particular. Seu esquema de operação pode ser visto na Figura 2.14:

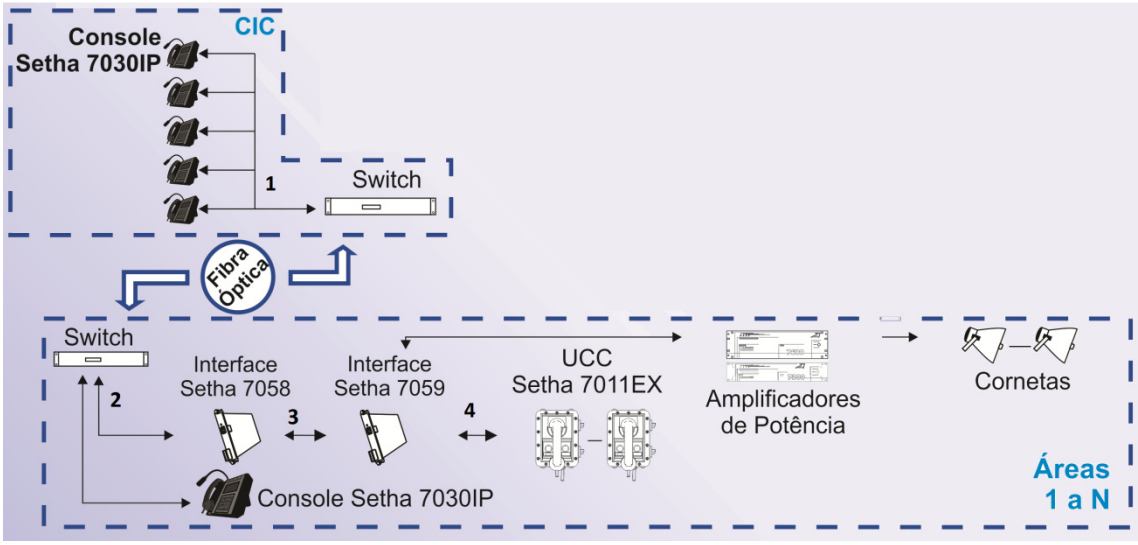

Figura 2.14 - Sistema Objetivo: processo de chamada em baixa voz.

1-A console Setha 7030IP envia o sinal de voz digital à rede por meio do switch.

2-A Interface 7058 recebe e converte o sinal de áudio digital em dois sinais analógicos não balanceados (Tx e Rx) e os transmite à Interface Bidirecional 7059.

3-A Interface Bidirecional 7059 transforma os dois sinais não balanceados em um sinal balanceado bidirecional e o envia para a linha de transmissão de destino (Linhas K1 ou K2 – UCC's Setha 7011Ex).

2.6.1.3 - Fluxo de conversação em baixa voz (CIC -> Área) via Console IP

A conversação em baixa voz via console IP é utilizada quando o operador deseja conversar com o responsável pela área de processo localizado na sala de controle local. A Figura 2.15 aborda sua operação:

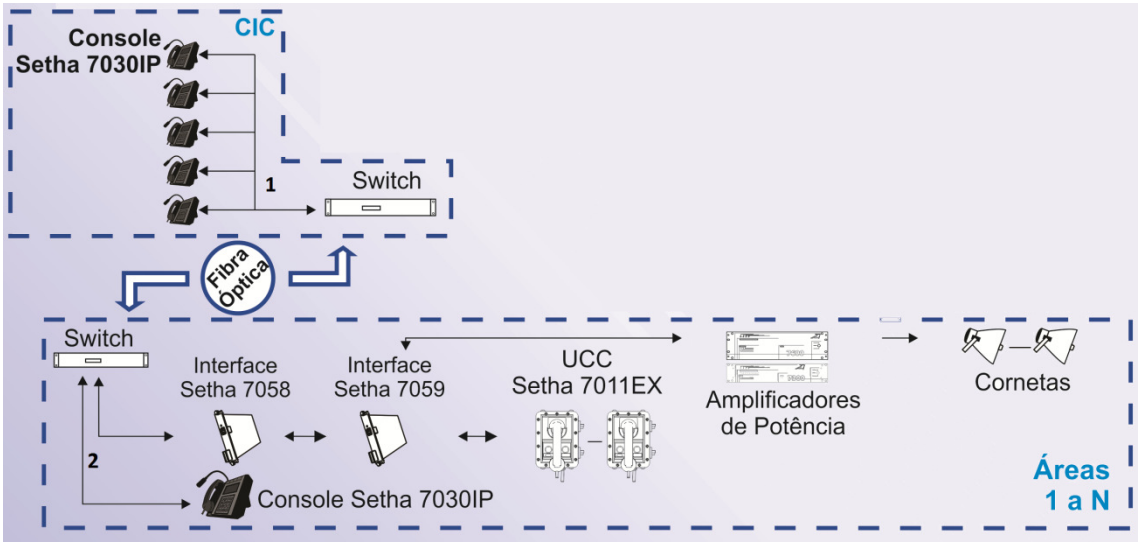

Figura 2.15 - Sistema Objetivo: processo de chamada IP em baixa voz.

1-A console Setha 7030IP do CIC envia o sinal de voz digital à rede por meio do switch.

2-A console Setha 7030IP do CIT recebe o sinal de voz digital da rede por meio do switch e o converte em analógico.

2.6.1.4 - Fluxo inverso (Área -> CIC)

Abordada na Figura 2.16, esta chamada é utilizada quando um operador na área classificada deseja conversar com o responsável pela área de processo.

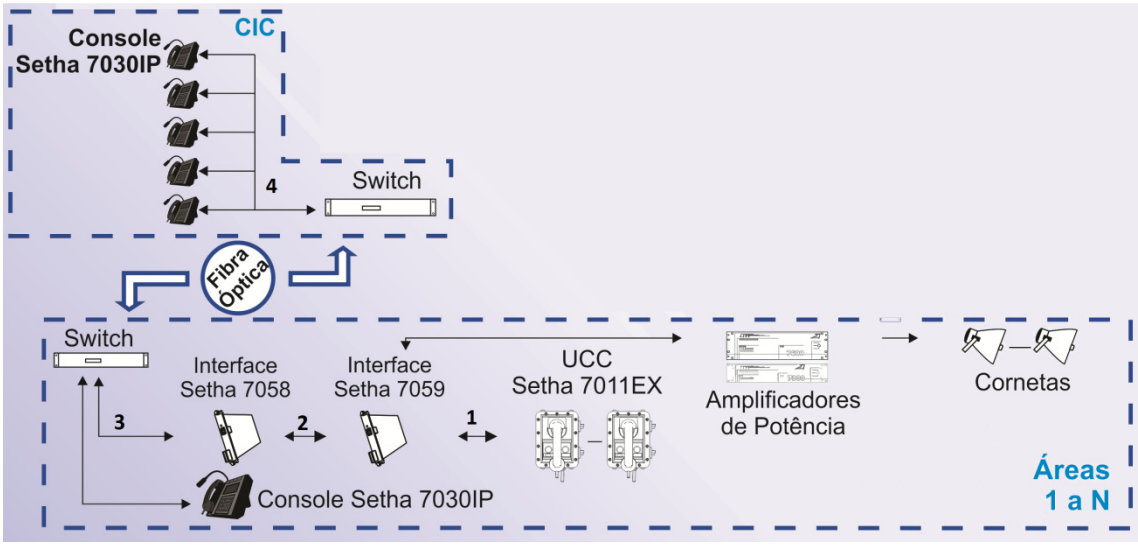

Figura 2.16 - Sistema Objetivo: processo de chamada em fluxo inverso.

1-A UCC 7011Ex transmite o áudio analógico balanceado bidirecional à Interface Bidirecional 7059 que o transforma em dois sinais analógicos não balanceados (Tx e Rx) e os retransmite à Interface Digital 7058.

2-A Interface 7058 converte estes sinais analógicos em um sinal digital e o transmite para a rede por meio do switch.

3- A console Setha 7030IP recebe o sinal de voz digital da rede por meio do switch e o converte em analógico.

2.6.2 - O Problema do Eco no Sistema Real

No sistema descrito existem os dois tipos de eco abordados neste trabalho: o eco acústico se apresenta no sistema de viva-voz da Console Setha 7030IP e o eco de linha está presente na Interface Bidirecional Setha 7059.

Conforme visto anteriormente, para o correto cancelamento do eco acústico são necessários filtros adaptativos de ordem elevada, pois estes devem rastrear os diferentes percursos de eco existentes em cada ambiente onde a estação de conversação possa ser instalada. Em um sistema real, isto implica em bufferização de dados, que, por sua vez, acarreta atrasos na transmissão/recepção do sinal digital que serão somados aos atrasos inerentes às redes de computador. Desta forma, a ordem dos filtros deve ser otimizada levando-se em conta a eficácia do cancelador de eco (MSE) e o atraso adicionado aos pacotes de voz.

Apesar de exigir filtros de ordem inferiores em relação ao eco acústico, o eco de linha presente no sistema objetivo tem seu efeito amplificado pelo fato deste ser convertido em sinal digital e ser reenviado para a rede pela Interface Digital Setha 7058. Ou seja, no sistema em questão todo e qualquer eco não atenuado será digitalizado e sofrerá um atraso inerente da rede de dados, o que possivelmente tornará a conversação impraticável.

A arquitetura do processador utilizado para a implementação dos filtros adaptativos pode trazer inúmeros desafios, pois grande parte dos algoritmos disponíveis na literatura são implementados em ponto flutuante despreocupando-se com problemas de saturação, precisão numérica e atraso de processamento. Uma implementação em um sistema embarcado deve preocupar-se em realizar o controle de ganhos, visando proteger os coeficientes adaptativos contra possíveis *overflows*, e, em casos de arquiteturas sem

suporte a ponto flutuante como a utilizada no sistema em questão, devem ser desenvolvidos algoritmos otimizados para operações em ponto fixo.

2.7 - Conclusões

Este capítulo explicitou o problema do eco em sistemas de comunicação, diferenciando suas naturezas elétrica e acústica. Mostrou o impacto causado pelo atraso na inteligibilidade da conversação e seus efeitos destrutivos com a adição do eco. Assim, demonstrou a importância dos filtros canceladores de eco, elucidando as características e leis de formação dos principais algoritmos utilizados na atualidade.

Este capítulo também apontou os cuidados a serem tomados ao implementar os algoritmos disponíveis na literatura em uma plataforma embarcada, explicando os percalços encontrados em uma aplicação real baseada em VoIP.

Apresentou-se também o sistema de intercomunicação industrial de interesse, plataforma embarcada na qual os filtros canceladores de eco desenvolvidos neste trabalho serão implementados e testados. Para isto, o modo de operação do sistema foi discutido, visando esclarecer os papéis de cada componente do sistema.

Capítulo 3 - Cancelamento de Eco Elétrico

3.1 - Introdução

Neste capítulo será discutido o desenvolvimento sem restrições de um cancelador de eco elétrico. Para isto, três dos modelos de canal explicitados na recomendação ITU-T G.168 [12] serão simulados mediante a inserção de amostras de voz anecóicas como sinal de referência.

Para escolher os parâmetros de cada algoritmo, será realizado um estudo de otimização envolvendo todos os algoritmos citados na Seção 2.5.4 do capítulo anterior.

Neste contexto, o objetivo deste capítulo é escolher por meio de um estudo comparativo de desempenho os algoritmos, e seus respectivos parâmetros, mais apropriados para uma implementação em uma plataforma embarcada real.

Assim, a Seção 3.2 aborda três dos percursos de eco explicitados na recomendação ITU-T G.168 [12], apresentando seus modelos de resposta ao impulso. A Seção 3.3 apresenta os sinais de referência utilizados ao longo de todo este trabalho. A Seção 3.4, por sua vez, apresenta a metodologia do estudo de otimização de parâmetros, explicando a métrica utilizada e o esquema de avaliação. A Seção 3.5 apresenta os resultados experimentais obtidos, onde os aspectos quantitativo e qualitativo são levados em conta para a escolha dos parâmetros ótimos de cada algoritmo citado na Seção 2.5.4 do capítulo anterior. A Seção 3.6 apresenta um comparativo entre o desempenho de todos os algoritmos testados, escolhendo aqueles mais apropriados para uma implementação em sistemas reais. A Seção 3.7 conclui o presente capítulo, sumarizando os assuntos nele abordados.

3.2 - Percurso de Eco

Segundo [12], o percurso de eco pode ser simulado por um filtro digital linear com resposta ao impulso $g(n)$. Com o objetivo de levar em conta diferentes valores de atraso (δ), atenuações (*ERL*), características de dispersão e duração, $g(n)$ é escolhido como uma versão atrasada e atenuada de qualquer uma das sequencias $m_i(n)$ para

$i = 1, 2, \ldots, 8$, que representam percursos de eco com diferentes características de dispersão e duração.

$$g(n) = \left(10^{-\frac{ERL}{20}} K_i\right) m_i(n - \delta). \tag{3.1}$$

No presente estudo, apenas as sequências $m_1(n)$, $m_5(n)$ e $m_6(n)$ serão analisadas. Os valores mínimos de *ERL* e K_i são fixados por [12] como mostra a Tabela 3.1 abaixo:

Tabela 3.1 – Fator de escalamento e ERL mínimo utilizados em todos os canais simulados.

Percurso de Eco	Fator de Escalamento K_i	*ERL* mínimo
$\boldsymbol{m_1(n)}$	$1.39\ x\ 10^{-5}$	$6\ dB$
$\boldsymbol{m_5(n)}$	$9.33\ x\ 10^{-6}$	$6\ dB$
$\boldsymbol{m_6(n)}$	$1.51\ x\ 10^{-5}$	$6\ dB$

Para as simulações, foram utilizados $\delta = 0$ e os valores de ERL e K_i mínimos, ou seja, quando a atenuação imposta pelo canal é a menor possível. Desta forma, o cancelador de eco foi exposto ao cenário mais crítico, onde o sinal a ser cancelado possui elevada energia, e, consequentemente, precisa sofrer uma grande atenuação.

As respostas ao impulso dos canais $\boldsymbol{m_1(n)}$, $\boldsymbol{m_5(n)}$ e $\boldsymbol{m_6(n)}$ podem ser vistas nas Figuras 3.1, 3.2 e 3.3, respectivamente [12]:

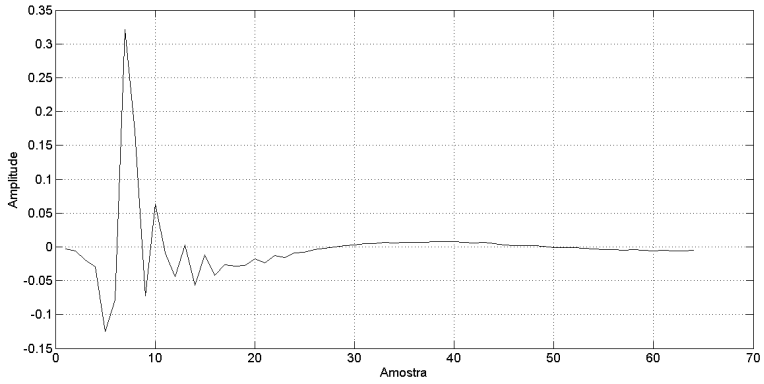

Figura 3.1 - Resposta ao impulso do modelo de percurso de eco 1 com ERL = 6dB e δ = 0.

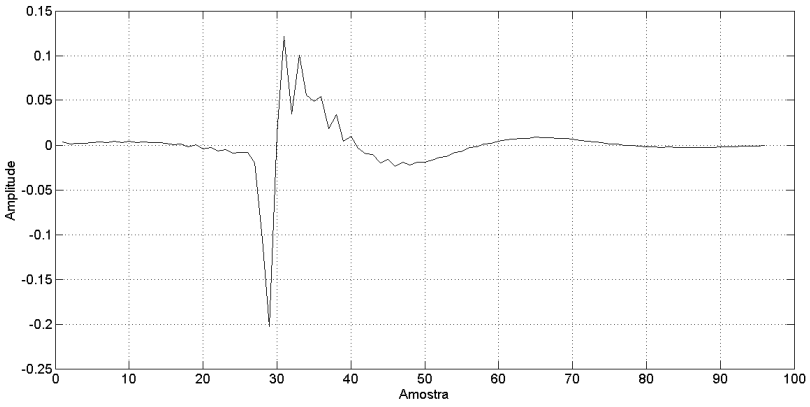

Figura 3.2 - Resposta ao impulso do modelo de percurso de eco 5 com ERL = 6dB e δ = 0.

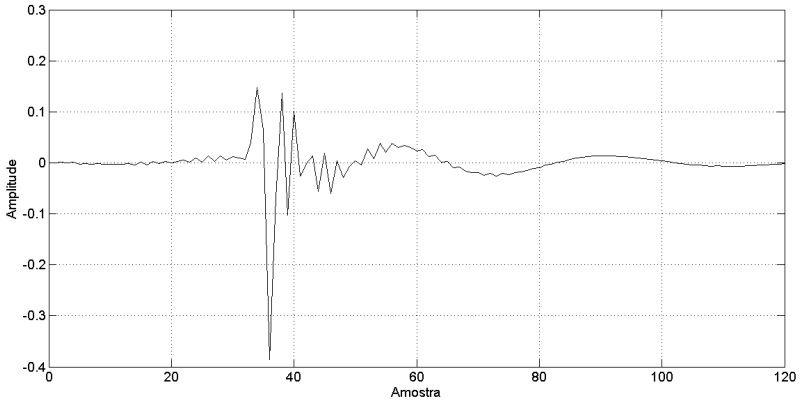

Figura 3.3 - Resposta ao impulso do modelo de percurso de eco 6 com ERL = 6dB e δ = 0.

3.3 – Sinais de Referência

Para simular o comportamento dos canceladores de eco, foram utilizados sinais anecoicos de voz masculina e feminina, obtidos a partir da gravação de orações na língua portuguesa com duração aproximada de oito segundos e taxa de amostragem de 48kbps.

Conforme recomendado por [12], os trechos de voz são compostos por períodos de sinal sonoro, sinal surdo e silêncio. Desta forma, o desempenho médio dos algoritmos levarão em conta tanto a convergência como a re-convergência, parâmetros de suma importância para o cancelamento ótimo do eco.

Apresentado na Figura 3.4, o sinal de referência 1 é composto por uma voz feminina falando: “A sensibilidade indicará a escolha, a Amazônia é a reserva ecológica do globo”.

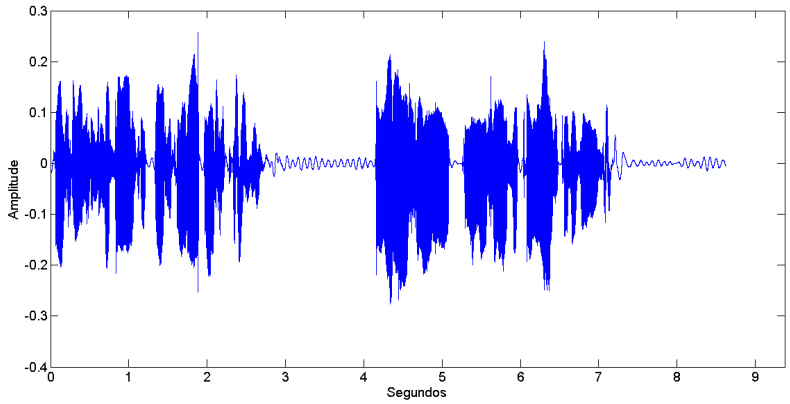

Figura 3.4 - Referência 1 - Voz feminina falando "A sensibilidade indicará a escolha, a Amazônia é a reserva ecológica do globo".

O sinal de referência 2, exposto na Figura 3.5, é composto por uma voz feminina falando: "O ministério mudou demais com a eleição, novos rumos se abrem para a informática".

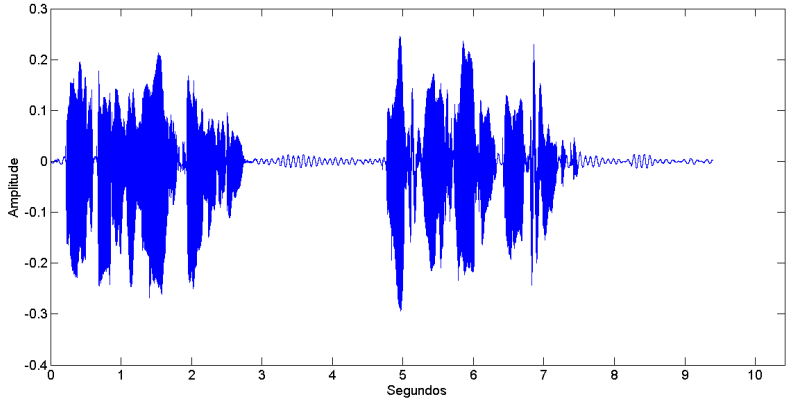

Figura 3.5 - Referência 2 - Voz feminina falando "O ministério mudou demais com a eleição, novos rumos se abrem para a informática".

O sinais de referência 3 e 4, retratados nas figuras 3.6 e 3.7, são compostos por uma voz masculina falando as mesmas frases dos sinais de referência 1 e 2, respectivamente.

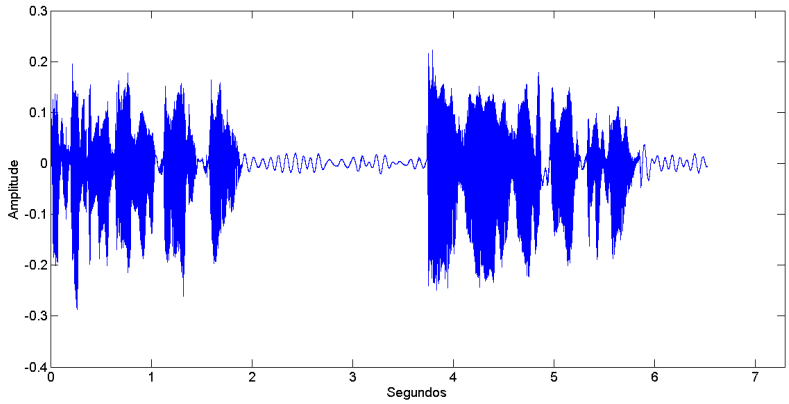

Figura 3.6 - Referência 3 - Voz masculina falando "A sensibilidade indicará a escolha, a Amazônia é a reserva ecológica do globo".

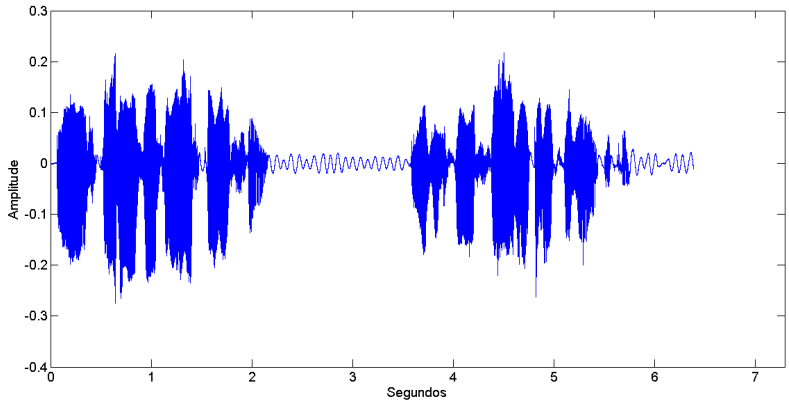

Figura 3.7 - Referência 4 - Voz masculina falando "O ministério mudou demais com a eleição, novos rumos se abrem para a informática".

3.4 – Otimização de Parâmetros

Visando uma comparação justa entre os algoritmos adaptativos, foi desenvolvido um estudo para encontrar os parâmetros ótimos de cada algoritmo para os sinais e canais considerados. Assim, todos os algoritmos serão testados com os parâmetros que asseguram o melhor desempenho médio e a comparação entre eles será baseada no desempenho real do *setup* ótimo de cada estratégia de adaptação.

A metodologia deste estudo foi baseada na efetuação de 210 (duzentas e dez) simulações para o algoritmo LMS e 140 (cento e quarenta) simulações para os demais algoritmos. As simulações foram realizadas para filtros de ordem iguais a 8, 16, 32, 64, 128, 256, 512, e, para cada ordem de filtro, o passo de adaptação variou desde 0,01 a 0,3 para o algoritmo LMS, de 0,5 a 1,4 para o algoritmo NDRLMS e 0,05 a 1,0 para os demais.

3.4.1 - Métrica

A métrica do estudo é baseada na minimização de um componente quantitativo e outro qualitativo, o *MSE (Mean Squared Error)* Médio e o *MOS (Mean Opinion Score)* Médio, respectivamente.

O objetivo é cruzar os efeitos de minimização do erro quadrático médio com a percepção de qualidade do áudio, onde as características psicoacusticas são levadas em conta. Neste cenário, estudaremos a relação entre qualidade sonora e minimização do *MSE*, o que nos levará a encontrar os valores ótimos de ordem e passo de adaptação do filtro.

3.4.1.2 – MOS (Mean Opinion Score)

Segundo [6], a forma mais eficiente para medir a qualidade de sinais de voz é a realização de testes subjetivos, onde os sinais são expostos a pessoas e estas são questionadas sobre sua qualidade. Estes testes subjetivos são padronizados em [13], onde são descritos os principais tipos de teste e as condições de como a nota média de qualidade *MOS* deve ser computada.

Basicamente dois tipos de teste são utilizados: o ACR *(Absoloute Category Rating)* e o DCR *(Degradation Category Rating)*. No ACR o avaliador deve realizar sua análise de qualidade baseando-se apenas em um sinal de teste apresentado. No DCR o avaliador é apresentado a dois sinais distintos, o sinal de referência e o sinal de teste. O avaliador deve quantificar o nível de degradação no sinal de teste comparando-o ao sinal de referência. A tabela de qualificação para ambos os métodos por ser vista abaixo:

Tabela 3.2 - Escala MOS para os testes ACR e DCR.

Escala MOS	ACR	DCR
5	Excelente	Inaudível
4	Boa	Audível mas não incômoda
3	Regular	Ligeiramente incômoda
2	Pobre	Incômoda
1	Ruim	Muito incômoda

3.4.1.2 – W-PESQ (Wideband Perceptual Evaluation of Speech Quality)

Para avaliar a qualidade do áudio foi utilizado o algoritmo W-PESQ, estado da arte em avaliação de sinais de voz. Segundo [14], este algoritmo apresenta uma correlação de 0.935 com o MOS subjetivo.

Por utilizar o método DCR para avaliação dos sinais, o algoritmo W-PESQ necessita de um sinal de referência e outro para análise. O esquema de avaliação adotado pode ser visto na Figura 3.8.

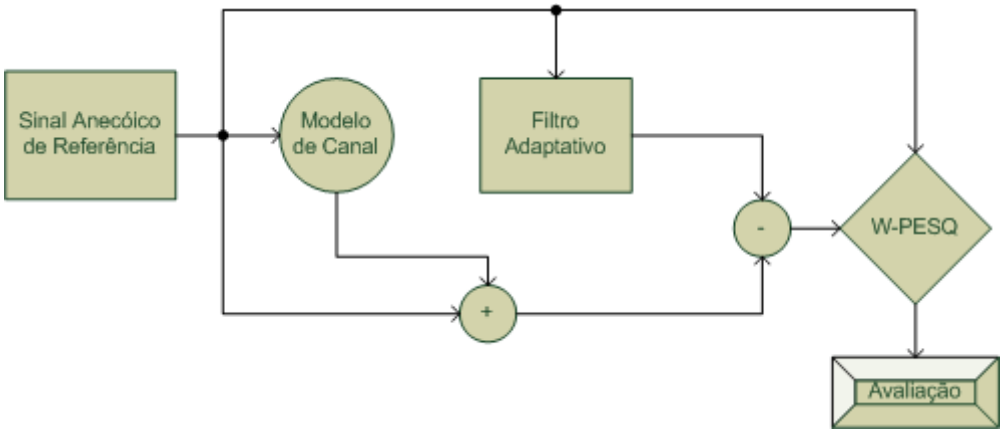

Figura 3.8- Esquema de Avaliação W-PESQ: O sinal anecoico é utilizado como referência na avaliação W-PESQ em comparação ao sinal obtido pela diferença entre o sinal acrescido de eco e o sinal de eco modelado pelo filtro adaptativo.

