Fernando Pasquini Santos

Projeto de um Filtro para Equalização Sonora Monoponto

São Carlos, SP, Brasil

Junho de 2012

Fernando Pasquini Santos

Projeto de um Filtro para Equalização Sonora Monoponto

Trabalho de Conclusão de Curso apresentado à Escola de Engenharia de São Carlos, da Universidade de São Paulo, como parte da graduação no curso de Engenharia de Computação.

Orientador:
Carlos Dias Maciel

Escola de Engenharia de São Carlos
Universidade de São Paulo

São Carlos, SP, Brasil

Junho de 2012

Ficha catalográfica preparada pela Seção de Atendimento ao Usuário do Serviço de Biblioteca – EESC/USP

S237p

Santos, Fernando Pasquini
Projeto de um filtro para equalização sonora monoponto. / Fernando Pasquini Santos ; orientador Carlos Dias Maciel. São Carlos, 2012.

Monografia (Graduação em Engenharia de Computação) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2012.

1. Processamento de sinais. 2. Equalização. 3. Áudio. I. Título.

FOLHA DE APROVAÇÃO

Nome: Fernando Pasquini Santos

Título: "Projeto de Filtro para Equalização Sonora Monoponto"

Trabalho de Conclusão de Curso defendido e aprovado em 26 / 06 / 2012,

com NOTA dez (10 , 0), **pela comissão julgadora:**

Prof. M. Sc. Wagner Endo - UFTPR

M. Sc. Jen John Lee - SEL/EESC/USP

Prof. Associado Evandro Luís Linhari Rodrigues
Coordenador pela EESC/USP do
Curso de Engenharia de Computação

Pois também se a trombeta der som incerto, quem se preparará para a batalha?

Primeira Carta de Paulo aos Coríntios, cap. 14, v. 8

Dedicatória

Dedico este trabalho aos meus pais, Fernando e Viviani, que desde o começo da minha vida até os dias de hoje, sempre me orientaram com muito amor e sabedoria.

Agradecimentos

Agradeço primeiramente a Deus, o autor da maravilhosa criação chamada Som, que me deu alegria, força e capacidade para realizar o trabalho. "Dele, e por meio dele, e para ele são todas as coisas"(Rm 11.36).

Agradeço à Jemima, que me ofereceu paz e força nos momentos difíceis, e a todos os amigos que contribuíram com conhecimentos, conversas, conselhos, almoços, risadas e orações.

Agradeço aos professores que orientaram meus estudos durante minha graduação com muita competência, caráter, paciência e atenção, com destaque aos professores Carlos Dias Maciel e Maximiliam Luppe.

Agradeço ao Flávio Casagrande Hirono, que também com muita atenção ajudou na realização deste trabalho, contribuindo com conhecimento teórico e prático que considero não só importante para o trabalho, como também para a minha graduação.

Resumo

O presente trabalho propõe um estudo sobre filtros de equalização, na faixa de áudio, para o cancelamento do efeito sonoro introduzido por uma sala. Dois modelos de aproximação da resposta ao impulso de uma sala são discutidos, considerando-se também o filtro inverso correspondente a cada um deles. Um destes filtros, baseado no modelo auto-regressivo, é projetado em MATLAB e implementado em C sob ambiente Linux.

Palavras-chave: Processamento de Sinais. Equalização. Áudio.

Abstract

This paper proposes a study of equalization filters in the audio range, to cancel the sound effect introduced by a room. Two models for approximating the impulse response of a room are discussed, considering also the inverse filter corresponding to each of them. One of these filters, based on autoregressive model, is designed in MATLAB and implemented in C under Linux environment.

Keywords: Signal Processing. Equalization. Audio.

Sumário

Lista de Figuras

1 Introdução

1.1 Contextualização

Muitas pesquisas já foram feitas acerca da filtragem inversa aplicada a sistemas de processamento de áudio. A ideia principal explorada consiste em obter um sistema capaz de equalizar o som que chega ao ouvinte, eliminando distorções geradas, na maioria das vezes, pelo som refletido e/ou absorvido pelas paredes e objetos de uma sala.

O som não viaja apenas diretamente de um transmissor até um receptor, mas também pode ser refletido e absorvido por elementos presentes no ambiente, conforme mostra a figura 1. Logo, o som que chega até o receptor não é exatamente aquele foi produzido, mas outro som, que consiste numa soma de vários sinais que sofreram diferentes atenuações e atrasos, de acordo com o caminho que percorreram. Este efeito recebe o nome de reverberação, e ocorre em vários ambientes, principalmente em salas, conforme mostra a figura 1.