Neste estudo, os sinais apresentados na Seção 3.3 foram introduzidos como referência. O sinal a ser analisado, por sua vez, é composto pela subtração entre a saída do filtro adaptativo e a soma do sinal de referência com o eco gerado por um dos modelos de canal explicitados na Seção 3.2. Ou seja, o algoritmo W-PESQ avaliará o quão bem o eco foi cancelado por parte do algoritmo adaptativo. Vale ressaltar que a nota máxima dada pelo algoritmo W-PESQ é igual a 4.5, quando os sinais de referência e teste são idênticos.

3.5 – Resultados Experimentais

A seguir, são apresentados os resultados obtidos pela otimização de parâmetros de cada algoritmo para cada modelo de canal. Apenas os resultados para o sinal de referência 1 serão explicitados, uma vez que os apresentados pelos demais foram equivalentes.

3.5.1 – Desempenho do Algoritmo LMS

O desempenho do algoritmo LMS de acordo com a variação de seus parâmetros pode ser visto nas figuras 3.10 a 3.14. As figuras 3.10 e 3.11 apresentam o desempenho em

minimização do MSE e o desempenho em MOS para o percurso de eco 1, as figuras 3.12 e 3.13 apresentam os mesmos experimentos para o percurso de eco 5 e as figuras 3.14 e 3.15 apresentam os mesmos experimentos para o percurso de eco 6.

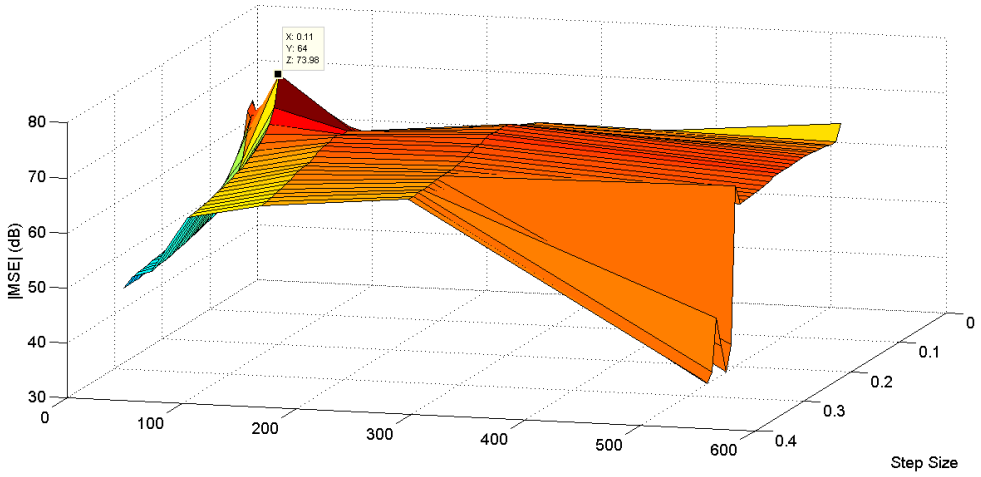

Figura 3.9 – LMS: Modelo de Percurso de Eco 1.

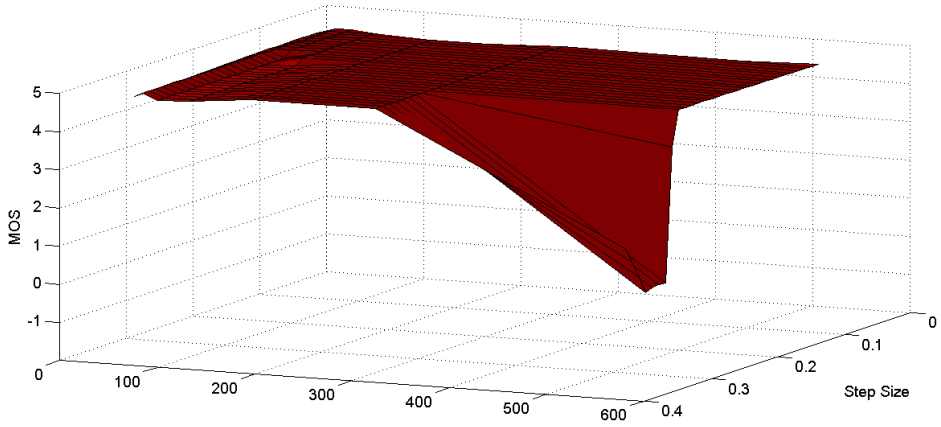

Figura 3.10 - MOS LMS: Modelo de Percurso 1.

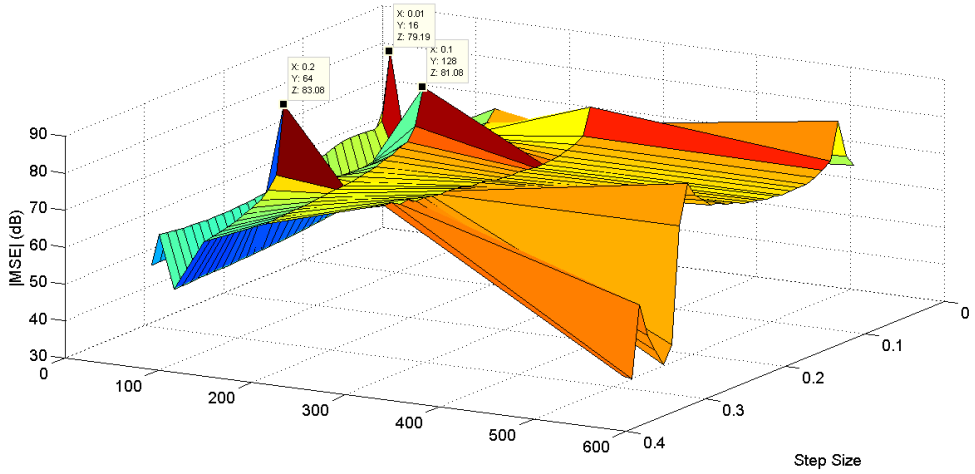

Figura 3.11 - LMS: Modelo de Percurso de Eco 5.

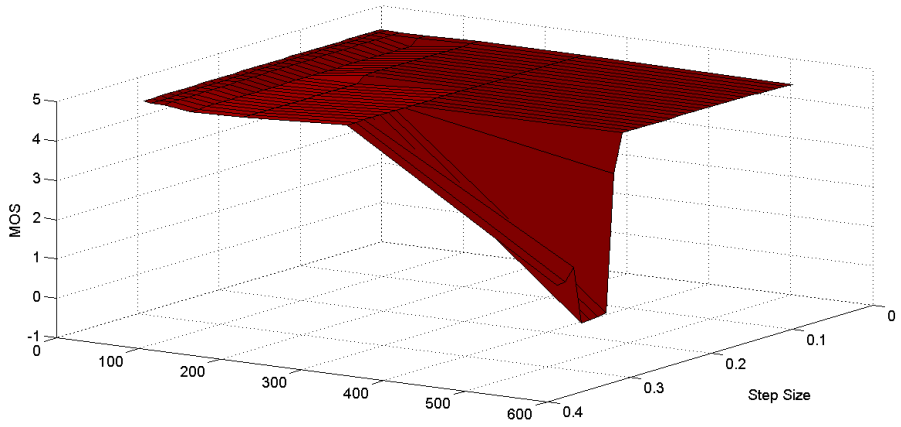

Figura 3.12 - MOS LMS: Modelo de Percurso 5.

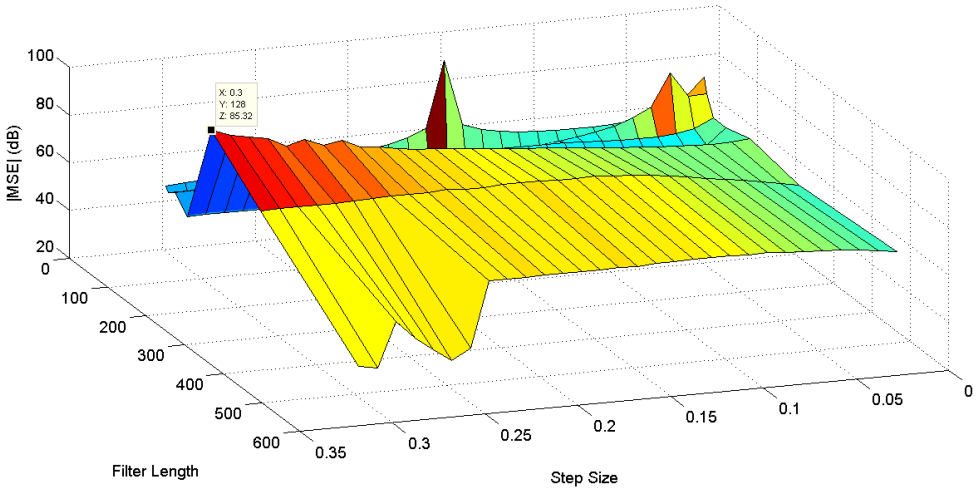

Figura 3.13 - LMS: Modelo de Percurso de Eco 6.

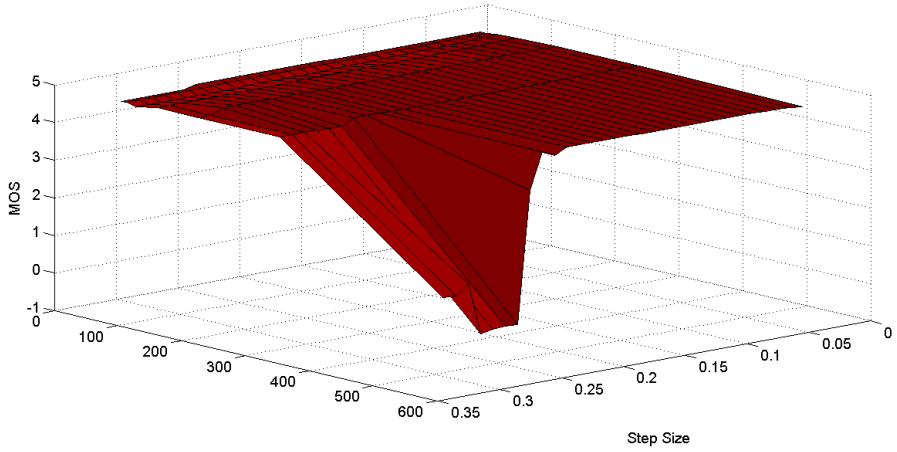

Figura 3.14 - MOS LMS: Modelo de Percurso 6

Pode-se observar a partir das simulações de parâmetros que o algoritmo LMS apresenta elevada sensibilidade em relação à ordem do filtro e passo de adaptação escolhidos. Para todos os canais simulados os melhores resultados são obtidos com o tamanho do filtro menor do que 256 amostras.

Outra característica interessante é a estabilização da minimização do erro quadrático médio, ou seja, independentemente do incremento na complexidade do filtro (ordem), o desempenho na minimização do erro quadrático medido permanece inalterado entre 50 e 60 dB em todos os canais simulados.

A estabilização da minimização do erro em conjunto com o desempenho inferior para filtros de ordem menor do que 256 estabelece uma clara orientação na escolha dos parâmetros de um filtro adaptativo. Quando não se conhece a resposta ao impulso do canal a ser utilizado, é interessante superestimar o comprimento do filtro adaptativo, pois a derivada da curva de desempenho é muito negativa para filtros menores do que a resposta do canal e praticamente zero para filtros maiores do que a resposta do canal. Em outras palavras, é melhor aumentar a complexidade do algoritmo para obter um resultado médio do que tentar obter uma resposta ótima e acabar em um ponto de mínimo na curva de desempenho.

O cruzamento de dados entre as métricas também oferece valiosas conclusões. Além de acompanhar a curva de minimização do erro quadrático médio, a curva de MOS demonstra que as pequenas variações de desempenho na minimização do erro não são percebidas na qualidade do áudio. Ou seja, se os parâmetros do filtro forem escolhidos de forma a garantir um ponto de operação médio (tamanho acima de 256 e passo entre 0

e 0,25), o filtro realizará um cancelamento de eco equivalente (MOS médio entre 4,0 e 4,3), para quaisquer parâmetros escolhidos na faixa especificada.

3.5.2 – Desempenho do Algoritmo NLMS

O desempenho do algoritmo NLMS de acordo com a variação de seus parâmetros pode ser visto nas figuras 3.15 a 3.20. Onde as figuras 3.15 e 3.16 apresentam o desempenho em minimização do MSE e o desempenho em MOS para o percurso de eco 1, as figuras 3.17 e 3.18 apresentam os mesmos experimentos para o percurso de eco 5 e as figuras 3.19 e 3.20 apresentam os mesmos experimentos para o percurso de eco 6.

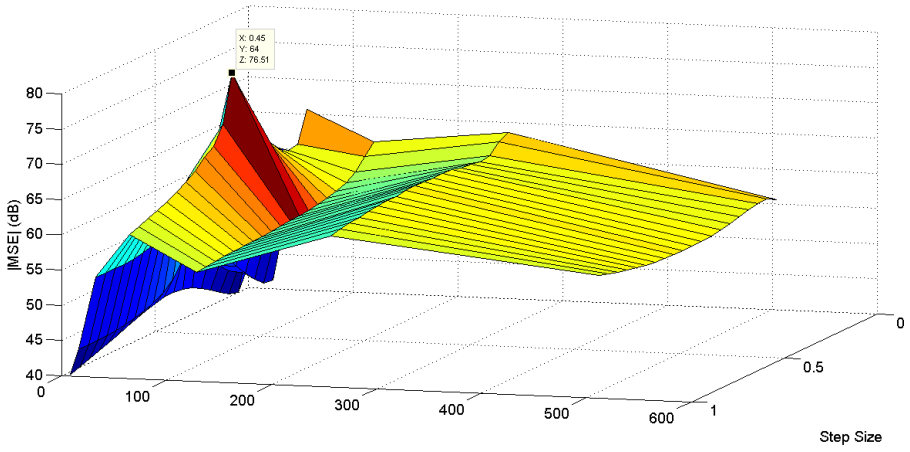

Figura 3.15 - NLMS: Modelo de Percurso de Eco 1.

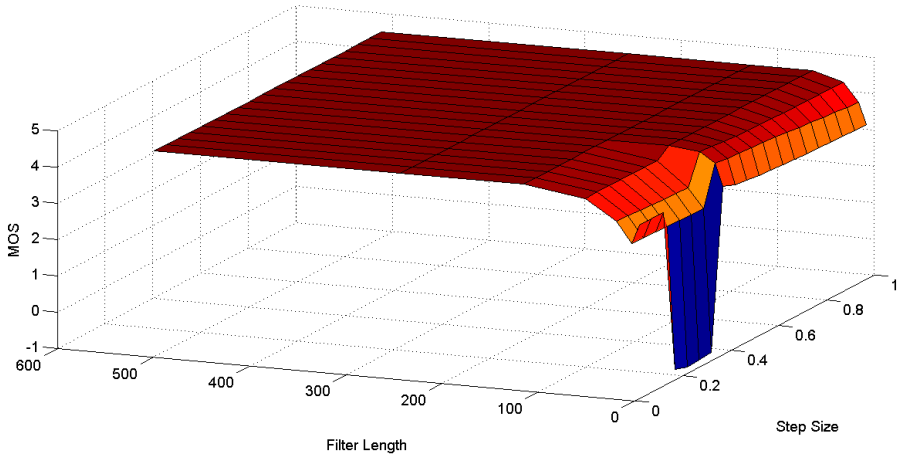

Figura 3.16 - MOS NLMS: Modelo de Percurso 1.

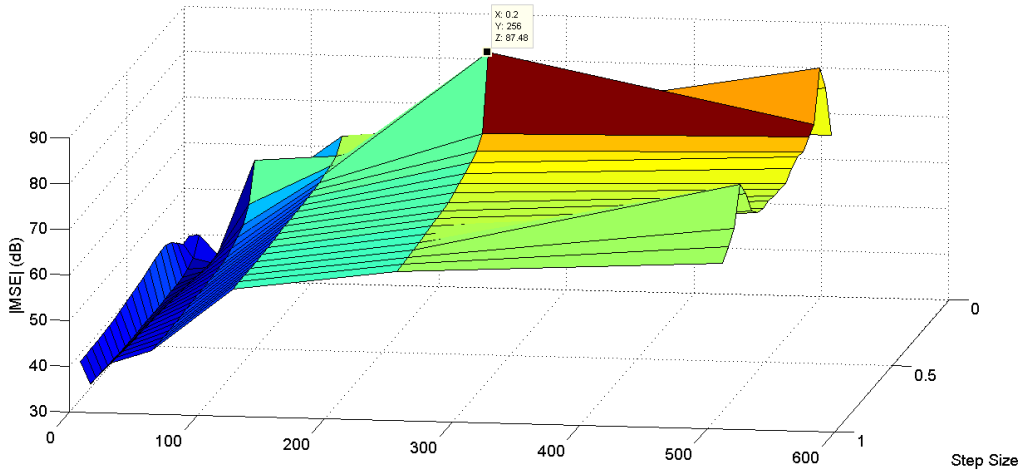

Figura 3.17 - NLMS: Modelo de Percurso de Eco 5.

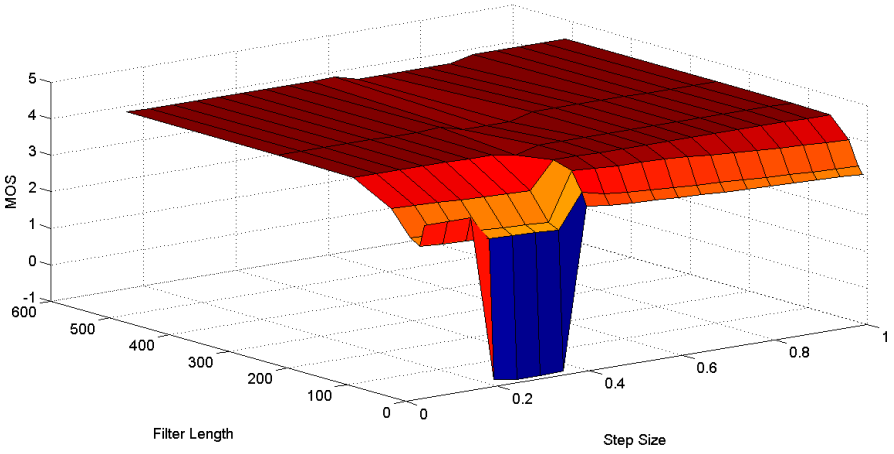

Figura 3.18 - MOS NLMS: Modelo de Percurso 5.

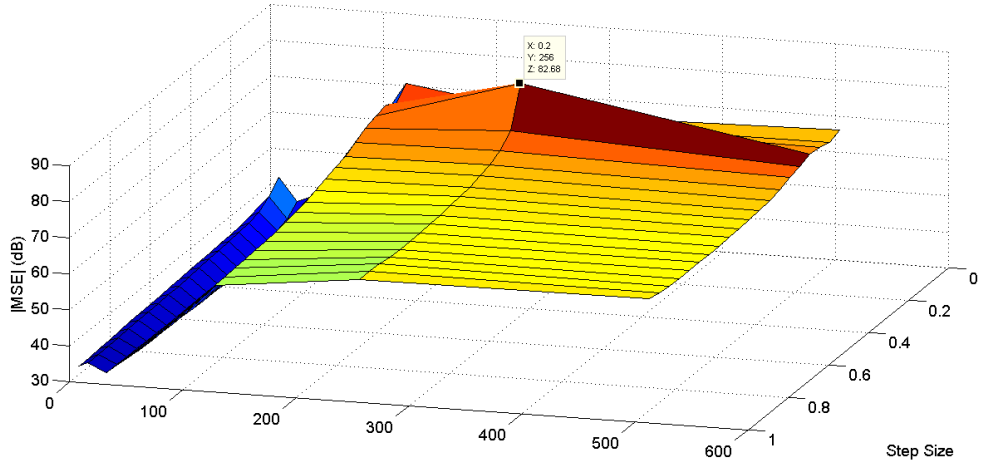

Figura 3.19 - NLMS: Modelo de Percurso de Eco 6.

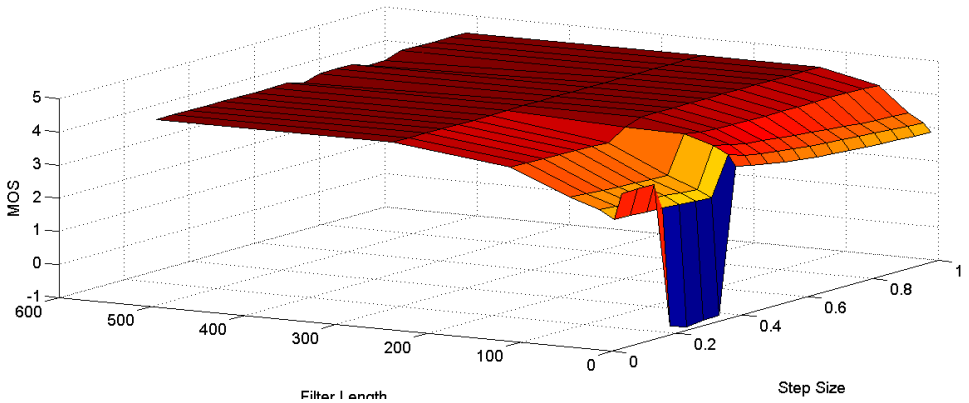

Figura 3.20 - MOS NLMS: Modelo de Percurso 6.

Pode-se observar a partir das simulações de parâmetros que a normalização pela energia do sinal do passo de adaptação faz com que o algoritmo NLMS apresente baixa sensibilidade em relação à ordem do filtro e passo de adaptação escolhidos. Para todos os canais simulados os melhores resultados médios são obtidos com o filtro de ordem 256.

A estabilização da minimização do erro quadrático médio se dá em níveis superiores aos apresentados pelo algoritmo LMS em todos os canais simulados (entre 55 e 65 dB).

Mesmo com desempenho superior nas regiões abaixo de 256 amostras, a orientação geral de escolha de parâmetros explicitada anteriormente ainda vale para o algoritmo NLMS.

O cruzamento de dados entre as métricas mostra a superior qualidade no cancelamento de eco do algoritmo NLMS. Para todos os canais simulados, o MOS médio obtido varia suavemente entre 3,8 e 4,4.

3.5.3 – Desempenho do Algoritmo PNLMS

O desempenho do algoritmo PNLMS de acordo com a variação de seus parâmetros pode ser visto nas figuras 3.21 a 3.26. Onde as figuras 3.21 e 3.22 apresentam o desempenho em minimização do MSE e o desempenho em MOS para o percurso de eco 1, as figuras 3.23 e 3.24 apresentam os mesmos experimentos para o percurso de eco 5 e as figuras 3.25 e 3.26 apresentam os mesmos experimentos para o percurso de eco 6.

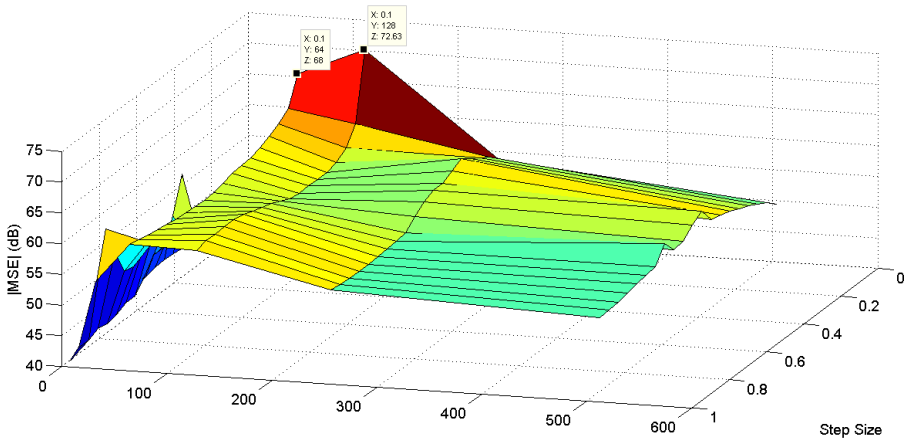

Figura 3.21 - PNLMS: Modelo de Percurso de Eco 1.

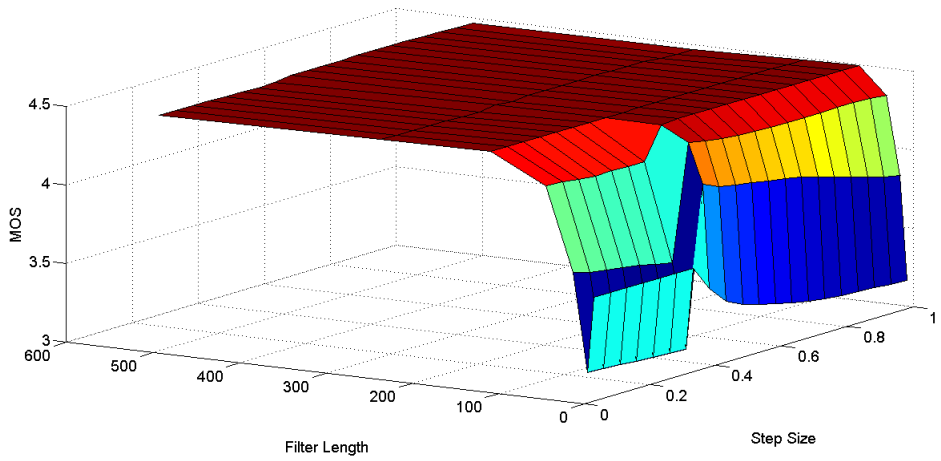

Figura 3.22 - MOS PNLMS: Modelo de Percurso 1.

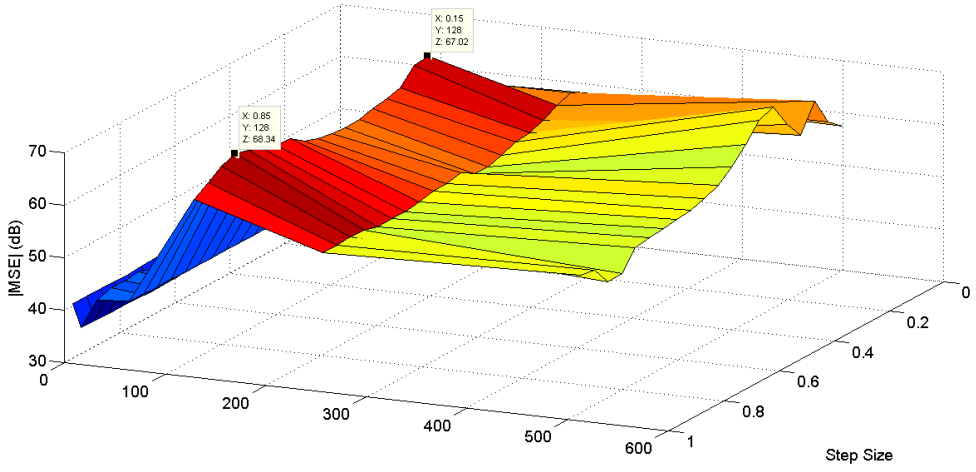

Figura 3.23 - PNLMS: Modelo de Percurso de Eco 5.

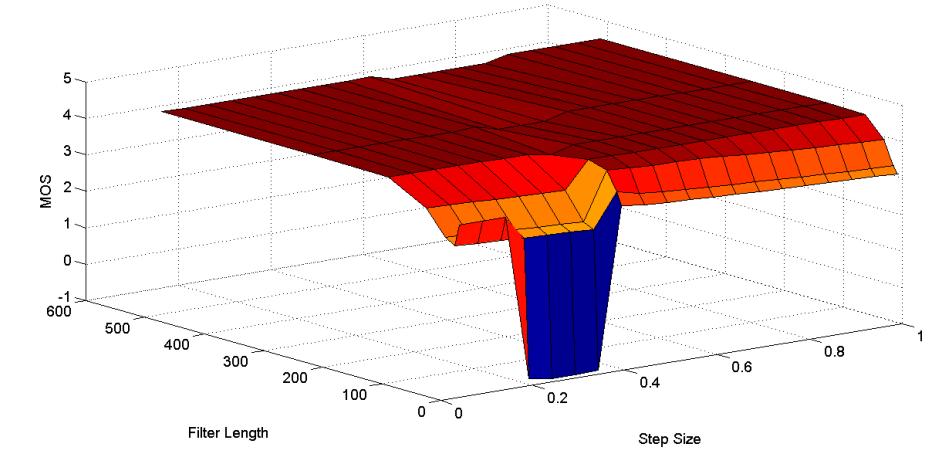

Figura 3.24 - MOS PNLMS: Modelo de Percurso 5.