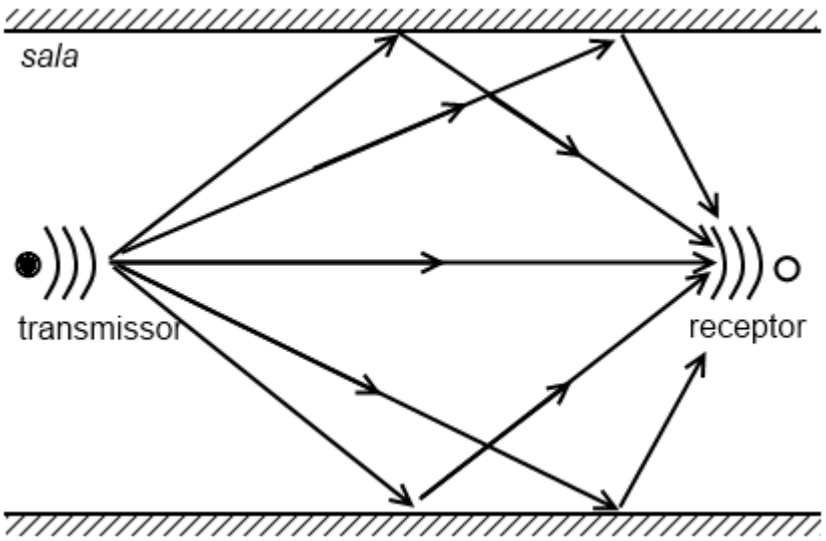

Figura 1: Efeito de reverberação que pode ser introduzido por uma sala, do transmissor até o receptor.

As distorções introduzidas por uma sala podem, então, ser modeladas e definidas como a resposta ao impulso do caminho do transmissor até o receptor do sinal sonoro. Esta resposta pode ser modelada de diversas maneiras, e vários trabalhos apontam métodos como modelos de média móvel (MA), auto-regressivos (AR), modelos auto-regressivos de média móvel (ARMA), filtros de Kautz [Paatero e Karjalainen 2002], e filtros adaptativos como filtros LMS [Haykin e Widrow 2003].

Defina-se $h(n)$ esta resposta. O filtro desejado, portanto, representa o inverso desta resposta, $h^{-1}(n)$,para que, quando convoluído com o sinal original, possa cancelar o efeito da sala ($h(n) * h^{-1}(n) = 1$). Este filtro é aplicado antes da transmissão do sinal, e anula o efeito introduzido, conforme o modelo da figura 2.

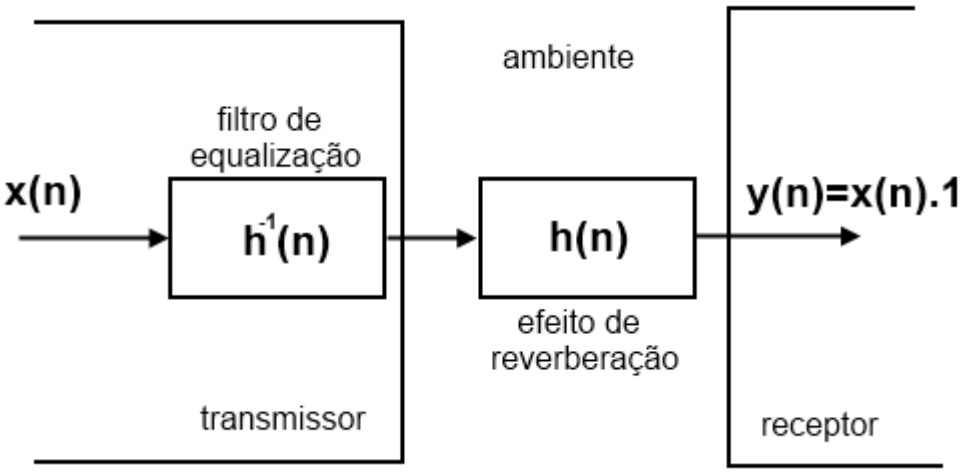

Figura 2: Modelo representando um filtro de equalização capaz de anular o efeito de reverberação introduzido pelo ambiente.

Para que este filtro inverso possa apresentar uma resposta estável, precisa-se garantir que ele seja um sistema de fase mínima. Um sistema de fase mínima é um sistema cuja transformada z possui ambos os polos e zeros no interior do círculo unitário. Uma vez que a condição de estabilidade de um sistema é a presença dos polos no interior do círculo unitário, quando desejamos um filtro inverso, precisamos aplicar esta condição também aos zeros, uma vez que esta inversa terá seus zeros no lugar dos polos e vice-versa. Várias técnicas existem e já foram pesquisadas para a obtenção de sistemas de fase mínima em filtragem inversa, entre as quais pode-se citar a decomposição de uma resposta em um componente de fase mínima, e outro componente de atraso puro [Oppenheim e Schafer 1998, p. 280].

1.2 Objetivo do Trabalho

O presente trabalho busca, portanto, estudar diferentes maneiras de se modelar o efeito introduzido por uma sala quando o som viaja de um transmissor até um receptor, e, juntamente com isso, obter um filtro inverso capaz de anulá-lo. Este filtro será baseado na função de transferência inversa do modelo encontrado para o ambiente, que deve ser de fase mínima, permitindo, assim, uma resposta estável.

1.3 Estrutura

A seção 2 irá lidar com a teoria estudada para dois modelos de resposta de uma sala (MA e AR), e como projetar o filtro inverso correspondente. A seção 3 discutirá os materiais e métodos utilizados para simulação, obtenção dos coeficientes e implementação de um filtro inverso baseado em uma resposta modelada pelo modelo AR. A seção 4 apresentará os resultados obtidos com o modelo AR e o filtro inverso correspondente, tanto na simulação como na implementação.

2 Fundamentação Teórica

2.1 Modelagem de um Ambiente de Reverberação Acústica

Conforme apontado na seção 1, existem diversos métodos para modelagem do efeito introduzido por uma sala (ou qualquer ambiente de reverberação acústica) em um som que viaja de um transmissor a um receptor. Um modelo é uma aproximação da relação entre um sinal recebido $y(n)$ e um sinal transmitido $x(n)$, e irá representar um sistema "caixa-preta"$h(n)$. Um modelo não precisa necessariamente caracterizar a resposta exata de uma sala, mas deve modelar os efeitos que se deseja anular com o filtro inverso, que será baseado na resposta inversa deste modelo.

Uma solução que pode ser proposta para a obtenção de $h(n)$ da sala é a de obter os coeficientes da DFT (*Discrete Fourier Transform*) de $x(n)$ e $y(n)$ e dividir uns pelos outros, obtendo a DFT da resposta $h(n)$, conforme a relação 2.1. Esta solução não é adequada, já que os coeficientes da DFT de $x(n)$ podem ser iguais a zero ou muito próximos de zero, causando desde erros de divisão por zero ou números muito grandes nos coeficientes da DFT de $h(n)$, gerando *overflow*, ou seja, números muito grandes que ultrapassam a capacidade de representação da máquina onde o processamento está sendo feito. A solução proposta na relação 2.1, apesar de teoricamente correta, é portanto, irrealizável. A aproximação de $h(n)$ deve ser feita por modelos baseados diretamente no domínio do tempo.

$$y(n) = x(n) * h(n) \xrightarrow{\text{DFT}} Y[k] = X[k]H[k] \Leftrightarrow H[k] = \frac{Y[k]}{X[k]} \tag{2.1}$$

Dois modelos básicos são propostos [Hayes 96] para a caracterização de um sistema a partir de suas entradas e saídas no domínio do tempo, e são eles o modelo de média móvel, abreviado como MA (*moving average*) e o modelo auto-regressivo, AR (*auto-regressive*). Como será observado adiante, estes modelos são complementares e conseguem descrever

a resposta de uma sala de maneiras diferentes, com o primeiro representando um filtro FIR (*finite impulse response*), e o segundo, um filtro IIR (*infinite impulse response*). A complementaridade destes modelos dá origem ao modelo ARMA (*auto-regressive moving average*), uma combinação que consegue unir a capacidade de modelagem de ambos os modelos, e pesquisas recentes [Paatero e Karjalainen 2002] apontam que ARMA é um modelo amplamente usado para modelagem e filtragem inversa na faixa de frequências de áudio.

As seções seguintes irão apresentar os modelos MA e AR e seus filtros inversos correspondentes, fazendo uma análise das suas capacidades de modelagem.

2.1.1 Modelo MA

O modelo MA representa uma das maneiras mais intuitivas de se descrever o comportamento da resposta de uma sala. Ele relaciona o sinal recebido $y(n)$ com o sinal transmitido $x(n)$ através de uma somatória de atrasos e atenuações de $x(n)$, conforme mostra a figura 3. O modelo é representado na equação 2.2.