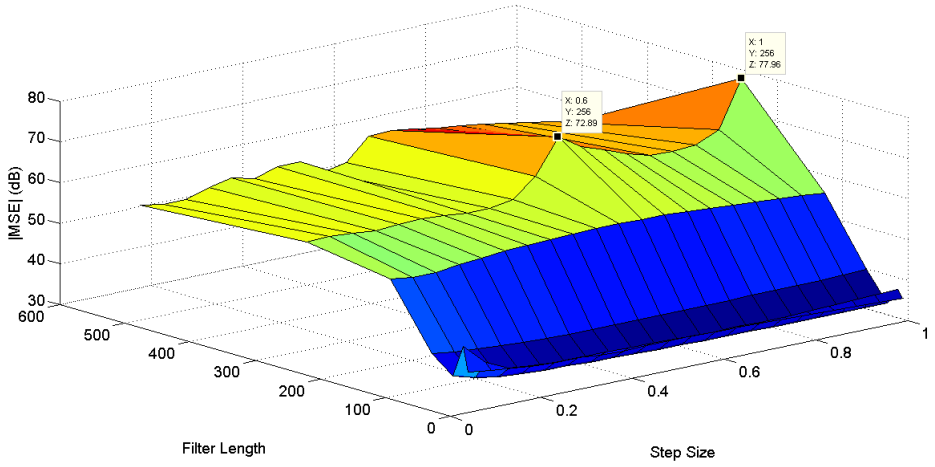

Figura 3.25 - PNLMS: Modelo de Percurso de Eco 6.

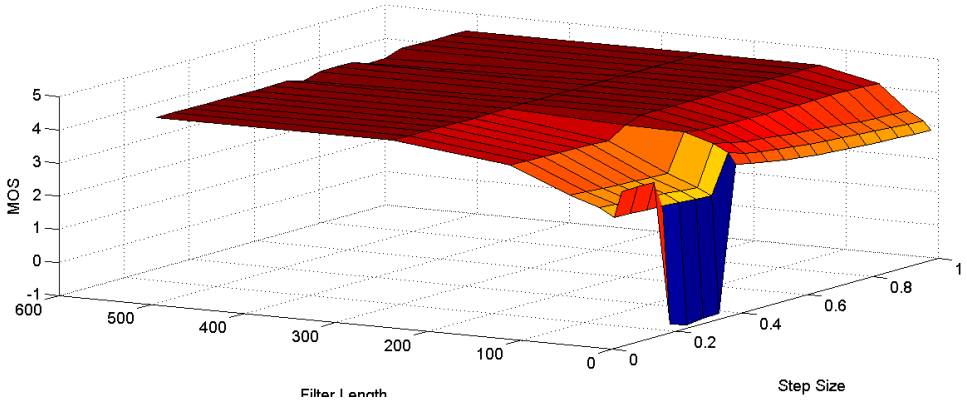

Figura 3.26 - MOS PNLMS: Modelo de Percurso 6.

Conforme o esperado, o algoritmo PNLMS apresenta baixa sensibilidade em relação à ordem do filtro e passo de adaptação escolhidos, assim como o NLMS. Para todos os

canais simulados os melhores resultados são obtidos com o tamanho do filtro entre 128 e 256 amostras.

A componente proporcional influencia apenas a velocidade de convergência e a estabilização da minimização do erro quadrático médio se dá em níveis equivalentes aos apresentados pelo algoritmo NLMS em todos os canais simulados (entre 50 e 60 dB).

O cruzamento de dados entre as métricas mostra a superior qualidade no cancelamento de eco do algoritmo PNLMS em relação ao NLMS, quando a ordem do filtro se aproxima da ordem da resposta ao impulso do canal. Para todos os canais simulados, o MOS médio obtido varia suavemente entre 4,0 e 4,4. Uma possível explicação para isto seria o fato do erro quadrático médio remanescente apresentar uma dispersão maior ao longo do sinal analisado, fazendo com que fosse menos percebido na avaliação de qualidade.

3.5.4 – Desempenho do Algoritmo NDRLMS

O desempenho do algoritmo NDRLMS de acordo com a variação de seus parâmetros pode ser visto nas figuras 3.27 a 3.32. Onde as figuras 3.27 e 3.28 apresentam o desempenho em minimização do MSE e o desempenho em MOS para o percurso de eco 1, as figuras 3.29 e 3.30 apresentam os mesmos experimentos para o percurso de eco 5 e as figuras 3.31 e 3.32 apresentam os mesmos experimentos para o percurso de eco 6.

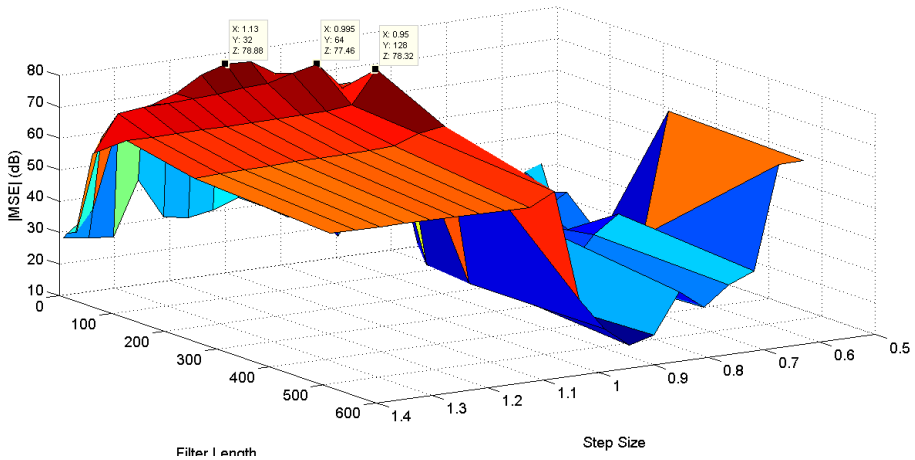

Figura 3.27 - NDRLMS: Modelo de Percurso de Eco 1.

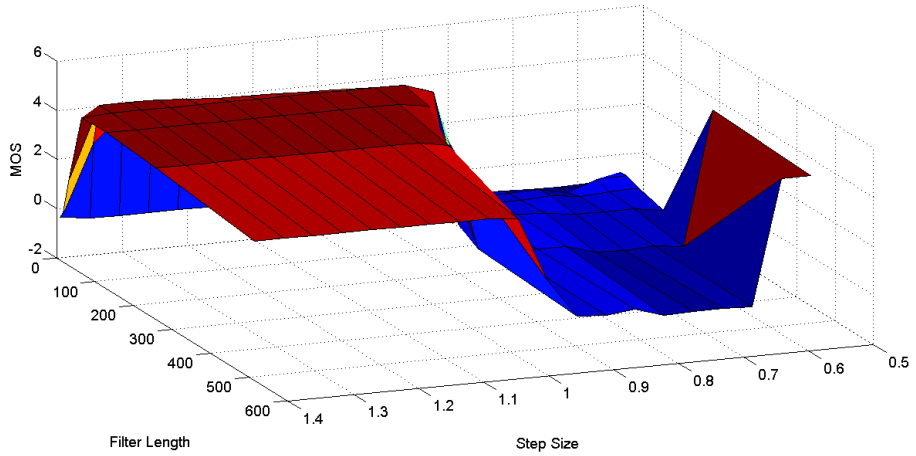

Figura 3.28 - MOS NDRLMS: Modelo de Percurso 1.

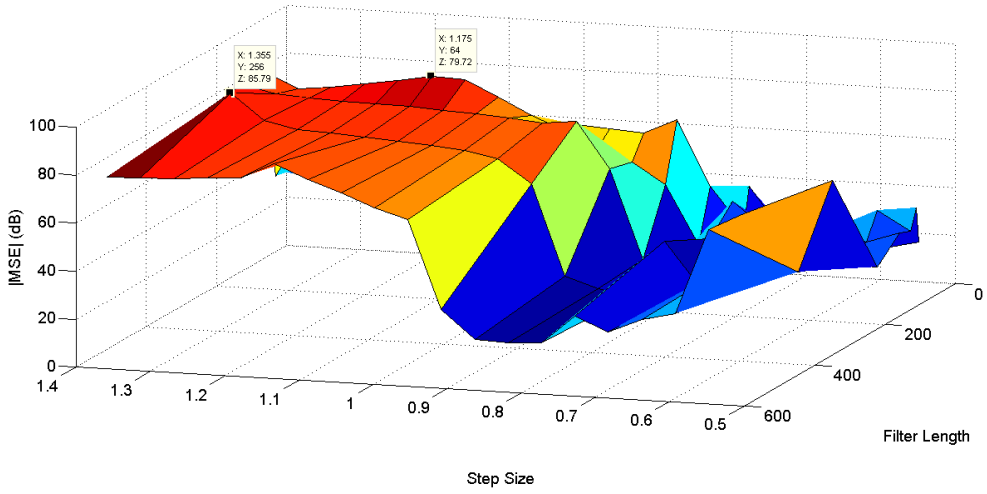

Figura 3.29 - NDRLMS: Modelo de Percurso de Eco 5.

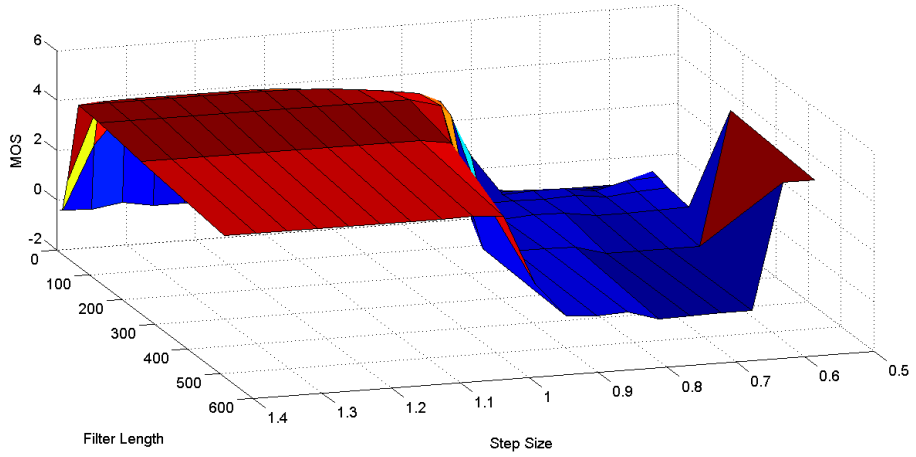

Figura 3.30 - MOS NDRLMS: Modelo de Percurso 5.

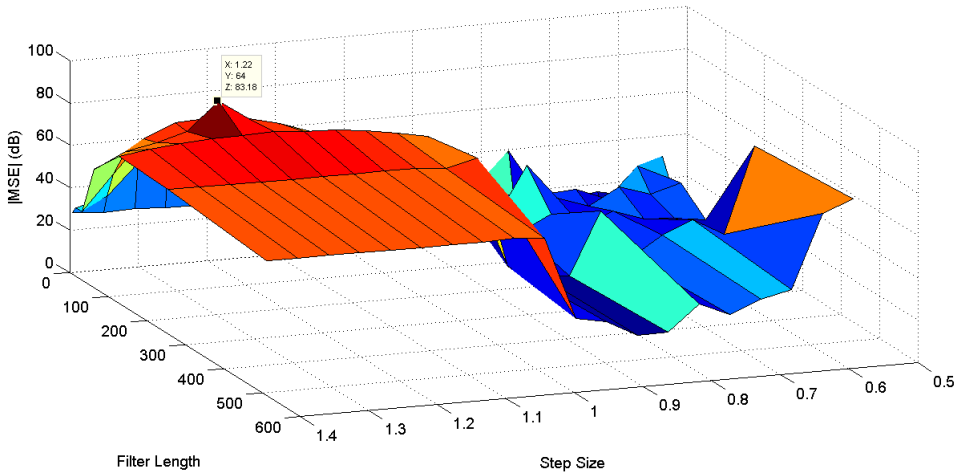

Figura 3.31 - NDRLMS: Modelo de Percurso de Eco 6.

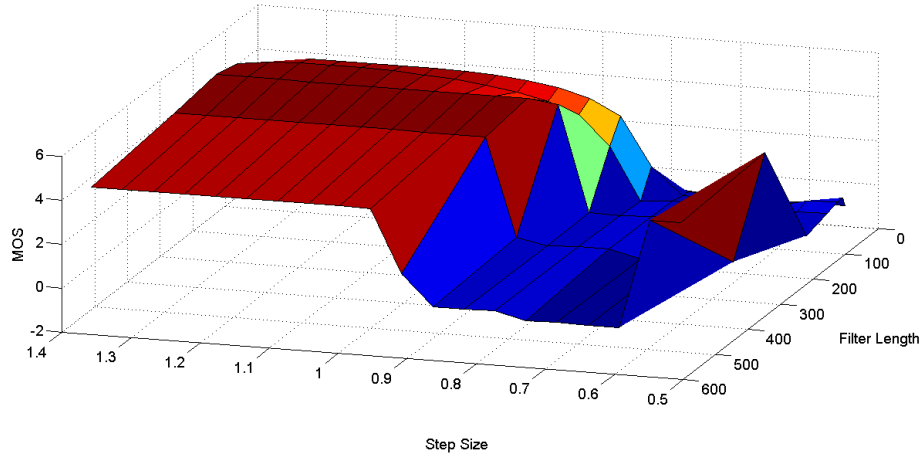

Figura 3.32 - MOS NDRLMS: Modelo de Percurso 6.

Apesar de apresentar a melhor resposta em minimização do erro quadrático médio, o algoritmo NDRLMS apresenta elevada sensibilidade em relação ao passo de adaptação escolhido. Para todos os canais simulados os melhores resultados são obtidos com o passo de adaptação variando entre 0,9 e 1,4.

A reutilização de dados influencia diretamente na minimização do erro quadrático médio, elevando os níveis em todos os canais simulados para a faixa entre 65 e 90 dB. Para todos os canais simulados, considerando a faixa correta de passo de adaptação, o MOS médio obtido varia suavemente entre 4,0 e 4,2.

3.5.5 – Desempenho do Algoritmo BNDRLMS

O desempenho do algoritmo BNDRLMS de acordo com a variação de seus parâmetros pode ser visto nas figuras 3.33 a 3.38. Onde as figuras 3.33 e 3.34 apresentam o desempenho em minimização do MSE e o desempenho em MOS para o percurso de eco 1, as figuras 3.35 e 3.36 apresentam os mesmos experimentos para o percurso de eco 5 e as figuras 3.37 e 3.38 apresentam os mesmos experimentos para o percurso de eco 6.

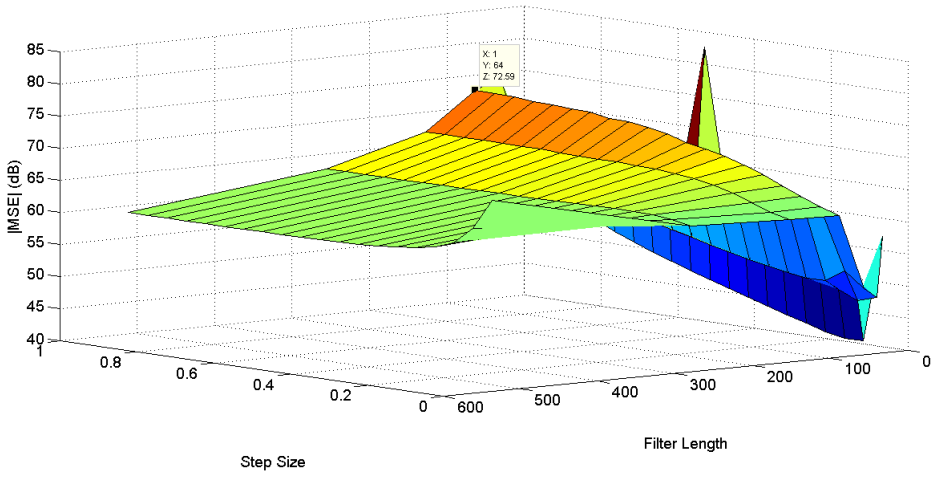

Figura 3.33 - BNDRLMS: Modelo de Percurso de Eco 1.

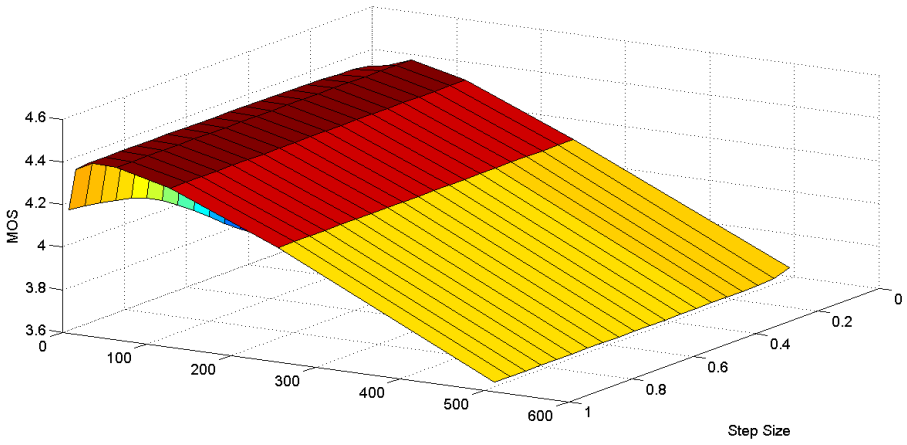

Figura 3.34 - MOS BNDRLMS: Modelo de Percurso 1.

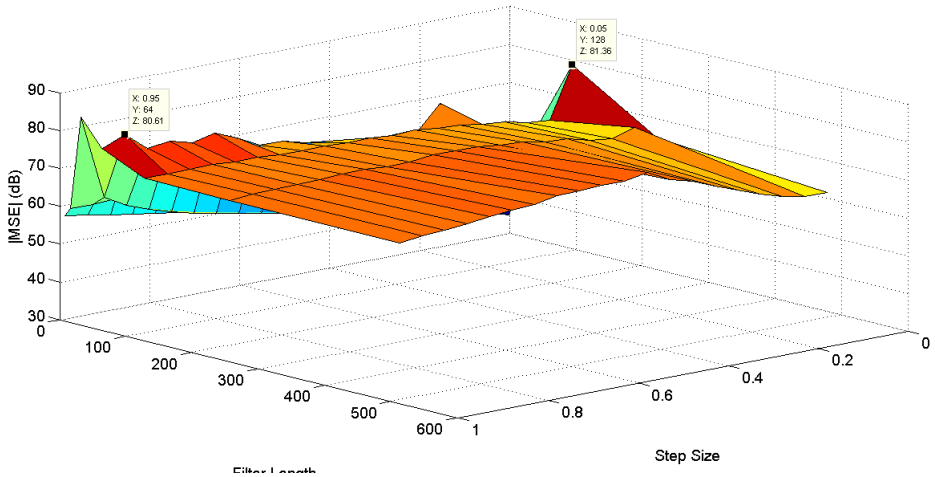

Figura 3.35 - BNDRLMS: Modelo de Percurso de Eco 5.

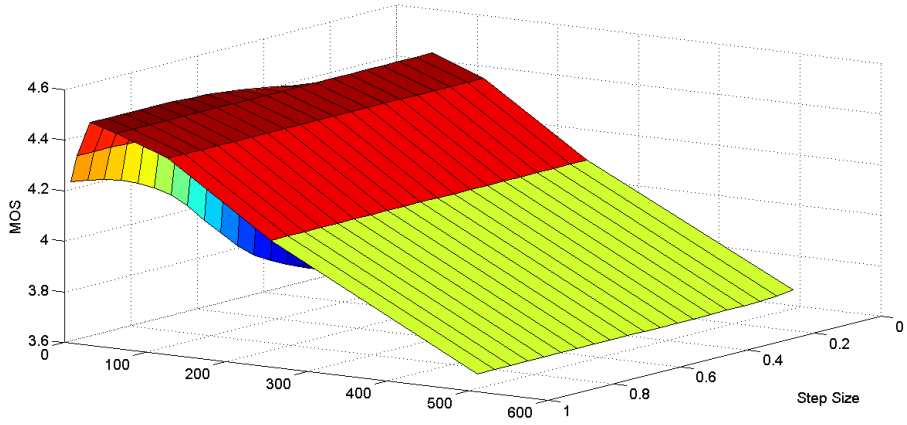

Figura 3.36 - MOS BNDRLMS: Modelo de Percurso 5.

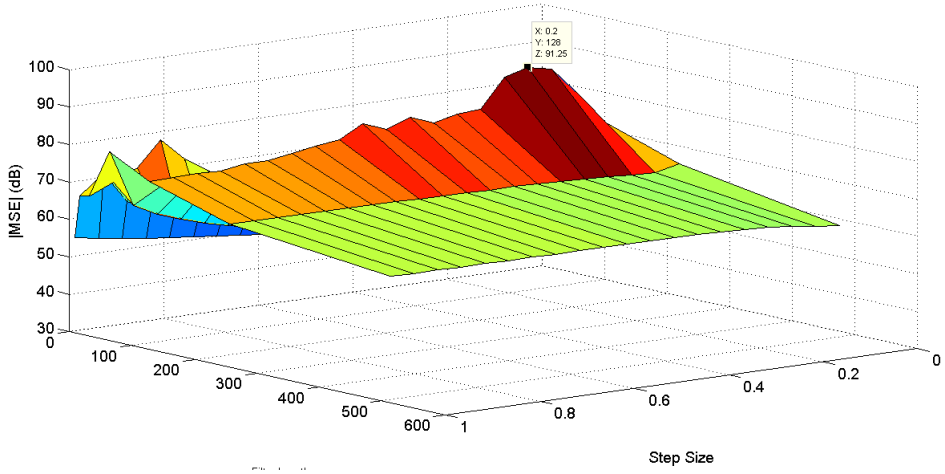

Figura 3.37 - BNDRLMS: Modelo de Percurso de Eco 6.

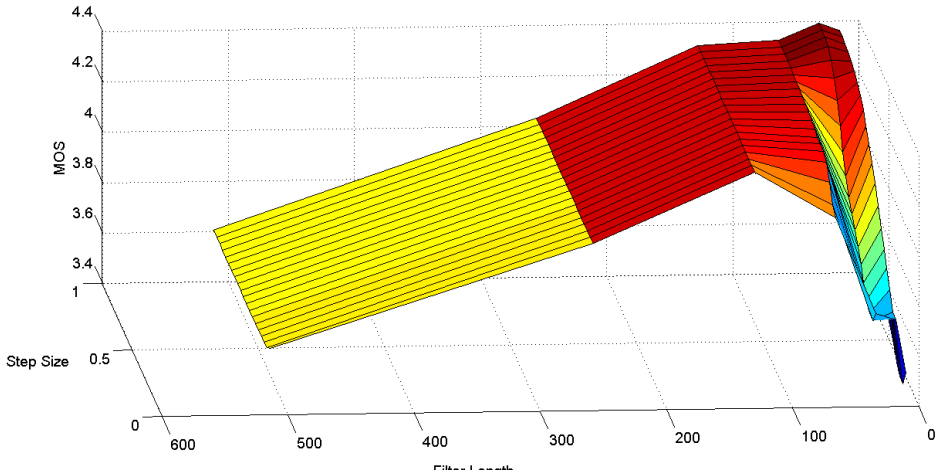

Figura 3.38 - MOS BNDRLMS: Modelo de Percurso 6.

O algoritmo BNDRLMS resolve o problema da sensibilidade em relação ao passo de adaptação escolhido para o NDRLMS. Com uma curva similar à encontrada no NLMS, o BNDRLMS apresenta melhor desempenho na minimização do erro médio quadrático. Para todos os canais simulados os melhores resultados são obtidos com ordem de filtro de até 128 amostras.

A reutilização de dados em conjunto com a binormalização levam a uma resposta mais robusta a nível de sensibilidade de parâmetros, mas inferior se comparada com a versão NDRLMS. Em todos os canais simulados a resposta se encontra entre 60 e 80 dB

O cruzamento de dados entre as métricas revela uma característica importante deste algoritmo: a qualidade percebida do áudio é muito afetada pela diferença entre a ordem do filtro e o tamanho da resposta ao impulso do canal. Para todos os canais simulados, o MOS médio obtido varia suavemente entre 3,6 e 4,4. Uma possível explicação para isto seria o fato do erro quadrático médio remanescente apresentar uma dispersão menor ao longo do sinal analisado, fazendo com que fosse mais percebido na avaliação de qualidade.

3.5.6 – Desempenho do Algoritmo PBNDRLMS

O desempenho do algoritmo PBNDRLMS de acordo com a variação de seus parâmetros pode ser visto nas figuras 3.39 a 3.44. Onde as figuras 3.39 e 3.40 apresentam o desempenho em minimização do MSE e o desempenho em MOS para o percurso de eco 1, as figuras 3.41 e 3.42 apresentam os mesmos experimentos para o percurso de eco 5 e as figuras 3.43 e 3.44 apresentam os mesmos experimentos para o percurso de eco 6.

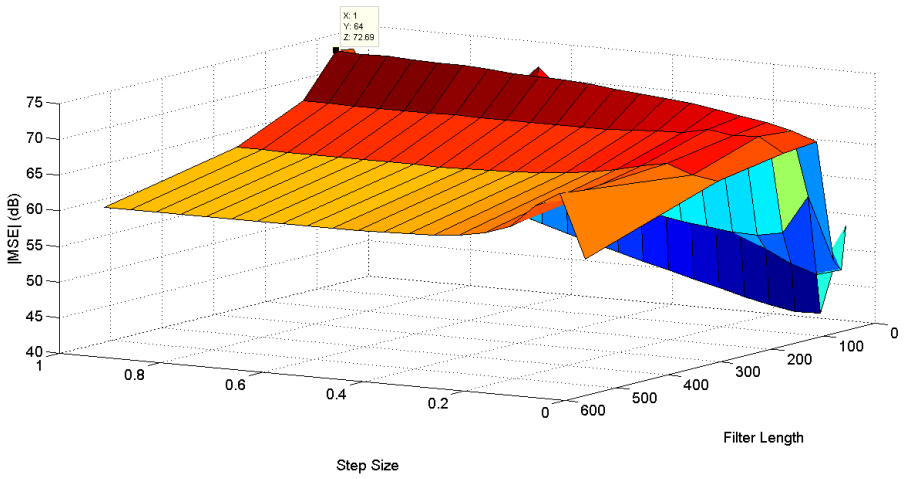

Figura 3.39 - PBNDRLMS: Modelo de Percurso de Eco 1.

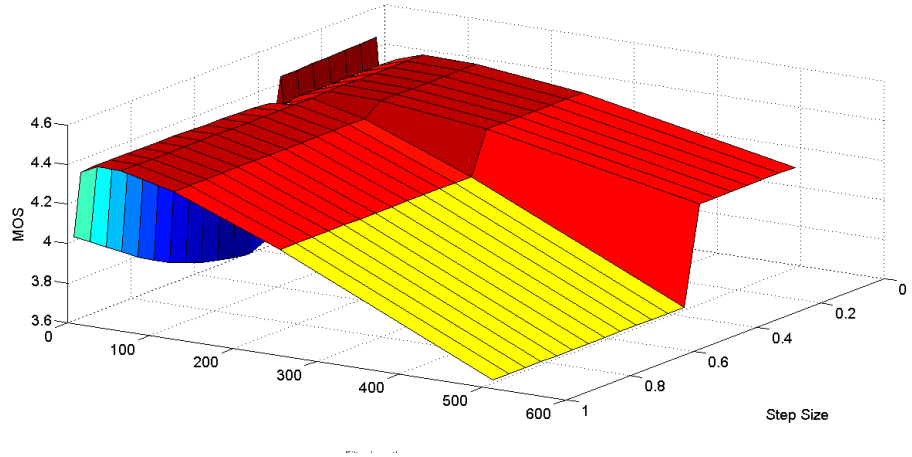

Figura 3.40 - MOS PBNDRLMS: Modelo de Percurso 1.