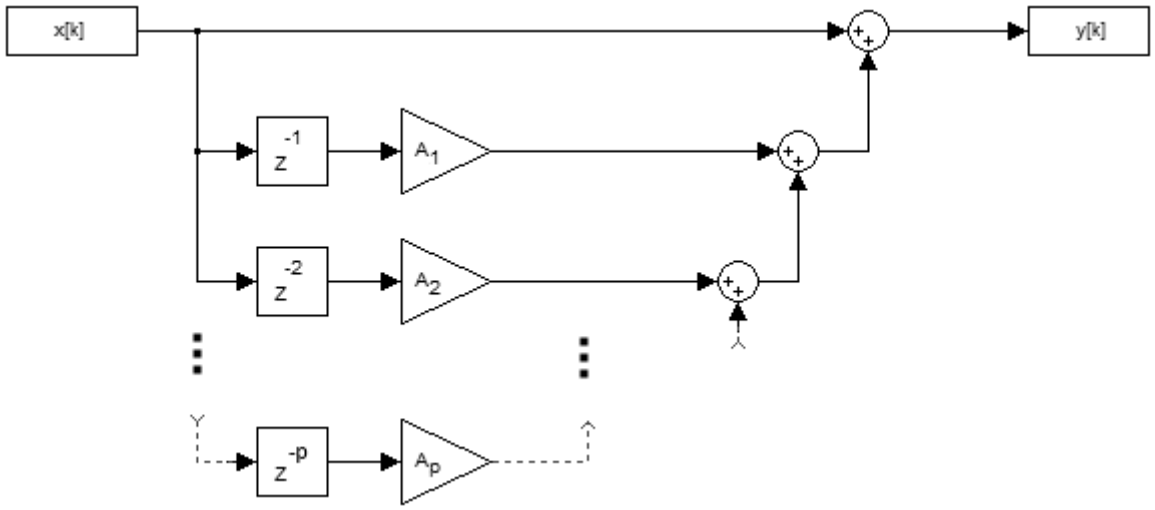

Figura 3: Diagrama de blocos de um modelo MA.

$$y(n) \approx x(n) + \sum_{i=1}^{p} A_i x(n-i) \tag{2.2}$$

onde p é a ordem do modelo MA.

A equação acima contempla o fato de que a saída do sistema equivale ao sinal original transmitido ($x(n)$) somado a valores anteriores desse mesmo sinal (até a amostra $n-p$), atenuados pelas constantes $A_i (i = 1, ..., p)$. Para uma descrição do efeito da sala sobre

o som transmitido, portanto, é necessário determinar as constantes A_i de forma que a equação 2.2 apresente o menor erro possível.

Aplicando a transformada-z sobre a equação 2.2, temos a seguinte relação:

$$Y(z^{-1}) = X(z^{-1}) + \sum_{i=1}^{p} A_i X(z^{-1}) z^{-i} \tag{2.3}$$

. Que, por sua vez, pode ser reorganizada de forma a representar a resposta ao impulso da sala $H(z^{-1})$ no domínio z:

$$\frac{Y(z^{-1})}{X(z^{-1})} = H(z^{-1}) = 1 + \sum_{i=1}^{p} A_i z^{-i} \tag{2.4}$$

.

Vários métodos são propostos para a obtenção dos coeficientes A_i deste modelo. Entre os dois principais utilizados, temos as equações de Yule-Walker [Hayes 96], um método amplamente utilizado tanto para aproximações de modelos MA (e que também pode ser generalizado para modelos AR e ARMA), e métodos iterativos de aproximação por minimização do erro LMS [Haykin e Widrow 2003].

O modelo MA apresenta limitações por ser de duração finita, ou seja, fornece coeficientes adequados somente com base na entrada $x(n)$ fornecida, e não acumula nenhuma informação sobre os sinais anteriores a $y(n)$. [Joaquim 2006] define este tipo de sistema como sendo um sistema sem memória. Embora o modelo tenha as vantagens de ser sempre estável e apresente uma característica de fase linear [Joaquim 2006], isso faz com que a aproximação ótima seja garantida somente para o intervalo de $x(n-p)$ a $x(n)$ analisado para o modelo, e para qualquer n em $x(n)$ fora do intervalo previsto, pode existir uma grande divergência entre $y(n)$ real e o aproximado. Assim, o modelo MA se torna inviável em muitos casos, pois exige uma ordem p grande para a aproximação, e oferece uma resposta sem memória para um sistema que pode apresentar memória.

Um efeito de reverberação introduzido por uma sala pode ser separado em dois componentes - distorção imediata e eco. O modelo MA, por não conter memória, é ruim para descrever efeitos de eco, uma vez este é uma repetição periódica de um trecho do sinal ao longo de toda a duração do som. No entanto, para a caracterização das distorções sofridas por amostras imediatas do sinal de entrada (de $x(n-p)$ a $x(n)$), o modelo MA se mostra bastante preciso. Isso ocorre pois ele se baseia na comparação de amostras atuais e anteriores do sinal de entrada $x(n)$ com o sinal de saída $y(n)$, o que já não ocorre no modelo AR, conforme será visto adiante.

2.1.2 Modelo AR

Complementando o modelo MA, o modelo AR é capaz de modelar o sinal $y(n)$ a partir de atrasos do próprio sinal $y(n)$ recebido, representando um modelo realimentado, conforme a figura 4. A equação 2.5 representa este modelo.

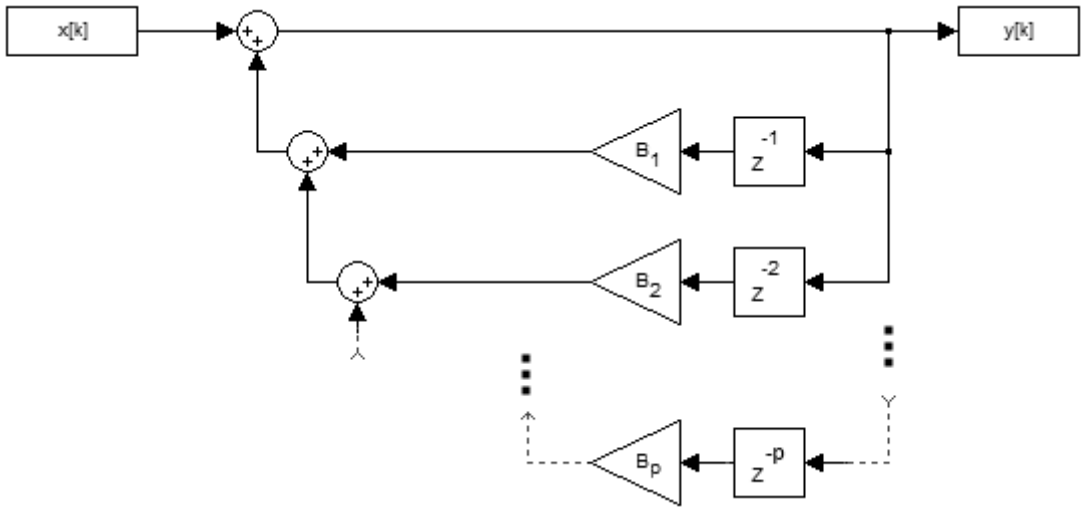

Figura 4: Diagrama de blocos de um modelo AR.

$$y(n) \approx x(n) + \sum_{i=1}^{p} B_i y(n-i) \tag{2.5}$$

A equação acima contempla o fato de que a saída do sistema equivale ao sinal original transmitido ($x(n)$) somado a valores anteriores do sinal recebido (até a amostra $n-p$), atenuados pelas constantes $B_i (i = 1, ..., p)$.

Aplicando a transformada-z sobre a equação 2.5, temos a seguinte relação:

$$Y(z^{-1}) = X(z^{-1}) + \sum_{i=1}^{p} B_i Y(z^{-1}) z^{-i} \tag{2.6}$$

. Que, por sua vez, pode ser reorganizada de forma a representar a resposta ao impulso da sala $H(z^{-1})$ no domínio z:

$$\frac{Y(z^{-1})}{X(z^{-1})} = H(z^{-1}) = \frac{1}{1 - \sum_{i=1}^{p} B_i z^{-i}} \tag{2.7}$$

.

A obtenção dos coeficientes para este modelo pode ser feita pelos mesmos métodos citados na seção anterior - Yule-Walker [Hayes 96], e métodos iterativos de aproximação por minimização do erro LMS [Haykin e Widrow 2003].

A vantagem introduzida pelo modelo AR é que este apresenta memória, e é capaz de modelar a resposta $h(n)$ em um intervalo infinito de tempo, e com poucos coeficientes. No entanto, o modelo AR oferece apenas memória, e não relaciona $y(n)$ com valores passados do sinal de entrada $x(n)$, tornando-se uma aproximação muito pouco precisa dos efeitos introduzidos pela sala no sinal $x(n)$ em intervalos de tempo imediatos. Conclui-se daí que o modelo AR é excelente para a caracterização de efeitos de eco, com a repetições periódicas por todo o sinal, porém inadequado para descrever as distorções imediatas introduzidas no sinal de entrada. Também pode-se concluir que as vantagens e desvantagens introduzidas pelos modelos AR e MA são complementares.

2.2 Filtros Inversos e Análise de Estabilidade

Uma vez com um modelo que caracteriza a resposta de uma sala a um som transmitido, a função de transferência do modelo pode ser invertida de modo a obter um filtro capaz de anular o efeito (conforme a figura 2). Conforme já comentado na introdução, alguns cuidados devem ser tomados na construção deste filtro, principalmente quanto à inversão da função de transferência, onde há também pólos e zeros invertidos. Isso pode tornar a resposta instável e/ou não causal, uma vez que a condição de estabilidade e causabilidade exige que os pólos da função de transferência, em função de z, estejam no interior do círculo unitário. Havendo inversão, devemos, portanto, também fazer esta exigência quanto aos zeros, e com esta exigência dizemos que a função de transferência do modelo da sala deve ser de fase mínima [Oppenheim e Schafer 1998], ou seja, com ambos os zeros e pólos no interior do círculo unitário. A presente seção irá apresentar as funções de transferência inversas dos modelos e a análise de estabilidade das mesmas.