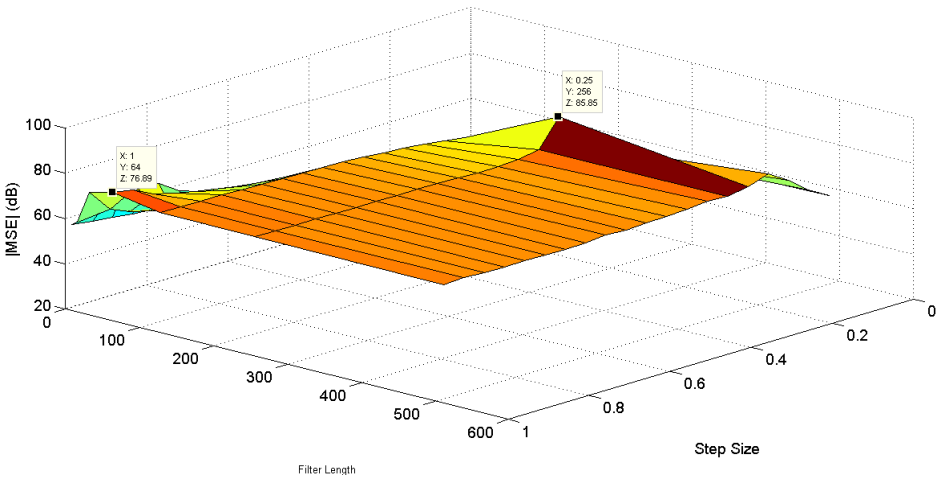

Figura 3.41 - PBNDRLMS: Modelo de Percurso de Eco 5.

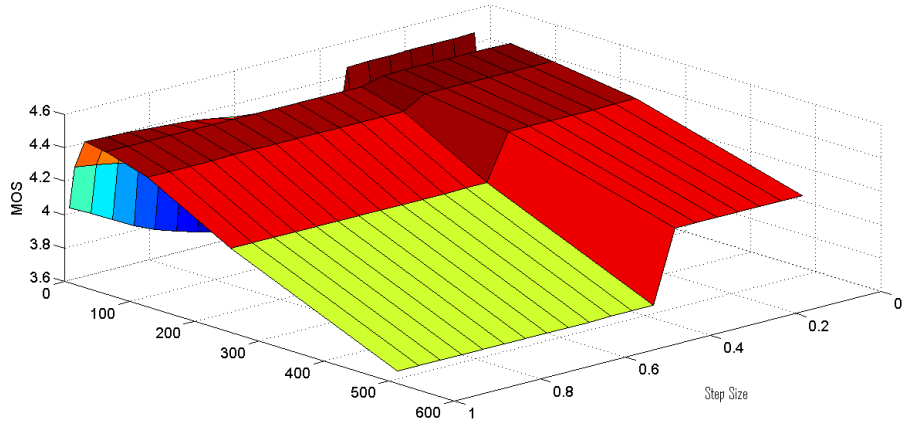

Figura 3.42 - MOS PBNDRLMS: Modelo de Percurso 5.

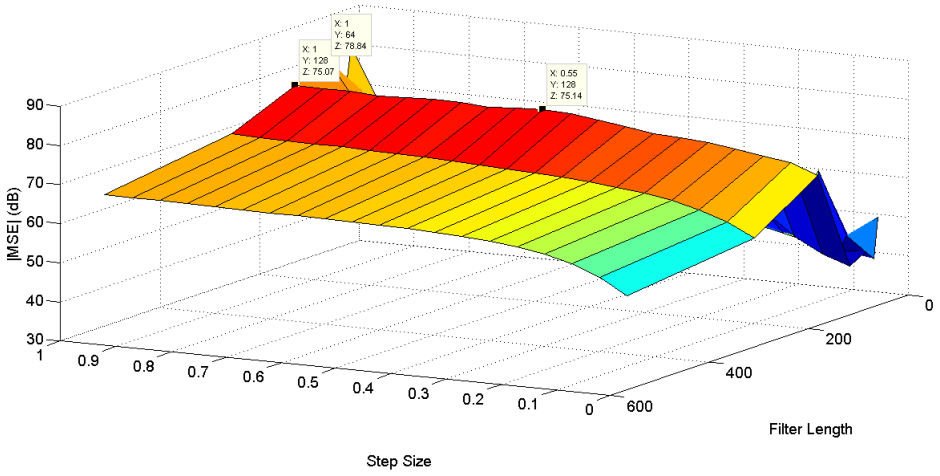

Figura 3.43 - PBNDRLMS: Modelo de Percurso de Eco 6.

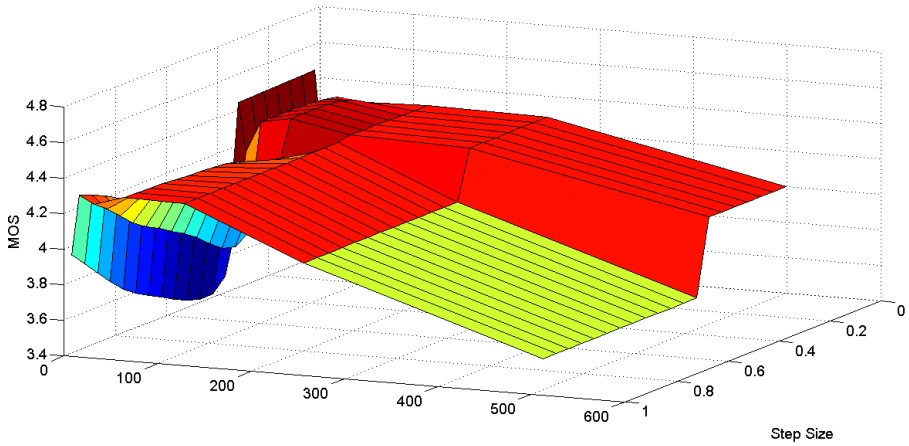

Figura 3.44 - MOS PBNDRLMS: Modelo de Percurso 6.

Conforme o esperado, o algoritmo PBNDRLMS apresenta o mesmo comportamento geral do BNDRLMS. Para todos os canais simulados os melhores resultados são obtidos com ordem de filtro de até 256 amostras.

A componente proporcional influencia apenas a velocidade de convergência e a estabilização da minimização do erro quadrático médio se dá em níveis equivalentes aos apresentados pelo algoritmo BNDRLMS em todos os canais simulados (entre 60 e 80 dB).

O comportamento no cruzamento de dados entre as métricas é similar ao algoritmo anterior: a qualidade percebida do áudio é muito afetada pela diferença entre a ordem do filtro e o tamanho da resposta ao impulso do canal. Para todos os canais simulados, o MOS médio obtido varia suavemente entre 3,6 e 4,4.

3.6 - Comparativo

A tabela abaixo traz de maneira resumida um comparativo entre os algoritmos estudados neste capítulo.

Tabela 3.3 - Comparativo Técnico englobando a complexidade computacional e os desempenhos quantitativos e qualitativos de todos os algoritmos testados.

Algoritmo	Complexidade Computacional	Desempenho MSE (dB)	Desempenho MOS
LMS	N + 1	60	4,0
NLMS	2N + 1	65	4,1
PNLMS	3N + 1	65	4,1
NDRLMS	5N + 2	90	4,2
BNDRLMS	6N + 8	80	4,3
PBNDRLMS	8N + 9	80	4,3

Face aos resultados apresentados, excetuando casos em que a velocidade de convergência é fundamental, o algoritmo mais adequado para uma implementação livre seria o PNLMS, uma vez que se apresenta mais robusto a variações dos parâmetros do filtro e possui baixa complexidade computacional. Para os casos em que a velocidade de convergência é importante, o algoritmo mais adequado seria o BNDRLMS, pois possui rápida convergência e complexidade computacional média dentre os algoritmos apropriados para a situação (NDRLMS, BNDRLMS e PBNDRLMS). Vale ressaltar, entretanto, que a aplicação dos algoritmos baseados em reutilização de dados é mais apropriada quando se tem algum conhecimento *a priori* da ordem de grandeza da

resposta ao impulso do canal a ser utilizado, pois todos apresentam elevada sensibilidade em relação a diferenças entre a ordem do filtro adaptativo e a resposta ao impulso do canal na avaliação qualitativa do áudio (MOS).

Para uma implementação com restrições, onde a complexidade computacional tem um peso elevado no desempenho do sistema, o algoritmo mais adequado seria o NLMS, pois apresenta um desempenho qualitativo (MOS) equivalente ao PNLMS com uma complexidade computacional inferior.

3.7 - Conclusões

Este capítulo apresentou conceitos importantes para o cancelamento livre de eco de linha, levando o leitor a considerar aspectos práticos na escolha da estratégia de adaptação. As seções 3.4 e 3.5 apresentaram um estudo de otimização de parâmetros inovador, onde foram levados em conta aspectos quantitativos e qualitativos para a escolha dos parâmetros ótimos.

O cruzamento de dados quantitativos e qualitativos proporciona riqueza na avaliação, uma vez que salienta características não visíveis no estudo quantitativo. Este estudo mostrou que mais importante do que ter um erro quadrático médio pequeno é ter um erro quadrático médio pequeno e disperso, pois a dispersão do erro é fator preponderante na avaliação perceptual da qualidade do áudio.

Em ambientes cuja velocidade de convergência não é determinante, os algoritmos PNLMS e NLMS com o passo de adaptação entre 0,2 e 0,5 se apresentaram como melhores soluções para cancelamento livre e restrito, respectivamente. Já em ambientes onde a velocidade de convergência é prioritária, o algoritmo recomendado é o BNDRLMS.

Capítulo 4 - Cancelamento de Eco Elétrico com Restrições

4.1 - Introdução

Neste capítulo será discutido o desenvolvimento com restrições de um cancelador de eco elétrico. Para isto, será considerado o algoritmo que obteve o melhor desempenho na etapa de cancelamento livre descrita no capitulo anterior, limitando as simulações a apenas um dos modelos de canal apresentados na recomendação ITU-T G.168 [12].

Para isto, a Seção 4.2 explicita os diferentes tipos de restrições presentes em aplicações embarcadas e suas consequências para o desempenho final dos algoritmos. São eles: precisão finitae aritmética em ponto-fixo.

Na Seção 4.3, são apresentadas as ferramentas disponíveis para a simulação dos ambientes restritos em questão, visando esclarecer a utilização de todos os recursos necessários à simulação de um sistema embarcado real.

Na Seção 4.4, é abordada a criação de um filtro NLMS em ponto-fixo, visando estudar seu desempenho para diferentes resoluções de bits.

Na Seção 4.5, o desempenho do filtro NLMS em ponto-fixo é comparado ao do NLMS comum, de forma a determinar se o algoritmo em ponto-fixo é recomendado para uma implementação em um sistema embarcado real.

A Seção 4.6, por sua vez, conclui o presente capítulo, sumarizando os assuntos nele abordados.

4.2 - Tipos de Restrição

A seguir serão descritos os tipos de restrições normalmente encontradas em sistemas reais.

4.2.1 – Precisão Finita

A precisão finita é uma das restrições mais encontradas em aplicações reais, pois sempre que se utilizam conversores analógico-digitais erros de quantização

inversamente proporcionais à quantidade de bits do conversor são introduzidos ao sinal de entrada.

Esta também passa a ser problemática em aplicações onde se fazem necessárias representações de números maiores do que o suportado pelo sistema, o que pode acarretar em truncamentos, saturações e perda de desempenho. Os efeitos da representação finita podem ser minimizados, porém nunca eliminados.

Os truncamentos são bastante comuns em operações de multiplicação de ponto-fixo, uma vez que a multiplicação de dois números de n bits resulta em um número de 2n bits. Este é o motivo pelo qual todos os processadores de ponto-fixo possuem registradores de produto e acumulação com o dobro do tamanho dos outros. Quando a multiplicação termina, o resultado contido pelo registrador de multiplicação com 2n bits deverá ser alocado em um registrador de n bits, o que acarretará em truncamento do resultado ou a utilização de mais um registrador para alocar o restante onerando o desempenho do sistema (dois ciclos de máquina). Ou seja, sempre haverá um compromisso entre erro de representação e perda de desempenho.

As saturações também ocorrem com bastante frequência, uma vez que a maioria dos algoritmos de processamento digital requer sucessivas adições e multiplicações. O resultado acumulado pode necessitar uma quantidade de bits superior à resolução interna do sistema para ser representado e então a saturação ocorre. Os erros de saturação não são muito importantes em filtros FIR, uma vez que o erro somente acontecerá para uma saída. Já para os filtros IIR este erro é realimentado o que pode causar erros ainda maiores nas saídas subsequentes.

Para minimizar estes erros alguns processadores suportam o modo saturação, onde resultados maiores do que a resolução máxima da arquitetura serão sempre representados com todos os bits em 1 e resultados menores serão sempre representados com todos os bits em 0.

4.2.2 – Aritmética de Ponto-Fixo

Os efeitos de todos os erros citados anteriormente dependem diretamente do tipo de aritmética utilizada no sistema objetivo.

Caso a aritmética de ponto-flutuante esteja disponível, os efeitos percebidos serão desprezíveis para a grande maioria das aplicações. A representação em ponto-flutuante

tem uma faixa dinâmica muito maior do que a representação em ponto-fixo, o que faz com que os erros tenham pouco impacto no resultado final.

Caso contrário, vários cuidados devem ser tomados de forma a minimizar os efeitos dos erros, como por exemplo, o controle de precisão de representação e saturação. Estas estratégias serão discutidas posteriormente.

4.3 – Aritmética de Ponto-Fixo no Matlab

O Matlab possui um extenso conjunto de funções e objetos destinados ao processamento em ponto-fixo chamado *Fixed-Point Designer*. Este trabalho se limitará a descrever as principais características desse pacote, salientando os detalhes mais importantes para a implementação do algoritmo em questão.

4.3.1 – Objetos fi, fimath e fipref [15]

O objeto responsável por representar números em ponto-fixo no Matlab é o *fi*. Basicamente o usuário pode criar objetos *fi* de quatro maneiras, a saber:

- Construtor *fi* para criar um novo objeto *fi*;
- Construtor *sfi* para criar um novo objeto *fi* sinalizado;
- Construtor *ufi* para criar um novo objeto *fi* não sinalizado;
- Qualquer dos construtores anteriores para copiar um objeto *fi* existente.

O objeto *fi* sempre está associado a um objeto *fimath*, cuja função é determinar propriedades aritméticas relacionadas com o objeto *fi* tais como tamanho máximo para representação de multiplicação, tamanho máximo para representação de adição, estratégia de saturação, método de arredondamento etc.

O objeto *fipref* por sua vez, define as preferências de apresentação e log de todos os objetos *fi*, ou seja, o usuário pode utilizá-lo para alterar a apresentação dos números na tela para binário utilizando mais ou menos bits, pode obter um log de todos os *overflows* e *underflows* ocorridos na última simulação etc. Vale ressaltar que o valor de *fipref* permanece inalterado durante toda a seção atual do usuário, e, portanto, as preferências serão consideradas em todos os algoritmos rodados nela. O usuário pode resetar ou salvar como padrão as preferências configuradas com os comandos *reset(fipref)* e *savefipref*.

A seguir serão descritas as principais propriedades do objeto *fi*. Note que mesmo se tratando de objetos separados, as propriedades de *fimath* e *numerictype* também são propriedades do objeto *fi* [16].

- **bin:** valor do número inteiro guardado no objeto *fi* em formato binário
- **data:** valor real guardado no objeto *fi*
- **dec:** valor do número inteiro guardado no objeto *fi* em formato decimal
- **double:** valor real guardado no objeto *fi* em formato MATLAB double
- **fimath: o**s objetos fi retiram suas propriedades aritméticas de um objeto *fimath* local ou de valores padrões do sistema. Os valores padrões são:
  RoundingMethod: Nearest
  OverflowAction: Saturate
  ProductMode: FullPrecision
  SumMode: FullPrecision
  As propriedades e valores do objeto *fimath* serão apresentadas mais a frente.
- **hex: v**alor do número inteiro guardado no objeto *fi* em formato hexadecimal
- **NumericType:** o objeto *numerictype* contém todos os tipos de dados e propriedades de escalamento de um objeto *fi.* Vale ressaltar que o usuário não pode alterar as propriedades *numeritype* de um objeto *fi* depois de sua criação.
- **oct:** Valor do número inteiro guardado no objeto *fi* em formato octal

A seguir serão descritas as principais propriedades do objeto *fimath* [16]:

- **CastBeforeSum:** determina se os dois operadores serão transformados no tipo de dado do resultado da soma antes da soma ser realizada ou não. Os valores possíveis para esta propriedade são 1 (*true*) ou 0 (*false*). O valor padrão do sistema para esta propriedade é *true*.
- **MaxProductWordLength:** tamanho máximo em bits para o tipo de dado utilizado no produto de números. O valor padrão para esta propriedade é 65535.
- **MaxSumWordLength:** tamanho máximo em bits para o tipo de dado utilizado na soma de números. O valor padrão para esta propriedade é 65535.
- **OverflowAction:** define a estratégia de *overflow*. Os valores desta propriedade podem ser:
  Saturate – satura para o maior ou menor valor da faixa de representação do número.

Wrap – representa em complemento de 2, ou seja, estoura a escala e apresenta o número sem o MSB.

O valor padrão para esta propriedade é *Saturate*.

- **ProductMode:** define como o tipo de dado do produto é estabelecido. Considere A e B operadores reais com pares [*word length,fraction length*] [W_aF_a] e [W_bF_b] respectivamente, e [W_pF_p] *word length* e *fraction length* do tipo de dado do produto. Os valores desta propriedade podem ser:

  FullPrecision – Neste modo a precisão máxima é mantida e um erro é gerado se o tamanho máximo (w*ord length*) calculado for maior do que *MaxProductWordLength*.

$$W_p = W_a + W_b$$

$$F_p = F_a + F_b$$

  KeepLSB – Guarda os bits menos significativos. Na prática o usuário especificará o tamanho de palavra para o produto (*ProductWordLength*), enquanto o tamanho de palavra para a parte fracionária será calculada para manter os bits menos significativos. Neste modo a precisão máxima (*full precision*) é mantida, mas *overflows* são possíveis. Este modo modela o comportamento das operações com inteiros em linguagem C.

$$W_p = especificado\ na\ propriedade\ ProductWordLength$$

$$F_p = F_a + F_b$$

  KeepMSB – Guarda os bits mais significativos. Na prática o usuário especificará o tamanho de palavra para o produto (*ProductWordLength*), enquanto o tamanho de palavra para a parte fracionária será calculada para manter os bits mais significativos. Neste modo não ocorrem *overflows,* mas a precisão pode ser afetada.

$$W_p = especificado\ na\ propriedade\ ProductWordLength$$

$$F_p = W_p - Tamanho\_inteiro$$

  Onde,

$$Tamanho\_inteiro = (W_a + W_b) - (F_a - F_b)$$

  SpecifyPrecision – O usuário especifica tanto o tamanho de palavra quanto o tamanho da parte fracionária.

$$W_p = especificado\ na\ propriedade\ ProductWordLength$$

$$F_p = especificado\ na\ propriedade\ ProductFractionalLength$$

O valor padrão do sistema para esta propriedade é *FullPrecision*.

- **ProductWordLength:** tamanho em bits do tipo de dado do produto de dois objetos *fi*. Este valor deve ser um inteiro positivo.
  O valor padrão desta propriedade é 32.
- **RoundingMethod:** define o método de arredondamento a ser utilizado. . Os valores possíveis para esta propriedade são:
  Ceiling – arredonda no sentido positivo
  Convergent – arredonda para o mais próximo. Empates são arredondados para o inteiro par mais próximo.
  Zero – arredonda para zero
  Floor – arredonda no sentido negativo
  Nearest – arredonda para o mais próximo. Empates são arredondados no sentido positivo.
  Round – arredonda para o mais próximo. Empates são arredondados no sentido negativo para números negativos e no sentido positivo para números positivos.
  O valor padrão do sistema para esta propriedade é *Nearest*.
- **SumMode:** análogo ao *ProductMode.*
- **SumWordLength:** análogo ao *ProductWordLength.*

A principal propriedade do objeto *fipref* segue descrita abaixo [16]:

- **LoggingMode:** determina as opções de log para as operações realizadas com objetos *fi*. Os valores possíveis para esta propriedade são:
  off – log desligado
  on – log ligado
  Caso a propriedade *LoggingMode* estiver ativa, os *overflows* e *underflows* ocorridos em operações de atribuição de valor, soma, diferença e multiplicação serão guardados e apresentados como *warnings*. O usuário pode utilizar as seguintes funções na linha de comando:
  maxlog – retorna o máximo valor ocorrido
  minlog – retorna o menor valor ocorrido
  noverflows – retorna a quantidade de *overflows* ocorrida
  nunderflows – retorna a quantidade de *underflows* ocorrida

O *LoggingMode* deve ser ativado antes de qualquer operação ser realizada para que esta seja guardada. Para limpar o log, o usuário pode utilizar a função *resetlog*.

O valor padrão para esta propriedade é *off*.

4.3.2 Exemplos

Para que os resultados demonstrados nos exemplos a seguir sejam iguais em qualquer máquina, o usuário deve antes configurar a formatação dos números no matlab utilizando os comandos:

```
format loose
format long g
% Salva as preferências de apresentação e log dos objetos fi
% e as reseta para os seus respectivos valores de fábrica.
fiprefAtStartOfThisExample = get(fipref);
reset(fipref);
```

Veja que o código acima salva as preferências do usuário e as resseta para os padrões de fábrica.

O código abaixo cria dois objetos *fi* **a** e **b** com os atributos padrões do sistema. Note que quando a propriedade *FractionLength* não é especificada, esta é configurada automaticamente para a melhor precisão possível para o tamanho de palavra especificado, mantendo os bits mais significativos do valor. Quando a propriedade *WordLength* não é especificada, seu valor padrão é 16 bits.

```
a = fi(pi)
a = 3.1416015625
DataTypeMode: Fixed-point: binary point scaling
Signedness: Signed
WordLength: 16
FractionLength: 13

b = fi(0.1)
b = 0.0999984741210938
DataTypeMode: Fixed-point: binary point scaling
Signedness: Signed
WordLength: 16
FractionLength: 18
```

O leitor mais atento pode estar se perguntando o porquê de o tamanho alocado para a parte fracionária do objeto *fi* **b** ser maior do que o tamanho total da variável. Isto ocorre porque o Matlab grava o valor das variáveis de forma escalada por $2^{-\text{FractionLength}}$. Este conceito será mais bem elucidado no exemplo seguinte.

No listagem abaixo, compare o valor do parâmetro *FractionLength* das variáveis **a** e **b**:

```
a = sfi(10,16,0)
a = 10
DataTypeMode: Fixed-point: binary point scaling
Signedness: Signed
WordLength: 16
FractionLength: 0


b = sfi(10,16)
b = 10
DataTypeMode: Fixed-point: binary point scaling
Signedness: Signed
WordLength: 16
FractionLength: 11
```

Note que os valores guardados pelo Matlab das variáveis **a** e **b** são diferentes, ainda que elas representem o mesmo valor real. Conforme dito no exemplo anterior, isto ocorre porque o matlab grava o valor das variáveis de forma escalada por $2^{-\text{FractionLength}}$. Ou seja, no caso da variável **a** em que se especificou o valor de *FractionLength* em zero, o valor guardado pelo Matlab foi escalado em $2^0 = 1$, já no caso da variável **b** o valor de *FractionLength* calculado para se manter a melhor precisão foi 11 e o valor guardado pelo Matlab foi escalado por $2^{-11} = 0.00048828125$. O valor guardado pelo Matlab pode ser visto utilizando a função *storedInteger* ou acessando a propriedade *dec* do objeto *fi*:

```
storedInteger(a)
ans = 10


a.dec
```

```
ans = 10

storedInteger(b)
ans = 20480

b.dec
ans = 20480
```

Veja que o valor real de **a** e **b** pode ser calculado como:

$$a = 10 \times 2^{0} = 10$$

$$b = 20480 \times 2^{-11} = 20480 \times 0.00048828125 = 10$$

Até o momento os parâmetros foram especificados passando argumentos ao construtor do objeto *fi*. O usuário pode especificar os parâmetros informando o nome em uma *string* e o valor:

```
a = fi(pi,'WordLength',20)
a = 3.14159393310547
DataTypeMode: Fixed-point: binary point scaling
Signedness: Signed
WordLength: 20
FractionLength: 17
```

Comparativo *Floating-Point x Fixed-Point*: Filtro de segunda ordem

Neste exemplo, um filtro de segunda ordem será desenvolvido e simulado em *float-point* com precisão dupla. Posteriormente, será demonstrada a melhor forma de se portar o código para uma aplicação *fixed-point*, instrumentando o código desenvolvido para que se possa avaliar a faixa dinâmica das variáveis. De posse disso, o algoritmo será alterado para *fixed-point* e o resultado final será comparado. O código completo pode ser encontrado no Anexo A desta dissertação.

Algoritmo em Floating-Point

```
% Algoritmo em Floating-Point
b = [ 0.25 0.5 0.25 ]; % coeficientes do numerador
a = [ 1 0.09375 0.28125 ]; % coeficientes do denominador
```

```
% Sinal de entrada com frequências altas e baixas
x = randn(1000,1);
% Pre-alocar variável para o sinal de saída e estado para acelerar a simulação
y = zeros(size(x));
z = [0;0];
%y(n) = b(1)*x(n) + b(2)*x(n-1) + b(3)*x(n-2) - a(2)*y(n-1) - a(3)*y(n-2)
for k=1:length(x)
        y(k) = b(1)*x(k) + z(1);
        z(1) = (b(2)*x(k) + z(2)) - a(2)*y(k);
        z(2) = b(3)*x(k) - a(3)*y(k);
end
% Salva o resultado em Floating-Point
ydouble = y;
```

Instrumentação do código

Para se converter corretamente o código acima para *fixed-point* é necessário conhecer a faixa dinâmica das variáveis envolvidas. Dependendo da complexidade do algoritmo, esta tarefa pode ser fácil ou muito difícil. Neste exemplo, a faixa dinâmica do sinal de entrada é conhecida, então escolher a faixa dinâmica das variáveis *fixed-point* é simples. Para ver a faixa dinâmica da saída (y) e dos estados (z), o código será modificado incluindo dois objetos *NumericTypesScope*.

```
b = [ 0.25 0.5 0.25 ]; % coeficientes do numerador
a = [ 1 0.09375 0.28125 ]; % coeficientes do denominador
% Sinal de entrada com frequências altas e baixas
%s = rng; rng(0,'v5uniform');
x = randn(1000,1);
%rng(s);
% Pre-alocar variável para o sinal de saída e estado para acelerar a simulação
y = zeros(size(x));
z = [0;0];
%y(n) = b(1)*x(n) + b(2)*x(n-1) + b(3)*x(n-2) - a(2)*y(n-1) - a(3)*y(n-2)
hscope1 = NumericTypeScope;
hscope2 = NumericTypeScope;
```

```
for k=1:length(x)
        y(k) = b(1)*x(k) + z(1);
        z(1) = (b(2)*x(k) + z(2)) - a(2)*y(k);
        z(2) = b(3)*x(k) - a(3)*y(k);
        % processa os dados e atualiza o scope
        step(hscope1,z);
end
step(hscope2,y);
% Salva o resultado em Floating-Point
ydouble = y;
```

Avaliação das Faixas Dinâmicas

Conforme esperado, ao rodar o código acima, dois *NumericTypeScope* com toda a informação estatística das variáveis são apresentados. Seus conteúdos são reproduzidos nas figuras 4.1 e 4.2 por conveniência.

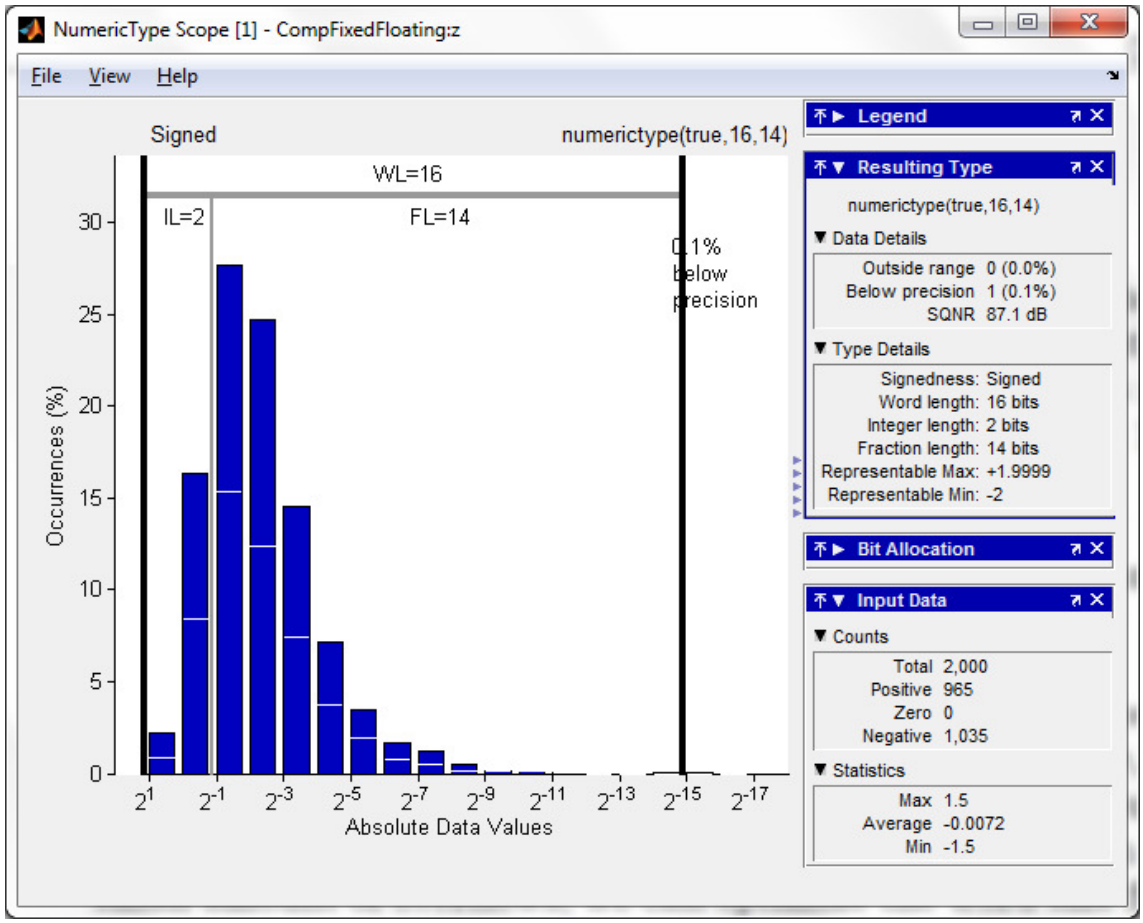

Figura 4.1 - *NumericTypeScope* dos estados (z): A faixa dinâmica da variável pode foi avaliada e a precisão ótima escolhida foi a de 14 bits para a parte fracionária assumindo erro de 0,1%.

Analisando o histograma da variável de estados (z), pode-se ver que a faixa dinâmica vai de 2^1 a aproximadamente 2^{-20}. No painel *Input Data* pode-se ver que os dados podem ser positivos ou negativos, e seus valores máximo e mínimo são 1.5 e -1.5, respectivamente. Como o *scope* utiliza variáveis de 16 bits sem tolerância a *overflows* por padrão, o tipo de dado sugerido é *numerictype(true,16,14)* uma vez que se faz necessário pelo menos um bit a mais para suportar possíveis *overflows*(serão utilizados dois bits para a representação da parte inteira).Utilizando o tipo de dado sugerido, valores que requerem mais de 14 bits para a parte fracionária para serem representados causarão *underflows*. No caso em questão, este universo se resume a 0.1% e pode ser desprezado.

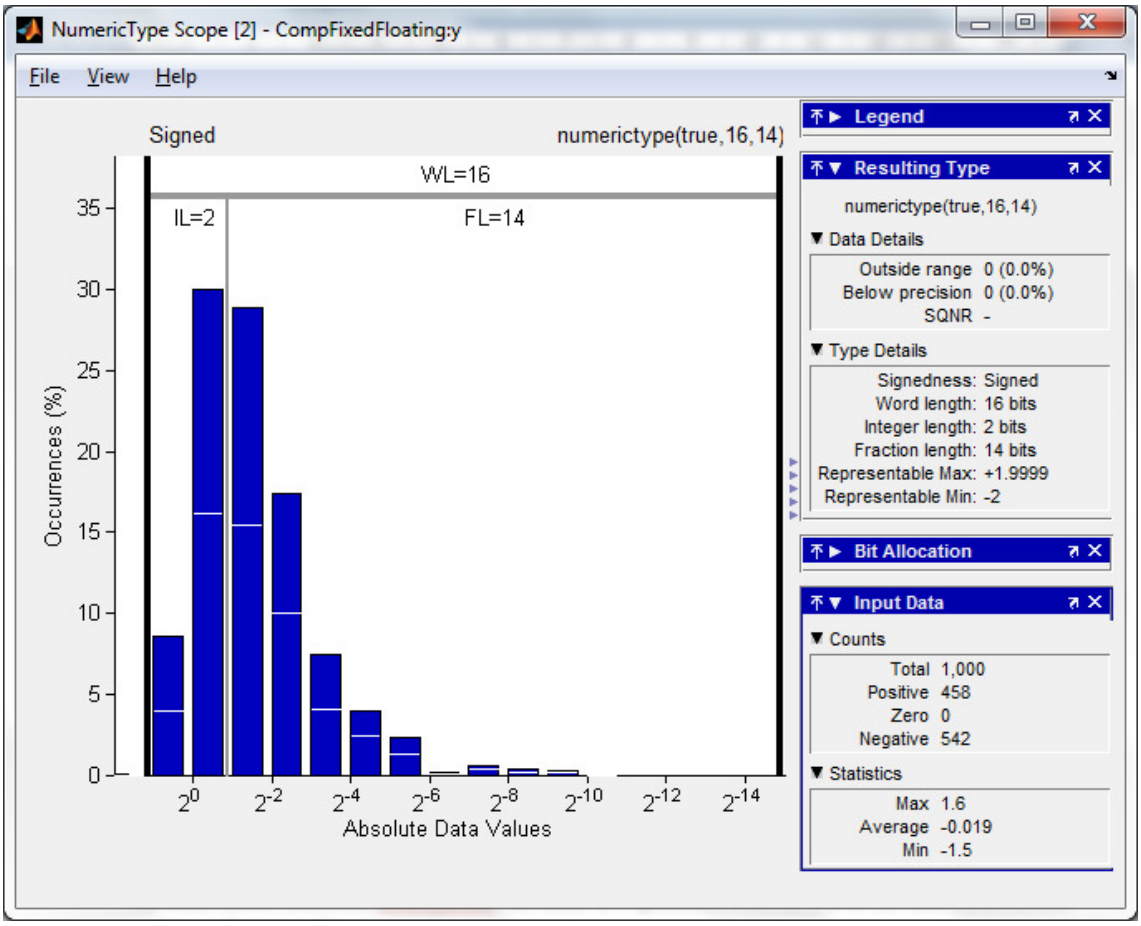

Figura 4.2 - NumericTypeScope da saída (y). A faixa dinâmica da variável pode foi avaliada e novamente a precisão ótima escolhida foi a de 14 bits para a parte fracionária sem qualquer margem de erro.

Realizando uma análise semelhante à anterior para o sinal de saída, vemos que a faixa dinâmica se encontra entre 2^{2} e 2^{-11}. Isto resulta no tipo de dado *numerictype(true,16,13)*, uma vez que é necessário ao menos 1 bit a mais para suportar possíveis *overflows*. Com este tipo de dado não haverão *overflows* nem *underflows*, uma vez que toda a faixa dinâmica da variável será contemplada. Caso a aplicação seja tolerante a uma pequena quantidade de *overflows*, o tamanho do tipo pode ser otimizado configurando o parâmetro *Maximum Overflow* no painel *Bit Allocation* para 0.5%, por exemplo.

Conversão para Fixed-Point

Agora a conversão das variáveis envolvidas pode ser realizada de forma correta, mas antes o log será ativado para que se possa ver todos os *overflows* e *underflows* ocorridos.

```
% Ativa o logging para ver overflows/underflows.
FIPREF_STATE = get(fipref);
```

```
reset(fipref)
fp = fipref;
default_loggingmode = fp.LoggingMode;
fp.LoggingMode = 'On';
% Grava o estado atual de fimath e reseta para os padrões de fábrica
globalFimathAtStart = fimath;
resetglobalfimath;
% Define as variáveis fixed-point no formato:
% fi(Data, Signed, WordLength, FractionLength)
b = fi(b, 1, 8, 6);
a = fi(a, 1, 8, 6);
x = fi(x, 1, 16, 13);
y = fi(zeros(size(x)), 1, 16, 13);
z = fi([0;0], 1, 16, 14);
for k=1:length(x)
    y(k) = b(1)*x(k) + z(1);
    z(1) = (b(2)*x(k) + z(2)) - a(2)*y(k);
    z(2) = b(3)*x(k) - a(3)*y(k);
end
% Reseta o logging
fp.LoggingMode = default_loggingmode;
```

Comparativo Floating-Point x Fixed-Point

Agora os resultados apresentados nas figuras 4.3 e 4.4 podem ser comparados:

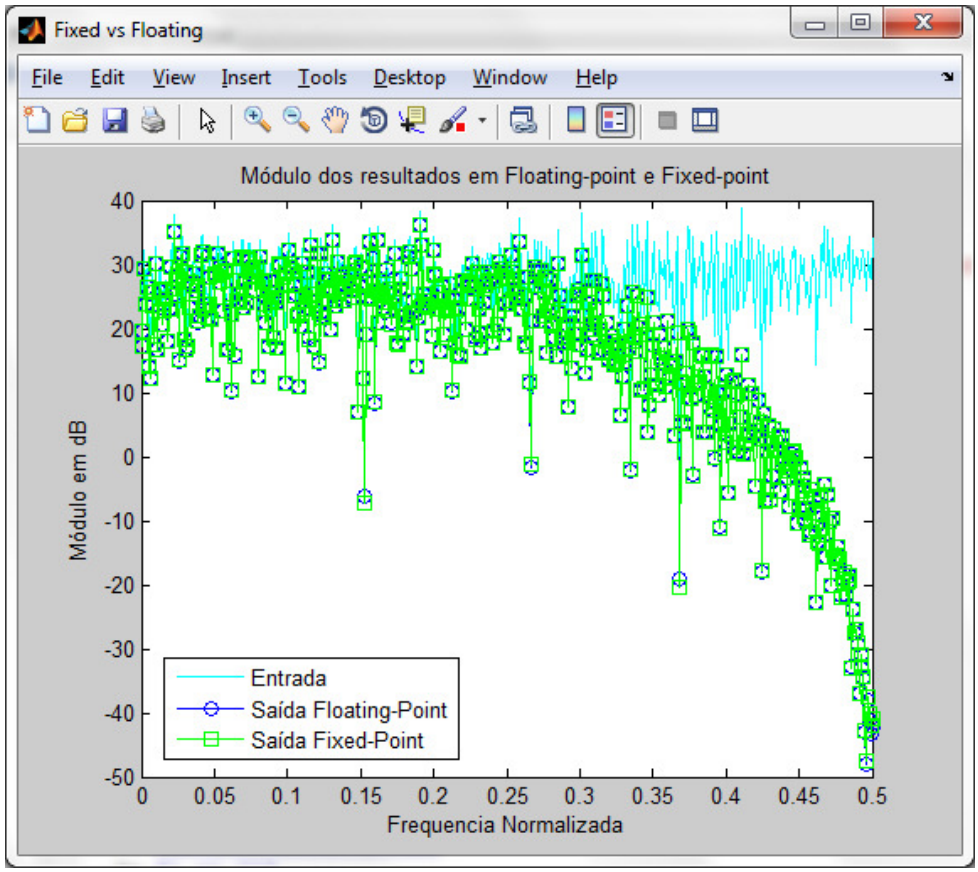

Figura 4.3 - Comparativo *Floating-Point x Fixed-Point* ressaltando a equivalência de desempenhos dos filtros.

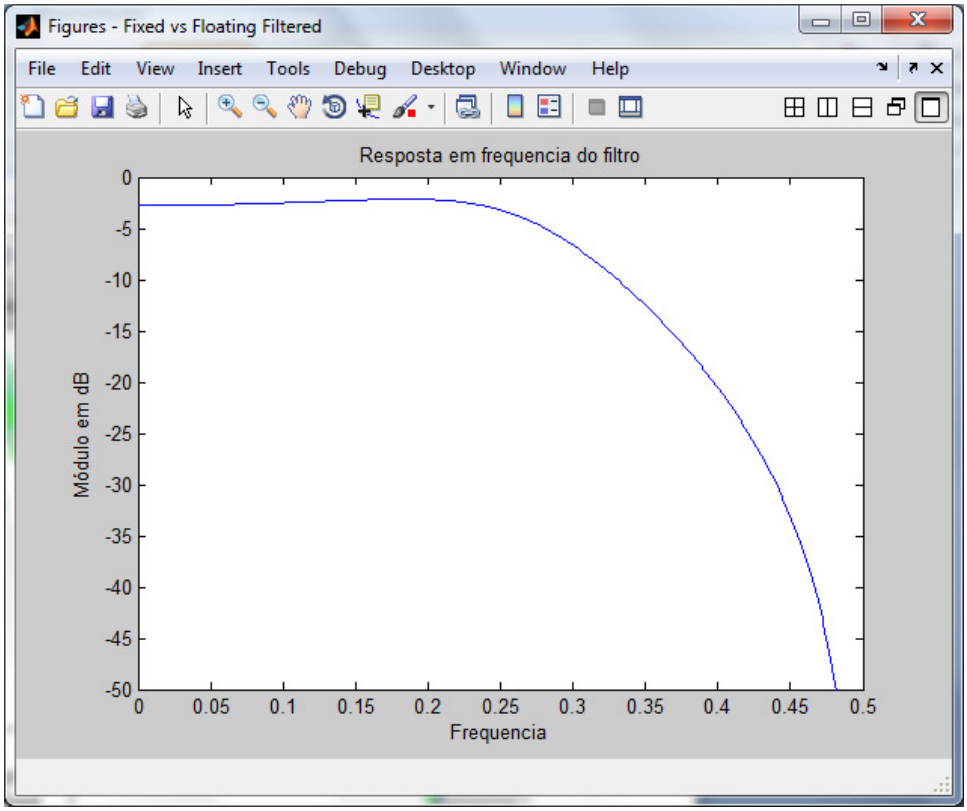

Figura 4.4 - Resposta em Frequência do Filtro: desempenho conforme especificado.

Conforme esperado, a resposta em *fixed-point* é muito próxima à reposta em *floating-point.* Isto ocorre apenas porque ao projetar o filtro em ponto fixo uma análise minuciosa das variáveis envolvidas foi realizada. Isto é comprovado pelo log de *overflows/underflows*:

```
Warning: 1 overflow occurred in the fi assignment operation.
> In embedded.fi.fi at 538
  In fi at 218
  In CompFixedFloating at 52
Warning: 1 underflow occurred in the fi assignment operation.
> In CompFixedFloating at 57
```

4.4 – Fixed-Point NLMS

Nesta seção, o algoritmo NLMS desenvolvido no capítulo anterior será alterado para operar em ponto-fixo e os resultados de sua simulação serão apresentados.

A seguir serão apresentadas as partes relevantes dos scripts Matlab desenvolvidos para a simulação do algoritmo NLMS no sistema alvo.

O algoritmo foi concebido para receber como parâmetro o vetor de entrada (*input*), o valor desejado (*desired*), o vetor de coeficientes (*coeficients*), o passo de adaptação (*step*) e o tamanho em bits das variáveis internas (word_*length*). A função retorna o valor dos coeficientes do filtro atualizado (*w*), o valor estimado pelo filtro (*y_hat*) e o erro entre o valor desejado e a valor estimado (*e*).

```
function [w,y_hat,e] =
fp_nlms_best_precision(input,desired,coeficients,step,word_length)
    fp_input = fi(input,1,word_length);
    fp_desired = fi(desired,1,word_length);
    fp_coeficients = fi(coeficients,1,word_length);
    fp_step = fi(step,1,16);
    fp_delta = fi(0.0001,1,word_length);
    div = fp_input'*fp_input + fp_delta;
    y_hat = fp_coeficients' * fp_input;
    fp_step = fp_step/div;
    e = fp_desired - y_hat
```

```
    w = fp_coeficients + fp_step * fp_input * e
end
```

O algoritmo calcula a resposta do filtro para cada passo de adaptação, o que obriga à existência de um script externo para realizar sucessivas chamadas à função *fp_nlms_best_precision* com os parâmetros corretos.

```
for n = sysorder:N
    u = y(n:-1:n-sysorder+1);
    [w,y_est_NLMS(n),err] = fp_nlms_best_precision(u',d(n),w,0.4,16);
    e_NLMS(n) = err;
end
```

Seu desempenho quantitativo pode ser avaliado a partir das figuras 4.5, 4.6, onde as superfícies de minimização média do MSE são apresentadas. Seu desempenho qualitativo, por sua vez, pode ser estudado a partir das figuras 4.7 e 4.8, onde as superfícies de avaliação perceptual são apresentadas.

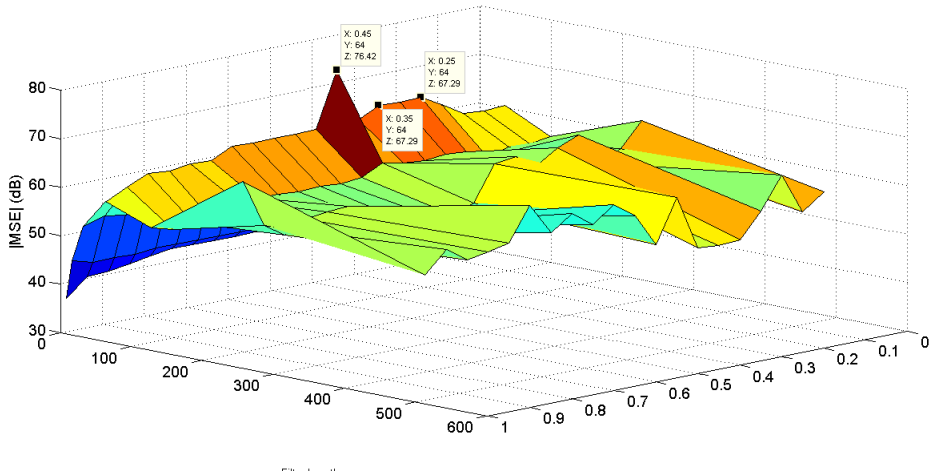

Figura 4.5 – 16 Bits Fixed-Point NLMS: Modelo de Percurso de Eco 1 (Vista 1).

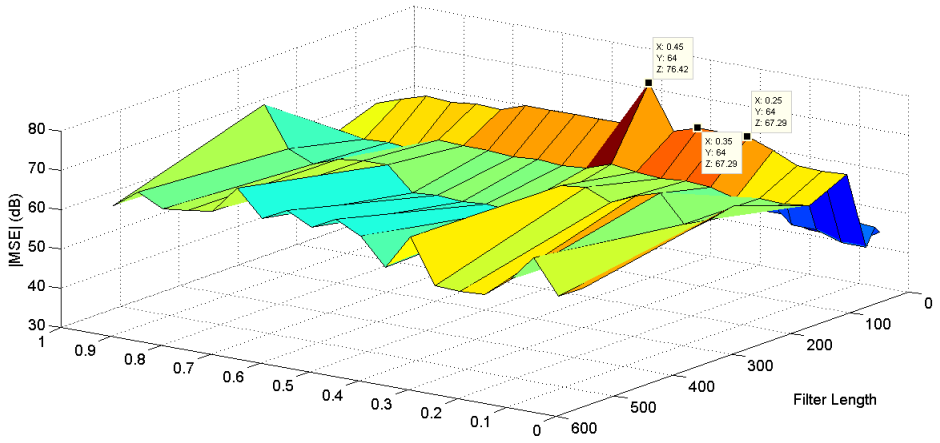

Figura 4.6 - 16 Bits Fixed-Point NLMS: Modelo de Percurso de Eco 1 (Vista 2).

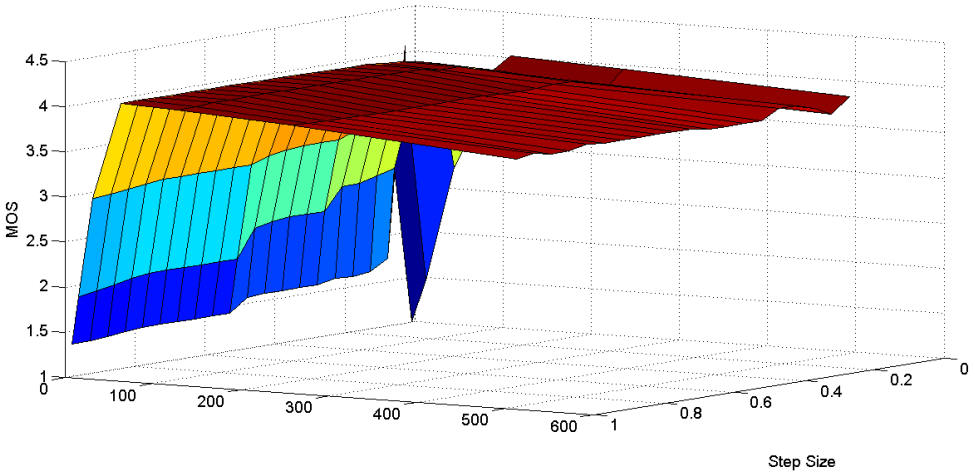

Figura 4.7 - MOS 16 Bits Fixed-Point NLMS: Modelo de Percurso 1 (Vista 1).

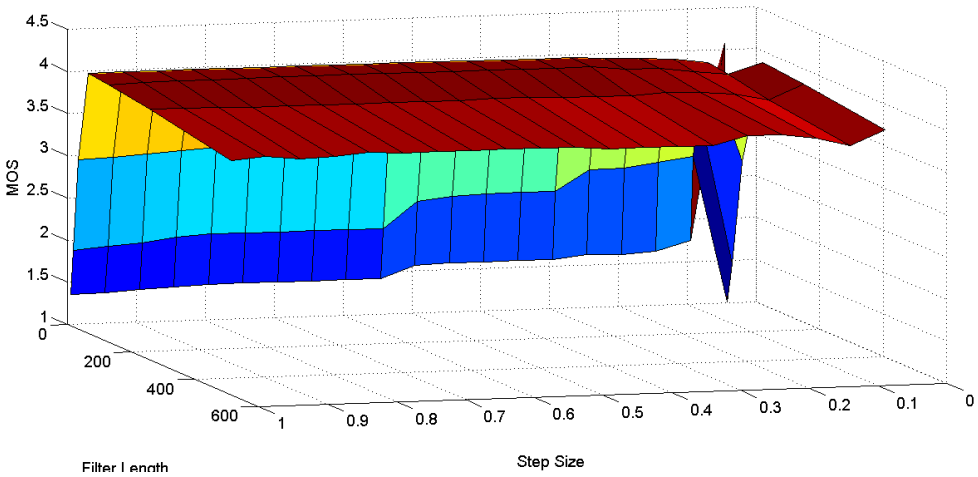

Figura 4.8 - MOS 16 Bits Fixed-Point NLMS: Modelo de Percurso 1 (Vista 2).

Pode-se observar a partir das simulações de parâmetros que o algoritmo em ponto-fixo obteve um excelente desempenho tanto na minimização do MSE quanto na avaliação MOS, considerando que o algoritmo avaliado utiliza variáveis de apenas 16 bits. A média de minimização do MSE está entre 50 e 60 dB enquanto o MOS médio está entre 3,5 e 4,0.

4.4.1 - Estudo de Resolução de Bits

Com o intuito de verificar o impacto da resolução de bits no projeto do filtro *Fixed-Point* NLMS em uma arquitetura de 16 bits, foi conduzido um estudo comparativo em que filtros NLMS com variáveis de 8 a 16 bits em ponto-fixo foram utilizados para o cancelamento de eco.

Foram avaliados o MSE absoluto e o MSE médio e os resultados são mostrados nas figuras 4.9, 4.10 e 4.11, subsequentes.

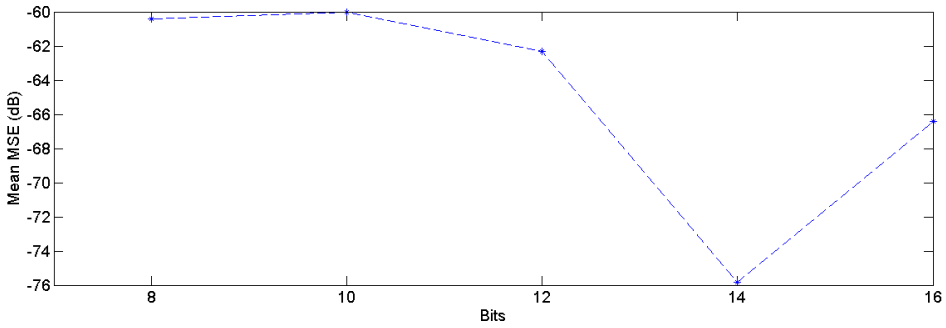

Figura 4.9 - Estudo de Resolução em Bits: Mean MSE.

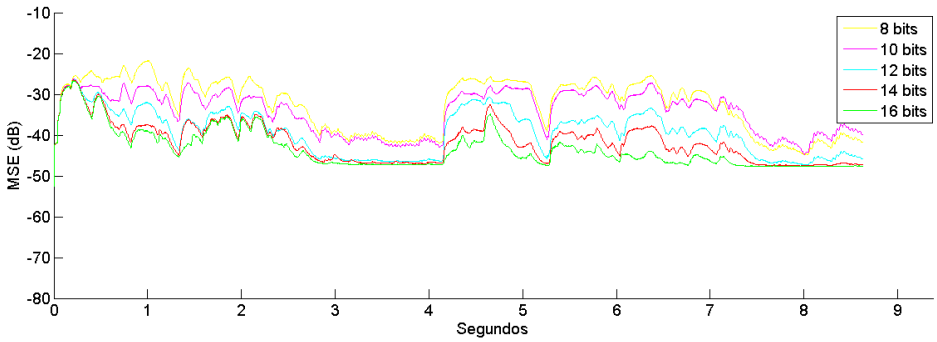

Figura 4.10 - Estudo de Resolução de Bits: MSE.

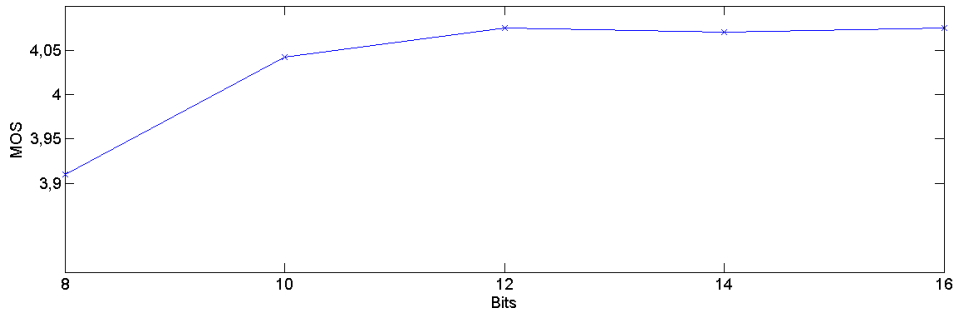

Figura 4.11 - Estudo de Resolução de Bits: MOS.