2.2.1 A partir do modelo MA

A resposta inversa da equação 2.4, $H^{-1}(z^{-1})$, tem a forma:

$$H^{-1}(z^{-1}) = \frac{1}{1 + \sum_{i=1}^{p} A_i z^{-i}} \tag{2.8}$$

(fazendo com que $H(z^{-1}).H^{-1}(z^{-1}) = 1$).

Observa-se a troca dos zeros pelos pólos. Para garantir que o sistema da equação seja estável e causal, todos os pólos do sistema devem se localizar no interior do círculo unitário. Pode-se provar que o sistema anteriormente apresentado possui esta característica.

[Hayes 96] define que um processo aleatório $x(n)$ é estacionário no sentido amplo (*wide sense stationary*) quando 1) sua média é constante, 2) a autocorrelação $r_x(k,l)$ depende apenas da diferença $k-l$, e 3) a variância do processo é finita, $c_x(0) < \infty$. Estas condições nos garantem que a função de distribuição de probabilidade do processo não muda de acordo com o tempo ou a posição. Os sinais $x(n)$ e $y(n)$, transmitido e recebido, respectivamente, devem possuir esta característica, por se tratarem de sinais sonoros digitais e quantizados.

O modelo MA, sendo um filtro cuja entrada é um processo estacionário no sentido amplo (ruído branco), deve também fornecer uma saída estacionária no sentido amplo, caso contrário, este não representará um modelo adequado da sala. Pode-se garantir que um modelo MA é estacionário em sentido amplo quando o polinômio $z^p - \sum_{i=1}^{p} A_i z^{p-i}$, formado com os seus coeficientes, apresenta raízes no interior do círculo unitário [Takalo, Hytti e Ihalainen 2005]. Métodos como o de Yule-Walker, Burg e aproximação LMS [Bourke 1998] garantem que o processo continua estacionário no sentido amplo, e portanto, as raízes estão necessariamente no interior do círculo unitário. Nota-se que o modelo MA obtido por métodos de aproximação como Yule-Walker gera necessariamente um sistema com uma resposta de fase mínima.

O polinômio acima é o mesmo encontrado no denominador de $H^{-1}(z^{-1})$ e, portanto, fornece os pólos do sistema, que estarão no interior do círculo unitário. O filtro inverso é, então, garantidamente estável e causal.

2.2.2 A partir do modelo AR

A resposta inversa da equação 2.7, $H^{-1}(z^{-1})$, tem a forma:

$$H^{-1}(z^{-1}) = 1 - \sum_{i=1}^{p} B_i z^{-i} \tag{2.9}$$

(fazendo com que $H(z^{-1})H^{-1}(z^{-1}) = 1$).

Nota-se que para conseguirmos um modelo AR que permita entradas e saídas estacionárias em sentido amplo, há também a exigência de que o polinômio $z^p - \sum_{i=1}^{p} B_i z^{p-i}$, formado com os seus coeficientes, apresente raízes no interior do círculo unitário [Takalo, Hytti e Ihalainen 2005]. As raízes deste polinômio também indicam os pólos do modelo AR, o que mostra que a resposta da sala é estável e causal.

Com a inversão desta resposta, os zeros trocam de lugar com os pólos. Os zeros

encontrados na resposta da sala encontram-se na posição 0 do círculo unitário, com multiplicidade p (ordem do modelo). Com a inversão e consequente transformação em pólos, a resposta continua estável e causal.

2.3 Filtros FIR

Tendo obtido, então, esta função de transferência inversa, deve-se procurar uma forma de implementar um filtro real que a represente. Algumas vezes é impossível obter uma implementação precisa da função de transferência obtida para um sistema, dada a limitação dos dispositivos eletrônicos e processadores digitais de sinais. Uma aproximação da função deve, portanto, ser gerada.

A forma mais simples de se conseguir um filtro real a partir de uma função de transferência em z é utilizar filtros FIR (*finite impulse response*), que são exatamente os modelos de média móvel descritos na seção anterior. Filtros FIR, ou filtros não-recursivos, são descritos por relações que descrevem a saída do sistema apenas em função de valores presentes e passados da entrada [Joaquim 2006], e são filtros de fácil implementação em ambientes de DSP, inclusive otimizados através de estruturas de hardware conhecidas como MAC (*Multiply-and-Accumulate*) [Yin 2011].

Com isso, pode-se concluir que os coeficientes do filtro FIR inverso para um modelo AR de uma sala são exatamente os coeficientes A_i. Porém, no caso de um filtro FIR inverso para um modelo MA, devemos realizar o cálculo dos coeficientes através de técnicas de janelas. Na versão mais simples desta técnica, através de janelas retangulares, a ideia básica consiste em avaliar a resposta ao impulso do filtro somente uma determinada quantidade de pontos, de 0 até certo valor n inteiro, fornecendo os coeficientes do filtro. Nota-se que quanto mais coeficientes, melhor será a aproximação do filtro.