Como esperado, o filtro com o melhor desempenho foi o que utiliza mesmo tamanho de palavra da arquitetura objetivo (16 bits). Apesar de a média do erro quadrático médio ser a segunda melhor (66 dB), ficando atrás do filtro com 14 bits de resolução (74 dB), o filtro com 16 bits apresenta desempenho absoluto superior ao longo de todas as amostras.

4.5 – Comparativo NLMS x *Fixed-Point* NLMS

Nesta seção, ambos os desempenhos quantitativos e qualitativos dos filtros NLMS e *Fixed-Point* NLMS serão comparados. Somente assim poder-se-á concluir se o desempenho do filtro em ponto-fixo é aceitável para uma possível implementação em um sistema real.

As figuras 4.12 e 4.13 apresentam o desempenho em minimização do MSE para os filtros NLMS e *Fixed-Point* NLMS, respectivamente.

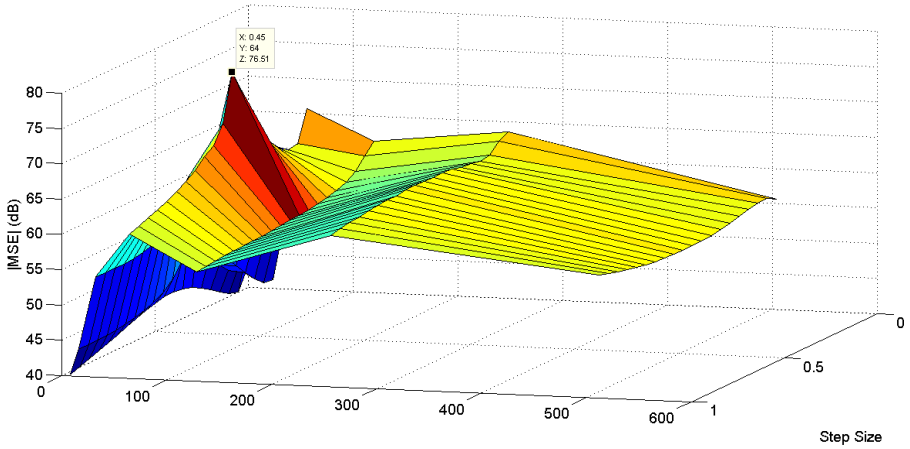

Figura 4.12 – Comparativo - NLMS: Modelo de Percurso de Eco 1.

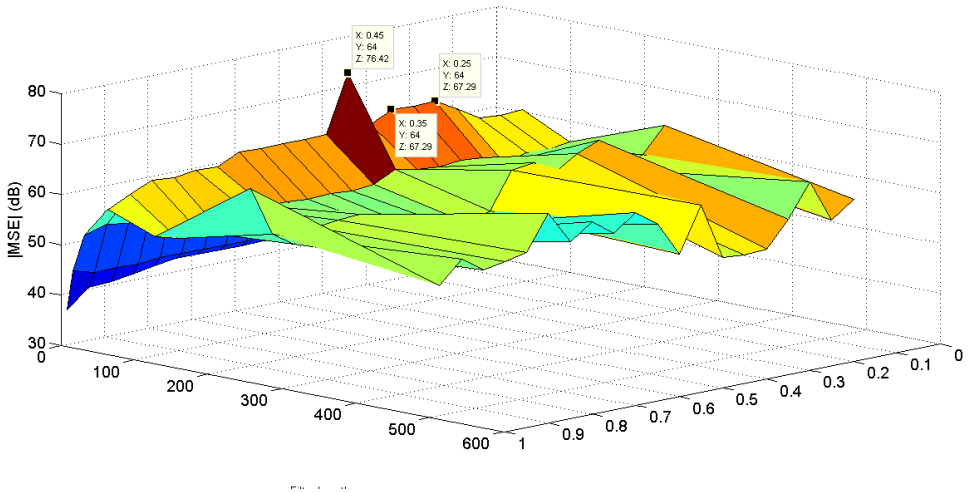

Figura 4.13 – Comparativo - 16 Bits Fixed-Point NLMS: Modelo de Percurso de Eco 1.

Neste experimento, duas características podem ser ressaltadas ao compará-las com o desempenho do NLMS normal: a estabilização da minimização do erro quadrático médio se dá em níveis inferiores e o filtro em ponto-fixo apresenta maior sensibilidade aos parâmetros. Estes efeitos já eram esperados, uma vez que se restringiu a quantidade de bits utilizada para todas as variáveis e operações matemáticas envolvidas no processo de memorização e cálculo dos coeficientes do filtro.

As figuras 4.14 e 4.15 apresentam o desempenho perceptual dos filtros NLMS e *Fixed-Point* NLMS, respectivamente.

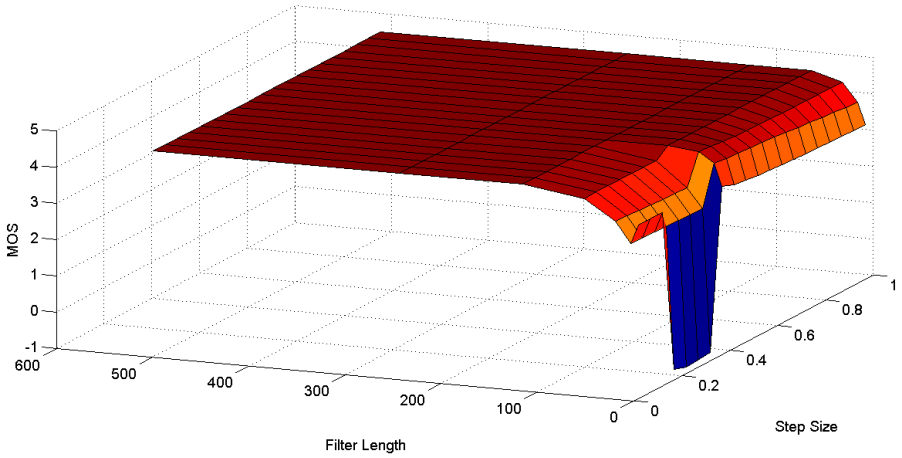

Figura 4.14 – Comparativo - MOS NLMS: Modelo de Percurso 1.

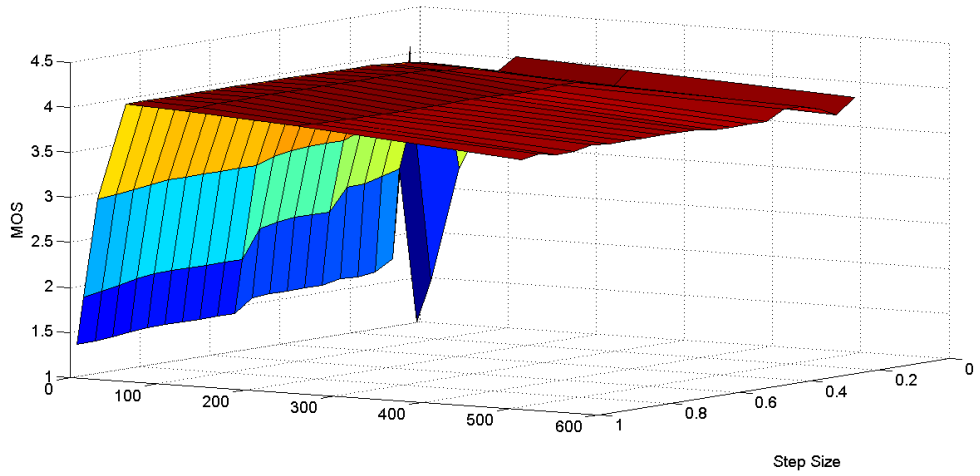

Figura 4.15 – Comparativo - MOS 16 Bits Fixed-Point NLMS: Modelo de Percurso 1.

Novamente o desempenho do filtro em ponto-fixo é inferior ao NLMS normal. Vale ressaltar que a derivada da curva de MOS é maior para o filtro *Fixed-Point* NLMS, o que indica que erros de dimensionamento do filtro, como, por exemplo, a superestimação de seu comprimento, podem afetar bruscamente a qualidade da conversação.

Mesmo apresentando desempenho inferior ao NLMS comum, o *Fixed-Point* NLMS se mostrou bastante confiável para uma implementação em uma plataforma embarcada real, uma vez que as superfícies de desempenho se mostram estáveis em níveis próximos ao NLMS comum.

4.6 – Conclusões

Este capítulo apresentou em detalhes as principais ferramentas utilizadas para o desenvolvimento de um cancelador de eco para operar em arquiteturas com restrições de precisão finita, representação finita e aritmética de ponto fixo.

Nesta etapa, o foco foi o desenvolvimento e validação do cancelador em uma plataforma virtual que simula com precisão o ambiente encontrado em processadores do tipo ARM de 16 bits.

O capítulo apresentou um estudo de resolução de bits visando esclarecer a relação entre resolução de palavra e desempenho. Com este objetivo, filtros NLMS em ponto fixo com resoluções variando de 8 a 16 bits foram testados em arquitetura de 16 bits, mostrando que o desempenho obtido é máximo quando se utiliza a resolução nativa da arquitetura.

Ao final, o capítulo apresentou uma comparação de desempenho entre as versões ponto-fixo e ponto-flutuante do filtro NLMS, comprovando o excelente desempenho comparativo do filtro em ponto-fixo com 16 bits de resolução.

Capítulo 5 - Cancelamento de Eco Elétrico em Plataforma Embarcada

5.1 – Introdução

Muitas vezes negligenciado pela literatura especializada, o processo de aplicação da teoria em sistemas reais pode se tornar um fator complicador para o desenvolvimento de aplicações comerciais. O ambiente restrito apresentado pelas plataformas de aplicação trazem inúmeros desafios ao desenvolvedor, e vão de encontro às especificações ideais muitas vezes empregadas em algoritmos teóricos.

Neste contexto, este capítulo tem como objetivo o desenvolvimento de um cancelador de eco para uma plataforma Linux embarcada em um processador do tipo ARM. Assim, o capítulo apresentará os principais aspectos práticos da implementação de um cancelador de eco em ambientes embarcados reais. Para isto, na Seção 5.2 a arquitetura ARM é apresentada ao leitor, evidenciando suas principais características. A Seção 5.3 aborda os principais componentes de uma plataforma embarcada, descrevendo seus respectivos papéis em uma aplicação real.

A Seção 5.4 apresenta as dificuldades impostas por um sistema embarcado real no desenvolvimento de algoritmos que exijam sincronismo, como, por exemplo, filtros canceladores de eco. Assim, o algoritmo denominado S-NLMS é proposto para mitigar os efeitos críticos destas restrições e seu esquema de operação é explicitado.

Na Seção 5.5 é realizado um estudo comparativo de desempenho, onde o algoritmo S-NLMS proposto neste trabalho é comparado ao NLMS desenvolvido no capítulo 4. Ambos os algoritmos são testados na plataforma embarcada real, em situações de falha de transmissão de pacotes e elevada latência de processos. Neste estudo, o desempenho de ambos é avaliado de forma quantitativa e qualitativa, visando a explicitar o impacto prático da correta aplicação da teoria em sistemas reais.

A Seção 5.6 resume o presente capítulo enfatizando suas principais contribuições.

5.2 – Arquitetura ARM (S3C6310)

O S2C6410X [17] é um microprocessador de 16/32 bits do tipo RISC desenhado para fornecer uma solução de baixo custo, baixa potencia e alto desempenho para aplicações de celulares. Para fornecer um desempenho de *hardware* otimizado para os serviços de comunicação 2.5G e 3G, o S3C6410X adota uma arquitetura de barramentos internos do tipo AMBA (*Advanced Microcontroller Bus Architecture*) de 32/64 bits composta por barramentos AXI (*Advanced Extensible Interface*), AHB (*Advanced High-performance Bus*) e APB (*Advanced Peripheral Bus*). Suas principais características estão listadas na tabela 5.1 [18]:

Tabela 5.1 - S3C6410X: Principais características dos Barramentos Internos.

Barramento	Principais Características
AMBA	1) Desenvolvimento Modular 2) Independência de Tecnologia 3) Flexibilidade 4) Compatibilidade 5) Suporte
AHB	1) Múltiplos mestres 2) Transferências Intermitentes 3) Operações com única borda de clock 4) Operações de transferência com a utilização de pipeline 5) Grande largura no barramento (de 8 a 1024 bits)
APB	1) Baixo consumo de energia 2) Pouca largura de banda 3) Baixo desempenho
AXI	1) Alta Largura de Banda 2) Baixa Latência 3) Compatível com AHB e APB

O S3C6410X possui uma interface para memórias externas otimizada, capaz de manter as altas larguras de bandas necessárias a serviços de comunicação *high-end*. O sistema de memórias possui duas portas de memória externa, DRAM e Flash/ROM. A porta DRAM pode ser configurada para suportar *mobile* DDR, DDR, *móbile* SDRAM e SDRAM. A porta Flash/ROM suporta memórias do tipo NOR-Flash, NAND-Flash, OneNAND, CF e ROM.

Para reduzir o custo e aumentar a funcionalidade do sistema, o S3C6410X inclui vários *hardwares* periféricos como interface de Câmera, controlador de LCD TFT 24 bits *true*

color, gerenciador de sistema (gerenciador de energia, etc), 4 canais de UART, 32 canais de DMA, 5 canais de *Timers* de 32 bits, 2 portas de PWM, interface I2S, interface I2C, USB host, USB OTG e controlador de SD/MMC de 3 canais.

O subsistema ARM é baseado no ARM1176JZF-S. Este inclui *caches* separados de 16 KB para instruções e dados, e inclui suporte à aceleração JAVA.

As principais características do S3C6410 são sumarizadas na Figura 5.1 abaixo:

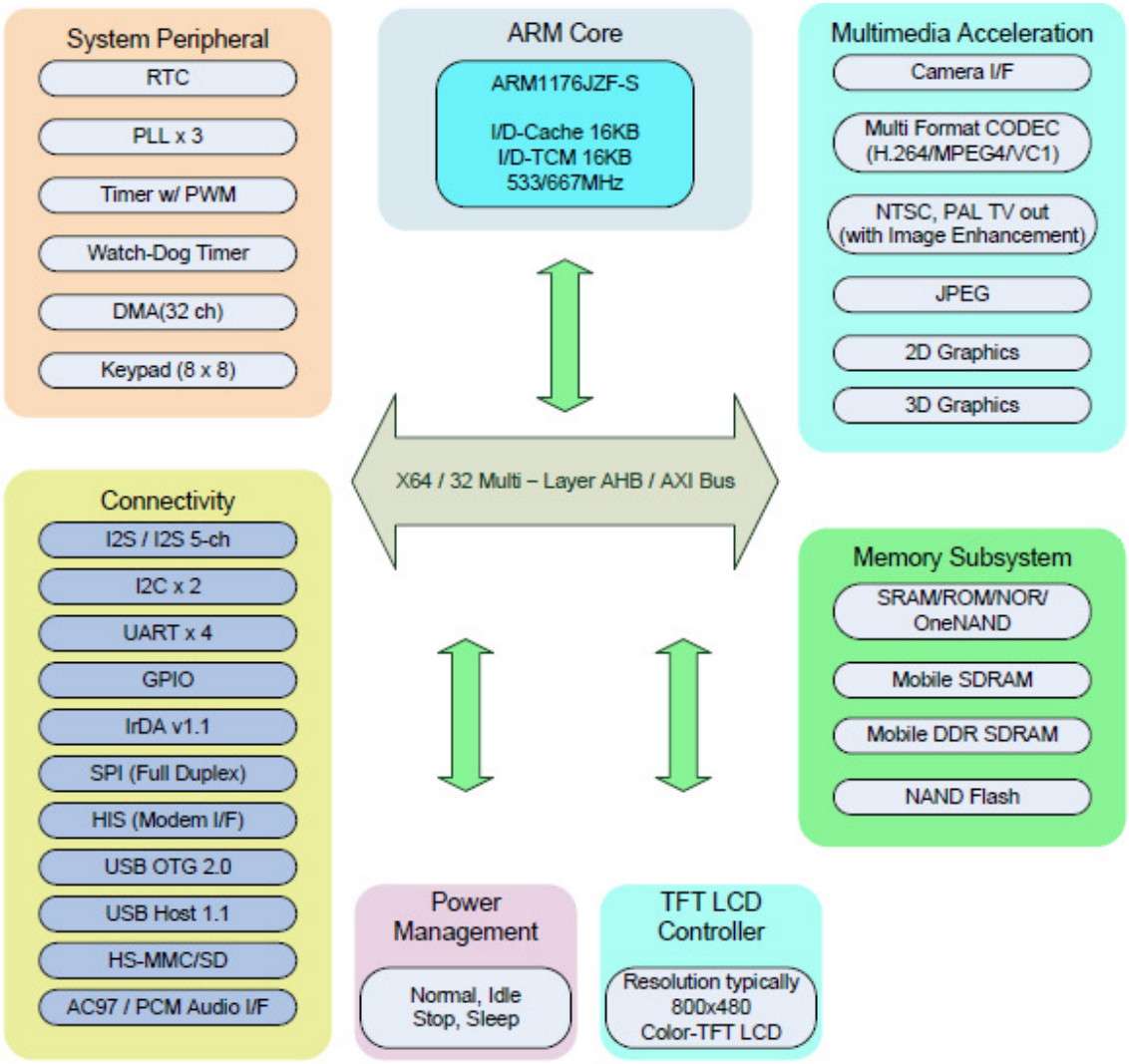

Figura 5.1 - S3C6410X: Características e Periféricos.

5.3 – Componentes Básicos da Plataforma Embarcada (*Embedded* Linux)

Esta seção descreverá de maneira breve os componentes indispensáveis para uma plataforma embarcada real, explicando os motivos pelos quais estes foram utilizados e enaltecendo seus respectivos papéis no desenvolvimento de aplicações baseadas em Linux embarcado.

Tendo em vista a natureza explanatória das sessões subsequentes potencializada pela baixa disponibilidade de bibliografia, tutoriais e afins que abordem a parte prática da criação de ambientes embarcados, um apêndice foi criado com o objetivo de detalhar todo o processo de criação de um sistema embarcado real. Assim, para uma descrição minuciosa do processo da criação de um ambiente embarcado, o leitor mais interessado pode consultar o Apêndice – Geração de um Ambiente Embarcado, ao final do texto.

5.3.1 – Toolchain

Toolchain é o nome dado ao conjunto de ferramentas utilizadas para a compilação. Como exemplo, pode-se citar o processo de compilação de um código escrito em linguagem C:

1. Pré-processador: trata todas as diretivas de pré-processamento e gera um código C intermediário.
2. Compilador: converte este código C intermediário em um código-fonte assembly.
3. Assembler: converte o código-fonte assembly em arquivo objeto.
4. Linker: converte um ou mais arquivos objeto no binário final (*firmware*, aplicação, etc).

Cada uma dessas etapas é executada por uma ferramenta, todas elas fazendo parte do *toolchain*. Existem basicamente dois tipos de *toolchain*: o *Native toolchain* e o *Cross-compiling toolchain.* O conceito é ilustrado na Figura 5.2:

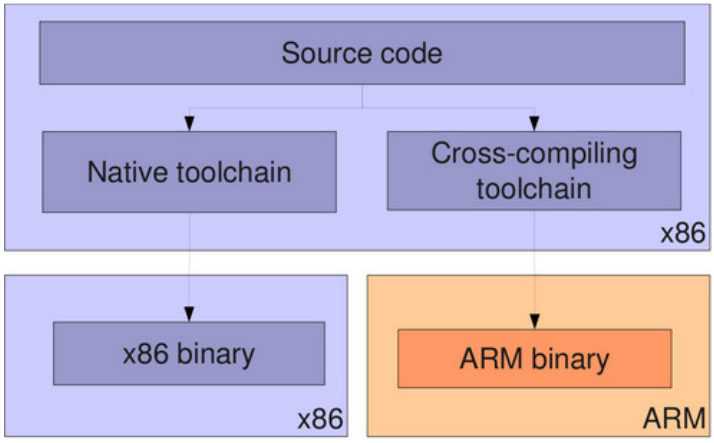

Figura 5.2 - Tipos de *toolchain* utilizados neste trabalho. O *Native toolchain,* utilizado para compilar aplicações para a arquitetura x86 e o *Cross-compiling toolchain,* utilizado para compilar aplicações para arquitetura ARM.

O bloco superior da Figura 5.2 é a máquina utilizada para desenvolvimento e o bloco de baixo representa a arquitetura alvo. Desta forma, o desenvolvedor utilizará o *native*

toolchain quando desejar compilar uma aplicação para a arquitetura da máquina utilizada para desenvolvimento, e utilizará o *cross-compiling toolchain* quando desejar gerar binários para uma arquitetura diferente da sua máquina de desenvolvimento. No exemplo da Figura 5.2, bem como neste trabalho, a máquina de desenvolvimento possui arquitetura x86 e a arquitetura alvo é ARM.

5.3.2 – *Bootloader*

O *bootloader* é uma aplicação que é carregada e executada assim que o *hardware* é ligado. No universo *desktop*, os mais utilizados são o LILO e o GRUB para carregar sistemas do tipo Unix, mas no universo *embedded* as coisas são um pouco mais complicadas. Um *bootloader* para sistemas embarcados possui três principais responsabilidades:

1. Inicializar o *hardware*.
2. Possibilitar a carga e gravação da aplicação na memória flash via alguma interface de I/O, como porta serial, USB ou interface de rede.
3. Carregar e executar aplicações na RAM.

Um *bootloader* bem conhecido no universo *embedded* é o Das U-Boot [19]. Também conhecido como apenas U-Boot, é um *bootloader* multiplataforma e *open-source*, com suporte a diversas arquiteturas como PowerPC, ARM, MIPS, Coldfire e x86.

Neste trabalho, o *hardware* utilizado possui duas memórias de programa, uma do tipo flash NOR ou outra do tipo flash NAND.

Com tempo de resposta para apagar e escrever muito grande, mas com a possibilidade de acesso à leitura em qualquer posição de memória, as memórias NOR são uma aplicação ideal para memórias do tipo ROM, para armazenar programas que dificilmente são alterados, como a BIOS de PCs e firmwares de equipamentos eletrônicos.

As memórias do tipo NAND, por sua vez, possuem um tempo de escrita bem menor e uma maior densidade, ou seja, mais MB de dados pelo mesmo espaço se comparadas com as memórias NOR. Por outro lado, não é possível o acesso de leitura a qualquer posição de memória. A leitura e gravação são feitas por blocos de memória (4KB, por exemplo). Isso dá a ela a aplicação ideal para armazenamento de dados. Quando usada como memória de programa, é necessário ter um *bootloader* que carrega seu conteúdo em memória RAM para ser executado.

Desta forma, o *hardware* deste trabalho possui um *bootloader* na memória NOR, responsável pela restauração do sistema e gravação de aplicações na memória NAND. Este *bootloader* foi utilizado para gravar o U-boot na memória NAND, de forma a ter um sistema inicializável tanto pela memória NOR quanto pela memória NAND.
Quando inicializado pela NAND, o sistema irá carregar o U-Boot, que por sua vez irá carregar e executar a aplicação. Para isto, o *bootloader* fará uso de uma região especial da CPU chamada *Boot Internal SRAM*, onde são carregados e executados os primeiros 4KB da NAND. Esses primeiros 4KB de programa serão os responsáveis por carregar o resto do programa da NAND para a memória RAM e rodar a aplicação a partir de lá.

5.3.3 - *Kernel*

O *kernel* Linux é um dos componentes do sistema operacional que requer bibliotecas e aplicações para prover funcionalidades aos usuários. Dentre suas principais características, pode-se destacar:

- Extremamente portável: possui suporte para mais de 25 arquiteturas e milhares de dispositivos de hardware.
- Modular: é capaz de rodar apenas o que é necessário para o projeto.
- Escalável: o mesmo *kernel* roda em relógios, celulares e servidores.
- Seguro: o sistema é aberto e revisado por muitos experts, não tem como esconder falhas.
- Estável: é capaz de rodar por muito tempo sem precisar de um único *reboot*.
- Copatível com padrões do mercado.
- Livre de *royalties*.
- Fácil de programar e com muitos recursos disponíveis na *internet*.

Como se pode verificar na Figura 5.3, na sua arquitetura existe uma separação bem definida entre o *kernel* (*kernel space*) e as bibliotecas e aplicações do usuário (*user space*).

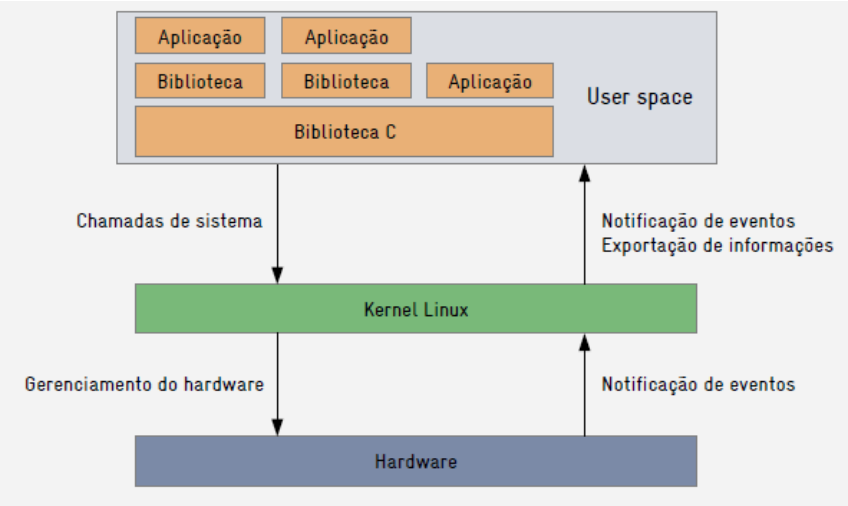

Figura 5.3 Kernel: arquitetura geral. O usuário não tem acesso direto aos controladores de *hardware*.

O *kernel* roda em modo privilegiado, com total acesso a todas as instruções da CPU, endereçamento de memória e I/O, enquanto que os processos do usuário rodam em modo restrito, com acesso limitado aos recursos da máquina. Por isso, existe uma interface de comunicação baseada em chamadas de sistema (*system calls*), para que as bibliotecas e aplicações tenham acesso aos recursos da máquina.

A versão oficial do código-fonte do *kernel* pode ser encontrada em [20], mas muitas comunidades e fabricantes de *hardware* podem manter versões alternativas do *kernel* para, por exemplo, adicionar suporte às suas plataformas de referência ou arquiteturas específicas (ARM, MIPS, PPC, etc).

5.3.4 – Qt

O *framework* Qt é um *software* livre que tem como principal vantagem a instalação e compilação em qualquer plataforma (Windows, Unix/Linux e Mac) sem alterações no código fonte. Vários exemplos de aplicações gráficas utilizando Qt podem ser facilmente encontrados no mercado como o Google Earth, Skype, VLC, Phoenix e várias aplicações para celulares.

Outra diferença do Qt em relação aos outros *frameworks* disponíveis no mercado é a notável independência entre os componentes, possibilitando um desenvolvimento estruturado dos objetos como partes isoladas (blocos) do *software*/*firmware* final. Para tornar esta propriedade possível, o Qt utiliza o conceito de *signals* e *slots*, em que os objetos podem emitir sinais para o sistema por meio dos *signals* e receber sinais por meio dos *slots*. Isto facilita a interconexão de processos em *threads* distintas, em que

funções (*slots*) na *thread* destino podem ser executadas a partir de *triggers* (*signals*) enviados pela *thread* remetente sem a utilização de troca de mensagens (*thread messaging*), semáforos ou *sockets*.

O principal motivo da escolha do Qt para a elaboração deste trabalho é que este está disponível para plataformas embarcadas tais como Windows CE, *Embedded* Linux e Android. Isto facilita o desenvolvimento estruturado de todos os aspectos do sistema, englobando os algoritmos de processamento de sinais, aquisição de dados, comunicação por rede e interface com o usuário.

A programação é realizada em C++, e um *cross*-compilador (abordado na Seção 5.3.1) é utilizado para converter o código nativo gerado em código para a plataforma escolhida.