3 Materiais e Métodos

Na presente seção serão discutidos os materiais e métodos utilizados para o projeto de um filtro FIR de equalização sonora utilizando uma aproximação por modelo AR de uma sala. Obviamente um modelo AR não conseguirá caracterizar todo o efeito introduzido por uma reverberação, no entanto, será suficiente para anular o eco produzido. Dois experimentos preliminares são feitos no MATLAB e em seguida um filtro FIR é desenvolvido em MATLAB e C sob plataforma Linux. Os resultados dos métodos e algoritmos desenvolvidos serão apresentados na seção 4.

3.1 Desenvolvimento em MATLAB

O projeto do filtro FIR de equalização foi feito primeiramente no ambiente MATLAB, para fins de testes e simulação. A razão disso está no fato de que esta ferramenta provê meios simples e eficientes de aplicar os métodos descritos na seção anterior, principalmente no cálculo dos coeficientes do modelo AR, e também provê várias facilidades para a geração de sinais, simulação de efeitos de eco e plotagem de gráficos.

Dois experimentos foram feitos sobre modelos AR e a anulação de eco em sistemas de áudio. O primeiro consiste em simular o efeito de eco em um sinal de impulso unitário na entrada, e retirá-lo com o projeto de um filtro inverso simples. O segundo apresenta um sinal de som real, gravado em um *software* de música, e com um efeito de eco aplicado por uma ferramenta de edição de áudio [Audacity 2012]; onde um filtro inverso também é calculado e aplicado.

3.1.1 Filtro FIR em um Sinal de Impulso Unitário com Eco

O modelo apresentado na figura 5 foi desenvolvido com a ferramenta Simulink e simula um modelo auto-regressivo que introduz eco no sinal de entrada. Nota-se que a repetição do eco é periódica a cada 5 amostras. O tempo de amostragem utilizado no modelo é de

0.25s, o que, avaliado entre os tempos 0 a 20s, gera sinais de entrada $x(n)$ e saída $y(n)$ com 80 amostras.

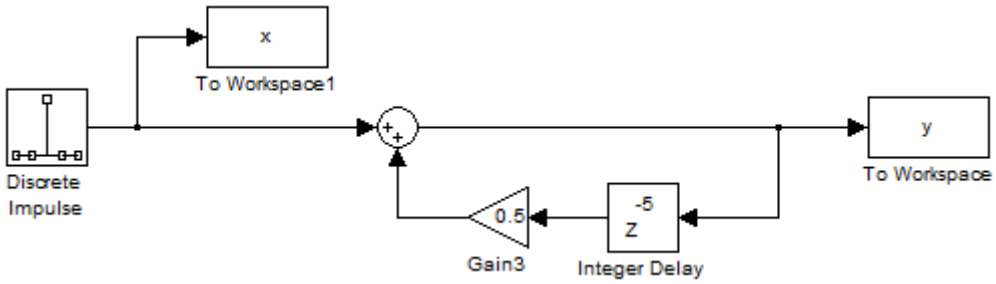

Figura 5: Diagrama de blocos em Simulink capaz de gerar amostras de um impulso unitário com eco.

Uma vez com as amostras geradas, o algoritmo a seguir é aplicado. A explicação do algoritmo se dá nas próprias linhas do código, porém convém destacar a função *ar*, utilizada no código. A função *ar*, fornecida no pacote padrão do MATLAB, recebe como argumentos o sinal que se procura aproximar ($y(t)$) e a ordem do modelo auto-regressivo, e retorna uma estrutura do tipo *idpoly*, contendo um polinômio cujos coeficientes são exatamente aqueles do modelo auto-regressivo. A função *polydata* extrai os coeficientes de *idpoly*.

```
1  % x: sinal sonoro sem o efeito da sala
2  % y: sinal sonoro com o efeito da sala
3  % order: ordem do modelo autoregressivo a ser usado na aproximação
4  % ta: tempo de amostragem
5  order=5;
6  ta = 0.25;
7
8  % estima o modelo AR da sala a partir de Yule-Walker
9  poly = ar(y,order,'yw');
10 % obtém os coeficientes do modelo
11 A = polydata(poly);
12
13 % filtro inverso Hsys
14 Hsys = tf(A,1,ta);
15
16 % plota o diagrama de zeros e pólos do filtro inverso
17 figure;
18 zplane(A,1);
19
```

```
20 % obtém coeficientes do filtro FIR com uma janela de 10 pontos
21 % (desnecessário, pois A já é o filtro FIR)
22 [h,t] = impz(A, 1, 10, 1/ta);
23
24 % x2 é o sinal y filtrado (deve se parecer com x, já que o eco é
       retirado)
25 x2 = conv(y,h);
26 % o tamanho de x2 é ajustado (apenas para fins de plotagem)
27 x2 = x2(1:length(x));
28
29 % plotagem de alguns resultados
30 figure;
31 stem(1:length(y),y);
32 hold on;
33 stem(1:length(x2),x2,'r');
34 legend('y','x2');
35 title('Sinal recebido e sinal recuperado','FontSize',12);
36 hold off;
37 figure;
38 stem(1:length(x), x2 - x);
39 legend('erro');
40 title('Erro entre o sinal recuperado e o sinal transmitido','FontSize
       ',12);
```