5.4 – Implementação

Esta seção abordará o processo de criação de um filtro adaptativo para o cancelamento de eco elétrico em um sistema embarcado real, ressaltando os principais desafios e soluções encontradas ao longo de seu desenvolvimento.

Por utilizar sistemas operacionais preemptivos, vários cuidados adicionais devem ser tomados ao implementar algoritmos de processamento de sinais em plataformas embarcadas. Todo algoritmo que requeira sincronismo entre dados de entrada e saída deve passar por uma elaborada etapa de pré-processamento. Neste trabalho, esta etapa foi dividida em dois processos: o alinhamento de quadro e o alinhamento de amostras.

Conforme descrito na Seção 2.6, o sistema objetivo utiliza o padrão G711 para a transmissão de áudio. Desta forma, os pacotes são amostrados em 8kHz e entregues em quadros de 20 ms (160 bytes). Para realizar o alinhamento de quadros fornecidos ao filtro adaptativo, foram utilizados *triggers* (*signals*) emitidos dos componentes (*threads*) de recepção de rede e amostragem de linha. Estes sinais são recebidos em outros componentes (*threads*) em funções (*slots*) desenvolvidas especificamente para a tomada de decisão dentro de seus respectivos contextos de execução. O esquemático descritivo do processo de alinhamento de quadro pode ser visualizado na Figura 5.4:

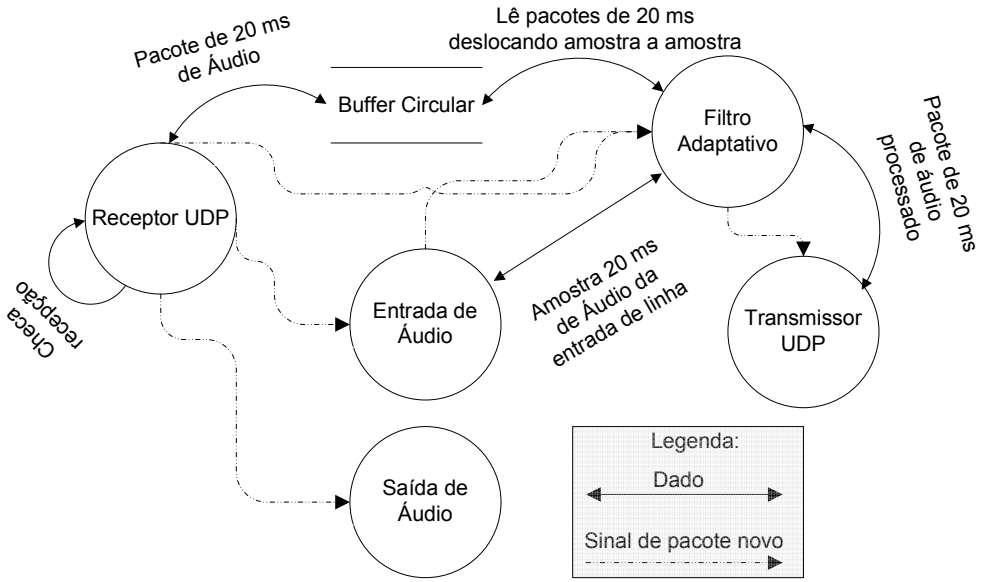

Figura 5.4 – Mapa do esquema de operação do mecanismo de alinhamento de quadro. Os componentes avisam uns aos outros os estados em que se encontram.

A *thread* de recepção UDP permanece aguardando a chegada de pacotes de rede. Para cada pacote recebido a *thread* envia um sinal (trigger) ao sistema. Este sinal passa como argumentos um vetor de bytes com o conteúdo do pacote, o endereço IP de quem enviou e a porta de recepção. Este sinal é conectado à funções (*slots*) em outros objetos por meio da palavra reservada *connect* :

connect(udpr,SIGNAL(hasNewDatagram(QByteArray,QHostAddress,quint16)),this,
SLOT(processDatagram(QByteArray,QHostAddress,quint16)));

No caso de alinhamento de quadros, o sinal de novo pacote emitido pelo receptor UDP é recebido pelos componentes de:

- Saída de áudio:
  O sinal é utilizado para mover o conteúdo útil (*payload*) do pacote para o buffer de saída de áudio.
- Entrada de áudio:
  Como o conteúdo do pacote é movido para a saída de linha, o sinal é utilizado para sincronizar a captura do retorno (eco) correspondente ao pacote recebido. Ao terminar a captura, esta *thread* emite um sinal avisando ao sistema a existência de um novo pacote amostrado de áudio.
- Filtro adaptativo:

O sinal é utilizado para a atualização dos índices internos de controle do buffer circular e move o conteúdo do pacote para o vetor de entrada do filtro. O sinal emitido pela *thread* de amostragem de linha é conectado à uma função responsável por mover o conteúdo do sinal para a vetor *desired*, utilizado para realizar as comparações entre a predição do filtro e o sinal de retorno.

Desta forma, o processamento sempre será executado no mesmo quadro de informação, uma vez que as funções são chamadas de forma sequencial, mesmo estando em *threads* (e contextos) diferentes.

A Figura 5.5 apresenta o mapa simplificado de interconexões do objeto CallManager, responsável por gerenciar todas as ligações em curso nas quatro linhas de conversação disponíveis. Neste esquemático, apenas os objetos que participam do cancelamento de eco são retratados.

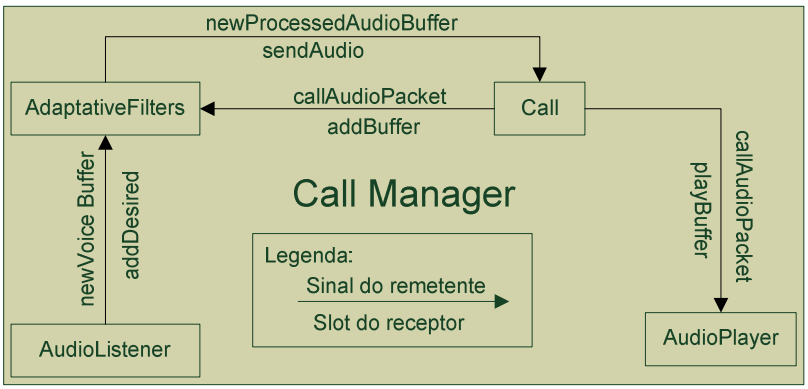

Figura 5.5 – Mapa de Interconexão: Alinhamento de Quadro. Os sinais enviados pelos componentes para comunicar seus estados internos são conectados à *slots* para o processamento e tomada de decisão.

Com o primeiro processo da etapa de alinhamento temporal resolvido, resta ainda o alinhamento de amostras dentro de um mesmo quadro. Em aplicações cujo processador é utilizado tanto para realizar a interface homem-máquina (interface gráfica, varredura de teclados, sensores de gancho, etc.) quanto para realizar o processamento de sinais (filtros, codificação/decodificação de dados, etc.), a latência dos processos pode subir vertiginosamente. Neste contexto, o alinhamento de amostras se faz indispensável para o correto processamento de dados.

Neste trabalho, o alinhamento temporal de amostras de um mesmo quadro foi realizado utilizando a correlação entre os vetores de entrada do filtro adaptativo. A correlação é aplicada ao longo de todo o vetor, buscando a posição onde seu valor é máximo. Esta

posição representa o atraso de amostras entre os dois vetores, o que possibilita o realinhamento dinâmico e correto processamento das amostras. Este processo é realizado para cada novo quadro de áudio recém-chegado no sistema.

Outra dificuldade encontrada na implementação do cancelador de eco na plataforma embarcada em questão foi a perda de pacotes de rede. Como o quadro de áudio é transferido via UDP, não há qualquer garantia de entrega, e mesmo quando entregues, não há garantia de ordenação dos quadros.

Existem várias formas de se contornar este problema. A mais simples delas, consiste na transmissão de pacotes via TCP, comunicação orientada a conexão. Desta forma se garante a chegada de todos os pacotes enviados, porém a latência entre os pacotes pode aumentar drasticamente, devido ao esquema de retransmissão de pacotes, e o problema de ordenação persiste. Outra forma seria adicionar à etapa de pré-processamento uma camada de verificação de pacotes, responsável por avisar ao sistema quando um pacote foi perdido. Uma vez que se conhece a taxa de transmissão (G711), a verificação se torna relativamente simples utilizando-se *timers* que estouram a cada Δt determinado. Se um pacote de rede não estiver no buffer quando o timer estourar, o componente avisa ao sistema a perda de pacote. Este sinal é recebido pelo componente de filtragem, que, por sua vez, deixa de atualizar os coeficientes do filtro adaptativo pelo período correspondente a um quadro de áudio, ou seja, 20 ms (160 bytes).

Esta ideia é complementar às utilizadas nos algoritmos do tipo *Set-membership* [21], onde os coeficientes do filtro apenas são atualizados se o erro associado for maior do que um patamar especificado, e do tipo *Intermittently-updated* [22], onde os coeficientes são atualizados se o intervalo de atualização, calculado a partir de sua linearidade com o logaritmo da variância do erro de estimação do filtro, for múltiplo do valor do tempo transcorrido até o momento da avaliação. No algoritmo S-NLMS (*Switched* NLMS) desenvolvido neste trabalho, a preocupação não é a otimização da complexidade do filtro, mas sim a otimização do desempenho por meio do alinhamento temporal e a proteção contra falhas de transmissão. Seu *modus-operandi* pode ser visualizado na Figura 5.6.

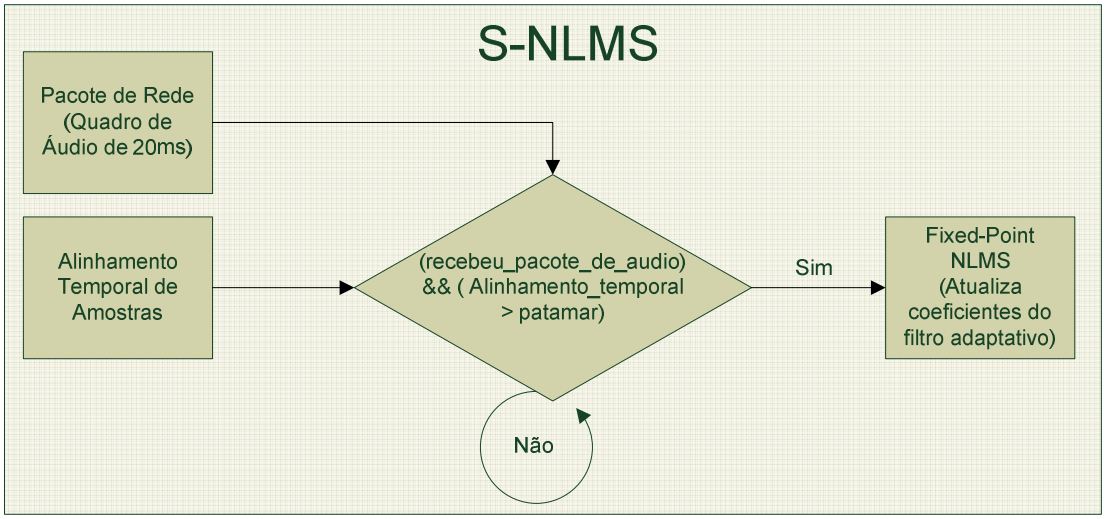

Figura 5.6 - Esquemático operacional S-NLMS. Uma camada de pré-processamento foi adicionada, visando proteger os coeficientes do filtro contra perdas de pacotes e contra desalinhamentos de amostras causados por latência entre processos.

5.5 – Resultados

Nesta seção serão apresentados os resultados obtidos com a utilização do algoritmo S-NLMS para o cancelamento de eco no sistema objetivo. Assim, as seções 5.7.1 e 5.7.2 subsequentes apresentam os efeitos dos mecanismos de proteção contra perda de pacotes e alinhamento temporal de amostras presentes no algoritmo S-NLMS.

No *setup* de teste, representado na Figura 5.7, foi utilizada a console Setha 7030IP descrita na Seção 2.6.1 para transmitir os pacotes de áudio no padrão G711 para a rede (Passo 1). Estes pacotes são recebidos pela Interface Setha 7058 (Passo 2), onde são processados e encaminhados à Interface Bidirecional Setha 7059 (Passo 3), que, por sua vez, encaminha ao barramento de áudio (Passo 4).

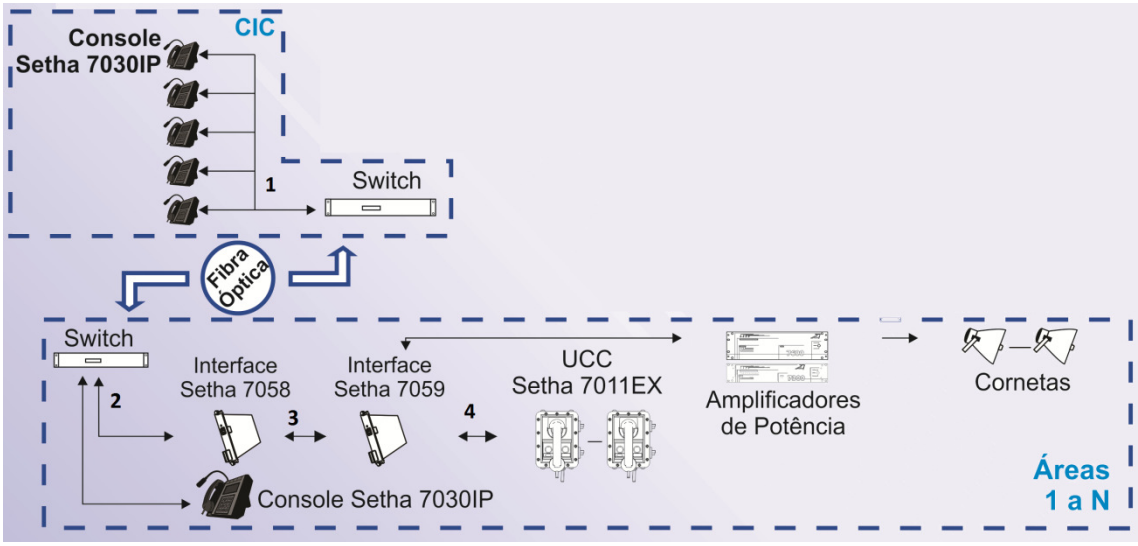

Figura 5.7 - Fluxo de áudio na plataforma de testes. Os passos estão numerados em ordem cronológica de forma crescente .

Para efeito comparativo, o sinal de referência 1 descrito na Seção 3.3, e reproduzido na Figura 5.8 por conveniência, foi utilizado em todos os procedimentos realizados, ou seja, um sinal de voz feminina anecoica com duração de aproximadamente 9 segundos.

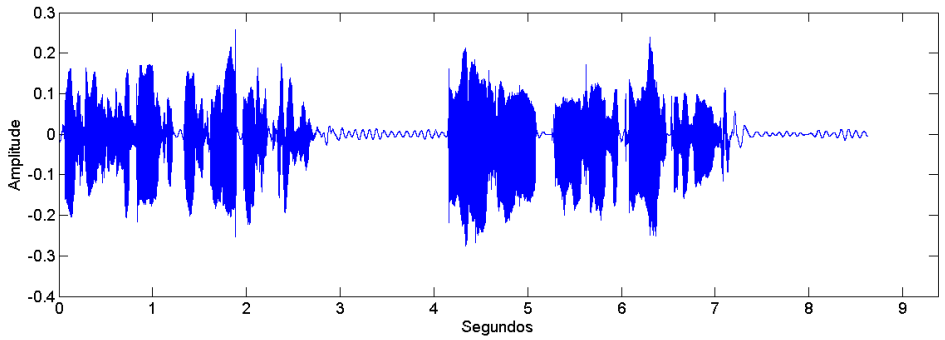

Figura 5.8 - Sinal de referência 1 utilizado nos testes, contendo sinal sonoro e surdo (silêncio).

Os resultados obtidos são comparados aos obtidos com a utilização do algoritmo NLMS padrão no cancelamento de eco na mesma plataforma e sob as mesmas condições. Em ambos os algoritmos foram utilizados filtros de comprimento igual a 128 com os parâmetros ótimos determinados na Seção 3.6, ou seja, os algoritmos foram avaliados com passo de adaptação e variável de regularização iguais a 0,3 e 0,001, respectivamente.

5.5.1 – Proteção contra perda de pacotes

O sistema objetivo foi concebido para operar nas mais diversas topologias de rede, onde, muitas vezes, o tráfego de voz pode ser transmitido por redes WAN (*wide area network*) sem qualquer tipo de política de QOS (*quality of service*) ou priorização de pacotes. Neste contexto, perdas de pacotes devem ser esperadas.

Esta seção focará na avaliação da robustez contra perda de pacotes de rede (quadros de áudio), assim, o desempenho dos algoritmos foi testado em ambientes reais com perdas percentuais de pacotes de 0,23, 0,46, 0,70, 0,93 e 1,15.

O desempenho em ambientes com perdas percentuais de pacotes inferiores a 0,50 pode ser visualizado nas figuras 5.9 e 5.10 abaixo:

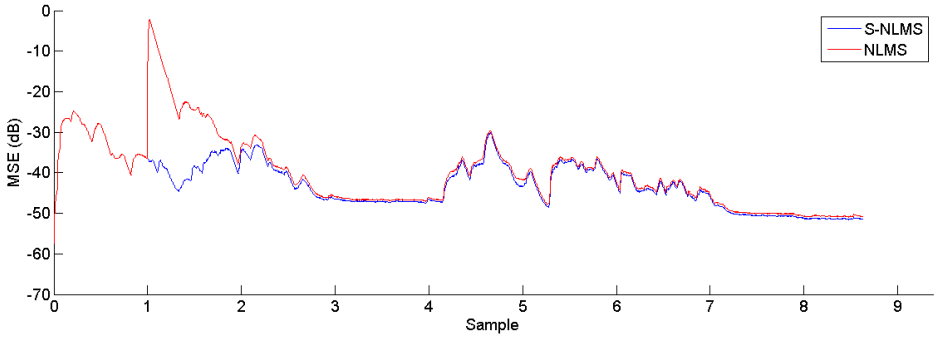

Figura 5.9 - Comparativo proteção contra perda de pacotes: 0,23% de perda.

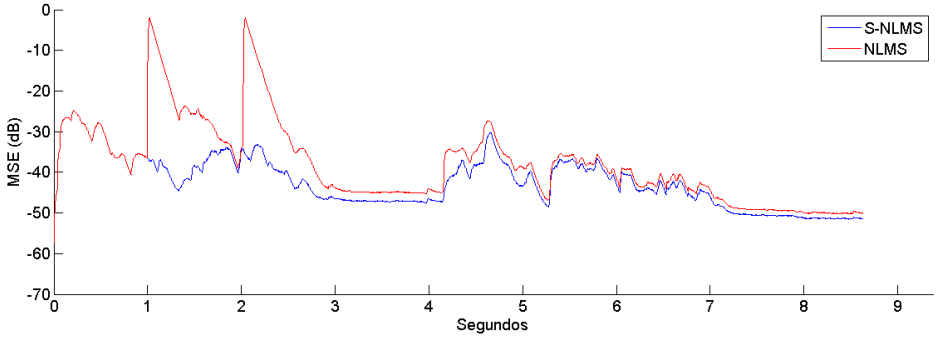

Figura 5.10 - Comparativo proteção contra perda de pacotes: 0,46% de perda.

Como se pode ver, neste contexto, o desempenho do algoritmo NLMS é bastante semelhante ao S-NLMS, apresentando um resultado em minimização do erro quadrático em estado estacionário de apenas 1 a 3 dB inferior.

No caso de ambientes com perdas percentuais superiores a 0,50 (figuras 5.11, 5.12 e 5.13), a comparação entre os algoritmos traz resultados bem mais expressivos:

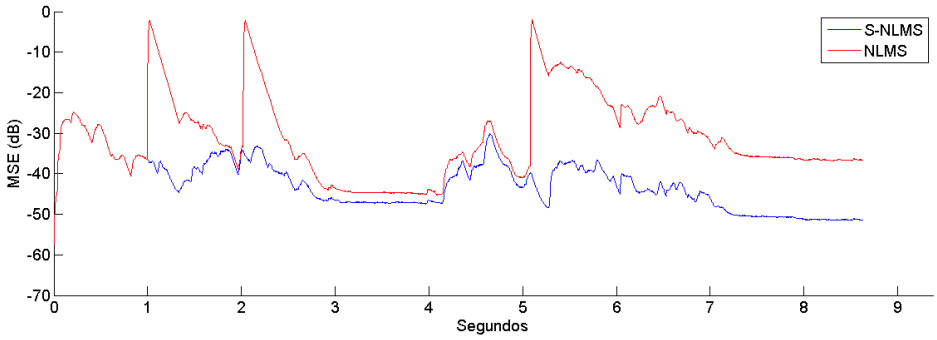

Figura 5.11 - Comparativo proteção contra perda de pacotes: 0,70% de perda.

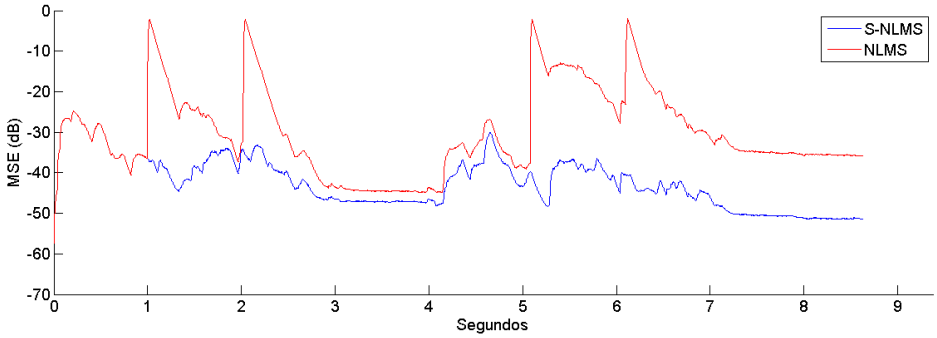

Figura 5.12 - Comparativo proteção contra perda de pacotes: 0,93% de perda.

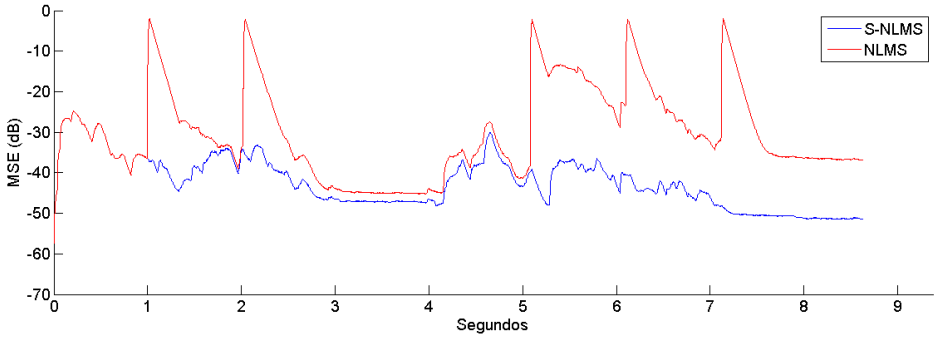

Figura 5.13 - Comparativo proteção contra perda de pacotes: 1,15% de perda.

Como se pode verificar, o algoritmo S-NLMS apresenta um resultado em minimização do erro médio quadrático em estado estacionário em média 20 dB superior.

O leitor mais atento verificou que em todos os ambientes anteriormente simulados as perdas de pacotes se deram em partes cujo sinal de referência apresenta elevada energia, ou seja, os pacotes perdidos continham sinal sonoro. Da mesma forma, os pacotes subsequentes recebidos pelo algoritmo apresentam energia alta, o que favorece a readaptação do filtro.

Caso o pacote perdido - e os recebidos em seguida - contenham sinal surdo, o comportamento é bem diferente. A Figura 5.14 apresenta um comparativo de desempenho dos filtros S-NLMS e NLMS em face da perda de apenas um pacote de áudio em diferentes momentos do sinal de referência 1 (sinal surdo e sinal sonoro).

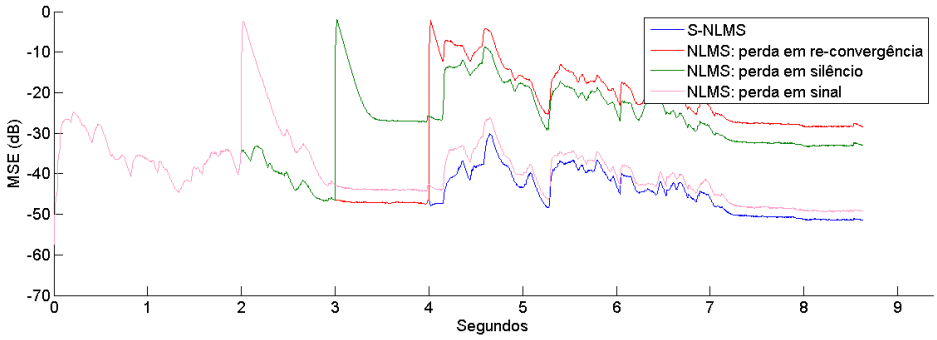

Figura 5.14 – Comparação 0,23% de perda com sinal e em re-convergencia.

Conforme esperado, o algoritmo S-NLMS cumpre seu propósito e apresenta um desempenho superior em todas as situações. Independentemente do momento em que a falha de recepção ocorre, o algoritmo responde aproximadamente da mesma forma, apresentando um desempenho médio em minimização do MSE entre -40 e -50 dB.

O algoritmo NLMS, por sua vez, sofre diferentes impactos de acordo com o conteúdo e momento do pacote perdido. Quando o pacote perdido possui alta energia (curva rosa da Figura 5.14), o filtro consegue se adaptar e apresenta desempenho próximo ao S-NLMS. Nos casos em que a perda ocorre no período de sinal de silêncio (curvas verde e vermelha da Figura 5.14), o desempenho do NLMS é aproximadamente 23 dB inferior ao apresentado pelo S-NLMS. O interessante é que o desempenho do NLMS quando a perda de pacote se dá imediatamente antes de receber sinal de energia elevada (curva vermelha da Figura 5.14) é aproximadamente 6 dB inferior ao seu desempenho quando o pacote perdido –e os recebidas na sequência- são de baixa energia (curva verde da Figura 5.14). Isto pode ser explicado pela perda de pacote no momento mais crítico do filtro: a re-convergência. Esta característica mostra um aspecto bastante frágil da aplicação direta da teoria em sistemas reais, pois, neste experimento, o desempenho de um filtro desenvolvido sem os devidos cuidados se mostrou aproximadamente 95% inferior ao proposto por este trabalho.

A natureza crítica da re-convergência e a superioridade do algoritmo S-NLMS também são verificadas no estudo do efeito acumulativo de perdas de pacotes apresentado na Figura 5.15.

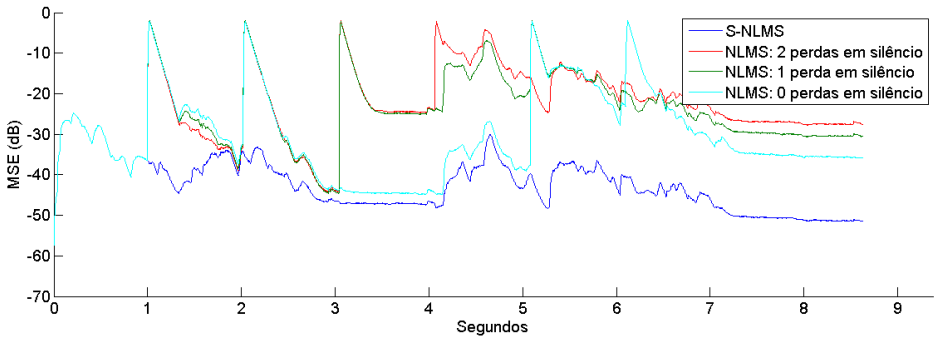

Figura 5.15 - Comparação efeito acumulativo perda com sinal e re-convergencia.

Mesmo perdendo pacotes no quinto e sexto segundos, o desempenho em estado estacionário do NLMS quando não perde pacotes de sinal de silêncio (curva ciano da Figura 5.15) é superior aos demais. Novamente, o pior desempenho é apresentado pelo filtro que perde pacote no momento da re-convergência (curva vermelha da Figura 5.15), ficando cerca de 3 dB abaixo do que perde pacote no início do sinal de silêncio (curva verde da Figura 5.15).

Para se entender o efeito prático da diferença entre os desempenhos dos algoritmos, uma avaliação perceptual foi conduzida. Os resultados apenas confirmam aqueles obtidos na análise quantitativa e podem ser visualizados na Figura 5.16.

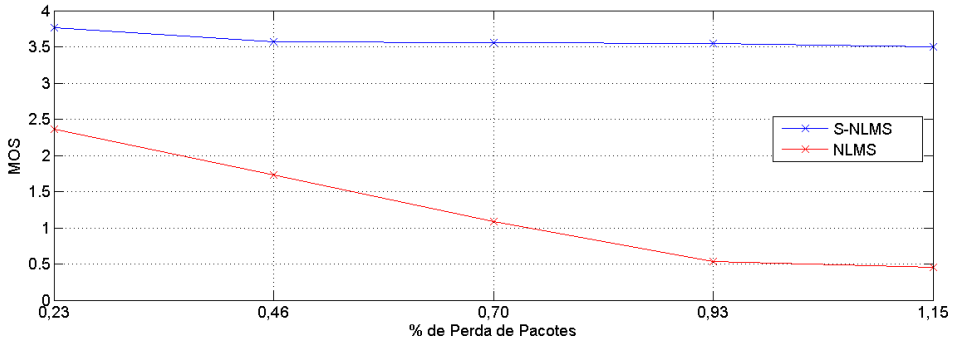

Figura 5.16 - MOS: Comparativo proteção contra perda de pacotes.

Ao observar os comparativos quantitativos e qualitativos em cada ambiente de operação, pode-se inferir a superioridade do algoritmo S-NLMS, principalmente quando a perda de pacote se dá em situação de re-convergência (sinal de silêncio). Em redes altamente congestionadas, a utilização de filtros protegidos se faz indispensável para se

obter uma conversação de qualidade. A avaliação perceptual mostra um aumento de qualidade de aproximadamente 53% a 700% em ambientes com perdas percentuais de pacotes de 0,23 a 1,15.

5.5.2 – Alinhamento Temporal

Nesta seção será avaliada a influência do alinhamento temporal das amostras de um mesmo quadro de áudio no desempenho dos algoritmos NLMS e S-NMLS em ambientes livres de perda de pacotes.

Para o algoritmo S-NLMS, as amostras são alinhadas por meio da correlação entre os vetores de áudio chegados pela rede (sinal) e amostrados do barramento (eco). A Figura 5.17 mostra o resultado:

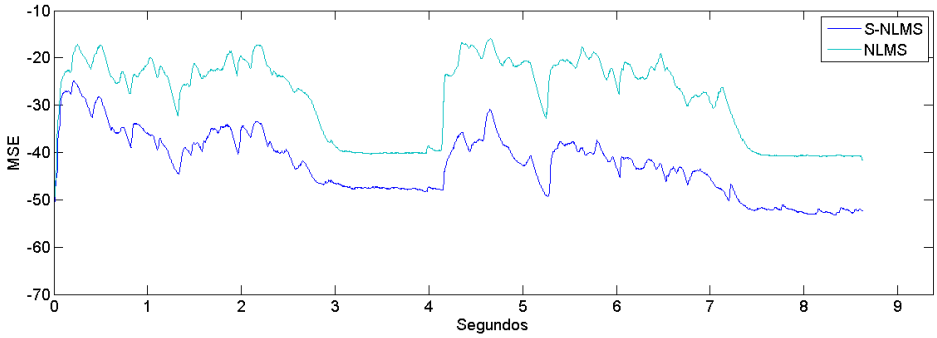

Figura 5.17 - Comparativo alinhamento temporal.

Novamente o desempenho do algoritmo S-NLMS se mostra superior, apresentando uma atenuação média do MSE 10 dB melhor do NLMS. A avaliação perceptual uma vez mais confirma o resultado quantitativo e pode ser visualizada na Figura 5.18.

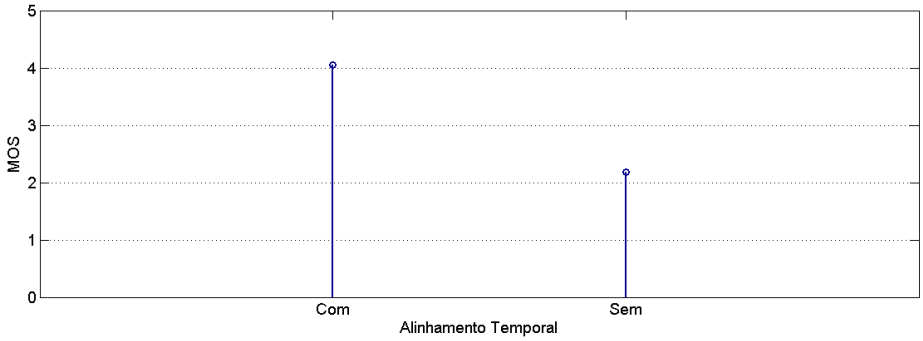

Figura 5.18 - MOS: Comparativo alinhamento temporal.

O desempenho qualitativo do algoritmo S-NLMS é aproximadamente 80% superior ao apresentado pelo NLMS. Esta diferença representa substancial ganho de qualidade na

conversação e pode determinar a viabilidade prática de um sistema de comunicação de alta confiabilidade.

Vale ressaltar que neste experimento desalinhamentos de até aproximadamente 10 milissegundos puderam ser detectados e corrigidos, mostrando a natureza crítica dos sistemas preemptivos embarcados. Este desalinhamento pode ser drasticamente reduzido com a utilização de processadores mais poderosos ou sistema operacionais baseados em núcleos de tempo-real (*real-time kernel*).

5.6 – Conclusões

No segmento inicial, este capítulo abordou as principais ferramentas para o desenvolvimento de aplicações em plataformas embarcadas, apontando ao leitor a importância de cada componente de um ambiente de desenvolvimento de aplicações para um sistema real.

Mostrou que a aplicação direta da teoria em sistemas reais muitas vezes não é adequada, principalmente em sistemas cujo processador é responsável tanto pela interface homem-máquina quanto pelo processamento dos sinais envolvidos. Em plataformas embarcadas, vários problemas devem ser solucionados antes mesmo de se iniciar o processamento dos dados, como, por exemplo, o tratamento de perdas de pacotes.

Ressaltou a natureza frágil dos sistemas sincronizados, mostrando o impacto causado pelo desalinhamento das amostras no desempenho do filtro adaptativo. Para solucionar este problema, propôs uma etapa de pré-processamento composta por algoritmos especializados no alinhamento temporal dinâmico das amostras de áudio.

Assim, o algoritmo S-NLMS (*Switched* NLMS) foi apresentado e seu *modus-operandi* foi esclarecido. Sua superioridade em relação ao algoritmo NLMS foi comprovada por meio da comparação do desempenho na plataforma real, onde ambos os algoritmos NLMS desenvolvido no Capítulo 4 e S-NLMS desenvolvido neste capítulo foram empregados com os parâmetros ótimos encontrados no Capítulo 3, ou seja, passo de adaptação (μ) igual a 0,3 e variável de regularização (γ) de 0,001.

Nas comparações realizadas, o impacto de cada problema foi avaliado de forma exclusiva. Assim, pôde-se obter uma comparação justa entre o S-NLMS e o NLMS em ambientes com perdas de pacotes e desalinhamento temporal das amostras. Ficou demonstrada a relevância da proteção contra falhas de transmissão empregada pelo

algoritmo S-NLMS, em que a perda de apenas um pacote contendo 20 ms de áudio no momento crítico da re-convergência do filtro NLMS pode arruinar o desempenho geral do algoritmo. O efeito acumulativo das perdas também foi avaliado, onde uma vez mais o algoritmo S-NLMS se mostrou superior.

Na comparação realizada para avaliar o efeito do alinhamento temporal das amostras utilizada pelo algoritmo S-NLMS, também ficou evidenciado o desempenho superior do algoritmo proposto por este trabalho. De forma que o desempenho do filtro operando em amostras alinhadas (S-NLMS) foi cerca de 10 dB superior ao NLMS operando em amostras desalinhadas.

Ao final, o efeito prático da etapa de pré-processamento proposta pelo algoritmo S-NLMS desenvolvido neste trabalho foi demonstrado por meio de avaliações perceptuais conduzidas nos filtros S-NLMS e NLMS. Nestas avaliações, o filtro S-NLMS apresentou um desempenho de 53% a 700% superior na conversação com perdas de pacotes e um desempenho aproximadamente 80% superior na conversação com alinhamento dinâmico das amostras.

Capítulo 6 - Conclusões

Esta dissertação abordou o problema do eco em sistemas de comunicação. O Capitulo 2 classificou os tipos de eco existentes (acústico e elétrico), explicando suas principais características. O capítulo apresentou as principais soluções numéricas encontradas na literatura especializada atual, descrevendo matematicamente todos os passos de cada estratégia citada. O sistema objetivo também foi descrito em detalhes, visando familiarizar o leitor às suas principais características de operação e construção. Ao final, as dificuldades de implementação dos algoritmos teóricos em sistemas reais foram discutidas, de forma a alertar o leitor os desafios a serem vencidos em uma aplicação profissional.

No Capítulo 3 foi apresentado o cancelamento livre de eco, onde os algoritmos citados no capítulo anterior foram simulados e comparados. Para tanto, foram descritos os modelos de percurso de eco utilizados, bem como a métrica e metodologia de comparação de desempenho. O capítulo apresentou uma análise comparativa inovadora, onde dados quantitativos foram cruzados com dados qualitativos visando obter os parâmetros ótimos de cada estratégia de adaptação. O capítulo se encerra com uma comparação direta entre todos os algoritmos testados, onde se decide pela implementação em ponto-fixo do algoritmo NLMS, devido à sua baixa complexidade aliada ao elevado desempenho e estabilidade.

O Capítulo 4 abordou o cancelamento com restrições de eco elétrico, onde as restrições impostas por uma plataforma real foram explicitadas e simuladas. Desta forma, o capítulo apresentou todo o ferramental necessário para o correto desenvolvimento e simulação de algoritmos em ponto-fixo, descrevendo a partir de exemplos o passo a passo da conversão de algoritmos em ponto-flutuante para algoritmos em ponto-fixo. Este capítulo apresentou um estudo do impacto da quantidade de bits utilizada para representar os números, comparando o resultado do cancelador de eco para resoluções de 8 a 16 bits. A partir deste estudo se concluiu que o desempenho máximo é obtido quando a resolução nativa da arquitetura é utiliza, e a relação custo x beneficio pende para a utilização da resolução máxima permitida. Ao final, o excelente desempenho do algoritmo NLMS em ponto-fixo foi comprovado a partir de comparações qualitativas e

quantitativas ao desempenho do algoritmo NLMS em ponto-flutuante desenvolvido no capítulo anterior, o que o credenciou para uma implementação em uma plataforma embarcada real.

O Capítulo 5 abordou em detalhes o cancelamento de eco em um sistema real. Para isto, apresentou as características do processador utilizado, bem como a as ferramentas necessárias para o desenvolvimento em um computador pessoal. Temas como a *cross*-compilação do sistema operacional Linux, bem como do *framework* Qt foram abordados.

Ainda no Capítulo 5, foi apresentado o algoritmo S-NLMS desenvolvido neste trabalho, uma versão protegida do tradicional NLMS. O algoritmo desenvolvido leva em conta o problema de alinhamento temporal e a perda de pacotes para otimizar o desempenho do cancelador de eco. Desta forma, também apresentou a natureza crítica da aplicação direta da teoria em sistemas reais, onde problemas de cunho prático podem deteriorar o desempenho dos filtros. Este fato foi demonstrado pelo efeito devastador da perda de apenas um pacote de áudio no momento de re-convergência do filtro NLMS, que produziu resultados aproximadamente 50% inferiores ao demonstrado pelo S-NLMS.

Ambientes sem perdas de pacotes também foram estudados no Capítulo 5, e os efeitos positivos do mecanismo de alinhamento dinâmico das amostras presente no algoritmo S-NLMS foi demonstrado. Utilizando o algoritmo S-NLMS na plataforma embarcada real, foram identificados e corrigidos desalinhamentos de até aproximadamente 10 ms, o que demonstra a natureza crítica deste tipo de sistema. Assim, devido à sua camada de pré-processamento adicional, o filtro S-NLMS garantiu um desempenho aproximadamente 80% superior ao NLMS.

As contribuições desta tese mostram que apesar desta área de pesquisa estar solidificada, a abordagem prática é bastante negligenciada pela literatura especializada. Ainda existem diversos temas que podem ser aprofundados, assim, as seguintes propostas de trabalhos futuros são apresentadas:

- Utilização de estratégias de adaptação com velocidade de convergência maior: implementações futuras devem levar em conta algoritmos como o NDRLMS e ou BNDRLMS, uma vez que apresentaram desempenhos muito superiores aos demais, com o preço de uma complexidade computacional superior.

- Utilização de filtragem *set-membership*: visando diminuir a complexidade dos algoritmos, e possibilitando sua utilização em sistemas embarcados reais, a filtragem *set-membership* deve ser considerada. Resultados anteriores [6] [21]mostram que a complexidade de algoritmos baseados em reutilização de dados pode ser reduzida em aproximadamente 25% a 30%.
- Utilizar *kernels* de tempo real como o RTLinux, o RTAI e o Xenomai: visando aumentar o controle sobre o sincronismo dos sinais por meio do controle direto das chamadas de sistema, *kernels* de tempo real podem ser utilizados em conjunto com o Linux. Desta forma, não só o controle de sincronismo é melhorado, como também a latência entre os processos é minimizada.
- Aplicar o conjunto de patches (PREEMPT_RT) ao kernel do Linux: o processo de adequação de outros *kernels* para utilização com o Linux pode ser bastante dispendioso, neste caso, existe um conjunto de patches que podem ser aplicados ao *kernel* nativo do LInux para melhorar o cenário de elevada latência de processos, e, consequentemente, facilitar o sincronismo entre eles.

Apêndice – Geração de um Ambiente Embarcado

Este apêndice abordará toda a parte prática do desenvolvimento de um ambiente embarcado profissional que pode ser utilizado para rodar os mais variados tipos de aplicações.

Toolchain

Gerando o *toolchain*

O processo de geração de *toolchain* é complexo e tedioso, mas existem diversas ferramentas que automatizam o procedimento. Dentre elas, a mais famosa é a crosstool-ng [23], que será utilizada para gerar um *toolchain* para *cross*-compilar aplicações Linux para ARM.

O crosstool-ng pode ser baixado da página do projeto em [23]. Inicialmente deve ser criada uma pasta de trabalho na máquina (no exemplo, /opt/toolchain), em seguida deve-se baixar e descompactar o arquivo:

```
$ cd /opt/toolchain
$ wget http://crosstool-ng.org/download/crosstool-ng/crosstool-ng-1.13.1.tar.bz2
$ tar jxfv crosstool-ng-1.13.1.tar.bz2
$ cd crosstool-ng-1.13.1/
```

Em seguida deve-se configurar e compilar o crosstool-ng:

```
$ ./configure --local
$ make && make install
```

As instruções acima gerarão o script “ct-ng”, que será utilizado para configurar o *toolchain*.

O crosstool-ng vem com algumas configurações de *toolchain* por padrão, que podem ser visualizadas com o comando abaixo:

$ ./ct-ng list-samples

Sample name	Status
alphaev56-unknown-linux-gnu	[L X]
alphaev67-unknown-linux-gnu	[L X]
arm-bare_newlib_cortex_m3_nommu-eabi	[L X]
arm-cortex_a15-linux-gnueabi	[L X]
arm-cortex_a8-linux-gnueabi	[L]
arm-davinci-linux-gnueabi	[L]
armeb-unknown-eabi	[L]
armeb-unknown-linux-gnueabi	[L X]
armeb-unknown-linux-uclibcgnueabi	[L X]
arm-iphone-linux-gnueabi	[L X]

...

A listagem acima está exibindo apenas uma parte das configurações disponíveis por padrão no crosstool-ng. Ele é capaz de gerar *toolchains* para ARM, MIPS, PPC, x86 e AVR, dentre outras arquiteturas e plataformas.

Apesar de ser possível criar uma configuração desde o início, o mais comum é usar uma das configurações pré-definidas, e então trabalhar em cima dela. Neste exemplo, será utilizada a "arm-unknown-linux-uclibcgnueabi" e pequenas alterações serão realizadas. Para isto, a configuração deve ser carregada com o comando:

$ ./ct-ng arm-unknown-linux-uclibcgnueabi

Agora o menu de configuração do crosstool-ng pode ser aberto utilizando o seguinte comando:

$ ./ct-ng menuconfig

Na opção "*Paths and misc options*" deve-se configurar a quantidade de *threads* de execução. Essa configuração ajuda a diminuir o tempo de compilação. O número de núcleos do processador da máquina deve ser multiplicado por dois:

(2) *Number of parallel jobs*

Ainda neste menu deve-se configurar o diretório de instalação do *toolchain* na opção "*Prefix directory*":

(/opt/toolchain/x-tools/${CT_TARGET}) Prefix directory

Voltando à tela principal, deve-se entrar na opção "*Toolchain options*" para configurar o *alias:*

(arm-linux) Tuple's alias
Configurando o *alias* com arm-linux, todas as ferramentas começarão com o prefixo "arm-linux". Por exemplo, o gcc do *toolchain* terá o nome "arm-linux-gcc".

Após salvar as configurações e sair, a geração do *toolchain* será iniciada com o comando:

$ ./ct-ng build
O processo de compilação pode demorar um pouco. No final, o *toolchain* gerado pode ser verificado no diretório abaixo:

$ ls /opt/toolchain/x-tools/arm-unknown-linux-uclibcgnueabi/bin

Utilizando o *toolchain*
Sempre que o *toolchain* for ser utilizado, a variável ambiente PATH deverá ser configurada:

$ export PATH=/opt/toolchain/x-tools/arm-unknown-linux-uclibcgnueabi/bin:$PATH
Para compilar nativamente uma aplicação qualquer o seguinte comando deve ser utilizado:

$ gcc main.c -o main
Para confirmar que a aplicação foi compilada nativamente, deve-se utilizar o comando file:

$ file main
main: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.15, not stripped
Para *cross*-compilar, basta trocar o gcc pelo compilador do *toolchain*:

$ arm-linux-gcc main.c -o main
Para confirmar a compilação cruzada, o comando file deve ser utilizado novamente:

$ file main
main: ELF 32-bit LSB executable, ARM, version 1 (SYSV), dynamically linked (uses shared libs), not stripped

Configuração

O *kernel* possui centenas de *drivers* de dispositivos, diversos protocolos de rede e muitos outros itens de configuração. O processo de configuração serve para customizar o *kernel* para ser compilado para a CPU/Plataforma de aplicação e, portanto, o conjunto de opções a serem habilitadas depende do *hardware* disponível e funcionalidades desejadas.

As configurações são salvas em um arquivo chamado .config no diretório principal dos fontes do *kernel*, e possuem o formato *key=value*, como, por exemplo, *CONFIG_ARM=y*. De qualquer forma, dificilmente o usuário editará o arquivo .config manualmente, uma vez que existem diversas ferramentas de interface gráfica para configurar o *kernel* e gerar o arquivo de configuração automaticamente (make menuconfig, make gconfig, make xconfig, make nconfig).

Toda a configuração do *kernel* é dependente da arquitetura. Por padrão, o *kernel* considera um *build* nativo, então ao executar o comando *make menuconfig*, irá utilizar a arquitetura da máquina de desenvolvimento (normalmente x86). Portanto, para configurar o *kernel* para a arquitetura ARM o usuário precisa especificar o tipo de arquitetura no comando:

```
$ make ARCH=arm menuconfig
```

O uso de arquivos pré-configurados é a forma padrão de configurar um *kernel* para uma plataforma específica. Os arquivos de configuração pré-definidos para diversas plataformas estão disponíveis em arch/<arch>/configs/. Para compilar o *kernel* utilizando um arquivo de configuração específico, basta utilizar a sintaxe:

```
$ make ARCH=arm arquivo_de_configuração
```

Compilação

Depois de configurado, pode-se proceder à *cross*-compilação informando a arquitetura e o prefixo do *cross-compiler*:

```
$ make ARCH=arm CROSS_COMPILE=arm-linux-
```

O comando acima irá gerar uma imagem genérica para ARM. Caso seja necessário gerar uma imagem específica para um determinado *bootloader* (U-Boot neste trabalho) deve-se adicionar o nome da imagem ao final do comando :

```
$ make ARCH=arm CROSS_COMPILE=arm-linux- uImage
```

Ao final do processo de compilação, são geradas as seguintes imagens:

- vmlinux: imagem do *kernel* no formato ELF, que não é inicializável, mas pode ser utilizada para debug.
- *.ko: módulos do *kernel*, dentro de seus respectivos diretórios.
- Em arch/<arch>/boot/:
    - Image: imagem final do *kernel*, incializável e descomprimida.
    - *Image: imagem inicializável e comprimida do *kernel* (bzImage para x86, zImage para ARM, etc).
    - uImage: imagem do *kernel* para o U-Boot.

Caso a compilação tenha sido realizada para a plataforma nativa, o *kernel* pode ser instalado utilizando o comando:

```
$ make install
```

No caso de plataformas embarcadas, este comando nunca é utilizado. Normalmente a imagem gerada (uImage, zImage, etc) é gravada em algum tio de dispositivo de armazenamento (cartão SD, memória flash, etc) e é inicializada pelo *bootloader*.

Qt

Embedded Qt

Para rodar aplicações Qt em plataformas embarcadas, é necessário que as bibliotecas Qt sejam *cross*-compiladas. Para isto, o primeiro passo é baixá-las e extrair o código-fonte:

```
$ wget http://get.qt.nokia.com/qt/source/qt-everywhere-opensource-src-4.8.1.tar.gz
$ tar zxfv qt-everywhere-opensource-src-4.8.1.tar.gz
$ cd qt-everywhere-opensource-src-4.8.1
```

Conforme explicitado na Seção 5.3.4, antes de utilizar o *toolchain* deve-se configurar a variável PATH:

$ export PATH=/opt/toolchain/x-tools/arm-unknown-linux-uclibcgnueabi/bin:$PATH
Agora deve-se configurar o Qt:

$ ./configure -opensource -confirm-license -embedded arm -xplatform qws/linux-arm-g++ -little-endian -qt-zlib -qt-libtiff -qt-libpng -qt-libmng -qt-libjpeg -prefix /usr/local/qt-arm

Esta configuração irá preparar o Qt para ser compilado para um sistema ARM *little-endian*, cujo *cross*-compiler tem o prefixo arm-linux-.

Agora basta compilar e instalar:

$ make
$ sudo make install

Na plataforma de destino, as bibliotecas geradas devem ser copiadas para a pasta de destino especificada no parâmetro –prefix da configuração, ou seja, neste exemplo devem ser copiadas para a pasta /usr/local/qt-arm. Como esta pasta não é o caminho padrão do Linux, seu caminho deve ser incluído na variável de ambiente LD_LIBRARY_PATH:

export LD_LIBRARY_PATH=/usr/local/qt-arm/lib:$LD_LIBRARY_PATH

Configurando o Qt Creator

Para criar aplicações na máquina de desenvolvimento e cross-compilá-las para a plataforma destino, basta configurar o Qt Creator para utilizar o qmake *cross*-compilado e o *toolchain* criado:

1. Abra o Qt Creator e acesse o menu "*Tools -> Options -> Qt4*"
2. Clique em "*Add"*
3. Configure o campo *"Version"* com *"ARM"*
4. Configure o campo *"qmake location"* com o local de instalação do qmake (/usr/local/qt-arm/bin/qmake)
5. Clique em *OK.*
6. Acesse o menu *"Tools -> Options -> ToolChains"*
7. Clique em *"Add"*
8. Configure o caminho completo do compilador C++ (opt/toolchain/x-tools/arm-unknown-linux-uclibcgnueabi/bin/arm-linux-g++)
9. Clique em *OK*

Bibliografia

[1] S. Vaseghi, *Lecture Notes Chapter 14 - Echo Cancellation,* Brunel University: Department of Electronics and Computer Engineering.

[2] Telchemy, *Application Notes - Impact of Delay in Voice over IP Services,* 2006.

[3] S. Haykin, Adapative filter Theory 3 edition, New Jersey: Pretice Hall, 1996.

[4] M. E. Knappe, *Acustic Echo Cancellation: Performance and Structures,* Ottawa: Carleton University, 1992.

[5] P. S. R. Diniz, E. A. B. da Silva e S. L. Netto, "Digital Signal Processing," em *Digital Signal Processing - System Analysis and Design*, United Kingdom, Cambridge University Press, 2010, pp. 58 - 59.

[6] B. C. Bispo, *Cancelamento de Eco Elétrico em Redes Telefônicas e Acústico em Sistemas de Teleconferência,* Rio de Janeiro: COPPE-UFRJ, 2008.

[7] D. L. Duttweiler, "Proportionate normalized least mean square adaptation in echo cancelers," *IEEE Transactions on Speech and Audio Processing,* pp. 508-518, September 2000.

[8] M. L. R. de Campos, P. S. R. Diniz e J. A. Apolinário Jr., "On normalized data-reusing and affine-projections algorithms," *6th IEEE Internacional Conference on Electronics, Circuits and Systems,* pp. 843-846, 1999.

[9] J. A. Apolinário Jr., M. L. R. de Campos e P. S. R. Diniz, "Convergence analysis of the binormalized data-reusing LMS algorithm," *IEEE Transactions on Signal Processing,* pp. 3235-3242, November 2000.

[10] ABNT, *ABNT NBR IEC 60079-10-1 Atmosferas Explosivas Parte 10-1: Classificação de Áreas - Atmosferas Explosivas de Gás,* ABNT, 2009.

[11] I. -. I. T. Union, "ITU-T Recommendation G.711," em *Pulse Code Modulation*

(PCM) of Voice Frequencies, Geneva, 1993.

[12] I. -. I. T. Union, "Recomendation ITU-T G.168 - International Telephone Connections and Circuits," em *Digital network eco cancellers*, 2012.

[13] I. -. I. T. Union, "ITU-T Recommendation P.800," em *Methods for subjective determination of transmission quality*, 1996.

[14] I. -. I. T. Union, "ITU-T Recommendation P.862," em *Perceptual evaluation of speech quality (PESQ)*, 2001.

[15] MathWorks, "Fixed-Point Designer," em *Matlab User's Guide R2013a*, Natick, MA, 2013, pp. 73-359.

[16] Mathworks, "Fixed-Point Designer Reference R2013a," em *Matlab User's Guide* , Natick, MA, 2013, pp. 1-20.

[17] Samsung Electronics, *S3C6410X RISC Processor User's Manual,* Yongin-City, Coréia, 2008.

[18] ARM, "AMBA Procotol Specifications," 2007 - 2010. [Online]. Available: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.set.amba/index.html. [Acesso em 21 05 2013].

[19] Das U-Boot, "U-Boot," [Online]. Available: http://www.denx.de/wiki/U-Boot. [Acesso em 25 05 2013].

[20] L. Torvalds, "The Linux Kernel Archives," [Online]. Available: https://www.kernel.org/. [Acesso em 25 05 2013].

[21] P. S. R. Diniz e S. Werner, "Set-membership binormalized data-reusing LMS algorithm," *IEEE Transactions on Signal Processing,* pp. 124 - 134, Janeiro 2003.

[22] F. Albu, M. Rotaru, R. Arablouei e K. Dogançay, "Intermittently-updated Affine Projection Algorithm," *IEEE Int. Conference on Acustics, Speech and Signal Processing,* 2013.

[23] Crosstool-ng, “Crosstool-ng,” [Online]. Available: http://crosstool-ng.org/. [Acesso em 21 05 2013].

[24] I. -. I. T. Union, “ITU-T - Recommendation P.862.2,” em *Wideband extention to recommendation P.862 for the assessment of wideband telephone networks and speech codecs*, 2005.