3.1.2 Filtro FIR em um Sinal de Som Real com Eco

O segundo experimento consiste em retirar o efeito de eco de um arquivo de som real, gravado por um *software* de edição de áudio. O som é gravado no formato *stereo*, com dois canais de áudio, porém no presente experimento será utilizado apenas um canal, para simplificar o ensaio.

O arquivo de som com eco, *virtual_ echo21.aif*, é produzido a partir do arquivo *virtual_ off.aif*, com a aplicação do efeito sendo feita pelo *software* de edição Audacity [Audacity 2012]. O software provê uma interface gráfica que permite a entrada de dois valores: o atraso, em segundos, e o fator de decaimento do sinal. O primeiro parâmetro define o intervalo de tempo no qual haverá a repetição periódica do sinal, e o segundo parâmetro define um coeficiente de multiplicação que será aplicado a cada repetição (causando ate-

nuação do som). Assim, os valores escolhidos para o atraso e fator de decaimento foram $0,3s$ e $0,4$, respectivamente, fazendo com que o sinal sonoro apresente uma repetição periódica a cada $0,3s$, e multiplicado por $0,4$ a cada nova repetição.

Os arquivos de som têm uma taxa de amostragem de $44,1kHz$, uma taxa comum em CDs de áudio, e são lidos pela pela função AIFFREAD, disponibilizada por [Eaton 2009]. Como o eco introduzido pelo *software* de edição é periódico a cada $0,3s$, temos um intervalo de $44100 \times 0,3 = 13230$ amostras a cada repetição. Assim, o filtro de equalização não pode ser de ordem menor que 13230, pois caso contrário, ele não seria capaz de descrever o efeito de eco introduzido. Aproximar um modelo auto-regressivo de ordem 13230 através de métodos como Yule-Walker é um processo extremamente dispendioso computacionalmente, de forma que, neste caso, e uma vez que conhecemos o efeito de eco aplicado no som, podemos declarar que o modelo auto-regressivo que gera o eco é da forma:

$$H^{-1}(z) = \frac{1}{1 - 0.4z^{-13230}} \tag{3.1}$$

O algoritmo abaixo é, então, aplicado de forma a obter um filtro FIR de equalização e analisar os resultados de sua aplicação sobre um arquivo de som com eco. O algoritmo é explicado nos próprios comentários.

```
% x_in e y_in: sinais sonoros com os dois canais
% x e y: primeiro canal dos sinais sonoros
% order: ordem do modelo autoregressivo a ser usado na aproximação
% fa: tempo de amostragem
[y_in,fs,nbits,trash] = aiffread('virtual_echo21.aiff');
[x_in,fs,nbits,trash] = aiffread('virtual_off.aif');
y_in = double(y_in);
x_in = double(x_in);
fa = 44100;

% retira o primeiro canal dos sinais sonoros
x=x_in(:,1);
y=y_in(:,1);

% coeficientes do modelo auto-regressivo
A = [1 zeros(1,13229) -0.4];

```

```
18 % resposta ao impulso da sala
19 [h_eco,t] = impz(1, A, 13231, fa);
20 % plota a resposta ao impulso da sala (para fins de análise)
21 figure;
22 stem(t,h_eco)
23 title('Resposta ao Impulso da Sala','FontSize',12);
24
25 % resposta ao impulso do filtro inverso
26 [h_filtro,t] = impz(A, 1, 13231, fa);
27
28 % aplica o filtro inverso e obtém x2, o sinal x recuperado
29 x2 = conv(y,h_filtro);
30 x2 = x2(1:length(x));
31
32 % calcula o erro rms entre o sinal recuperado e o transmitido
33 display('Erro RMS: %d');
34 rms = sqrt(sum((x2-x).^2)/length(x));
35 display(rms);
36
37 % comparação entre os sinais transmitido, recebido e recuperado
38 figure;
39 subplot(3,1,1);
40 plot(1:length(x),x);
41 legend('x');
42 title('Sinal transmitido','FontSize',12);
43 subplot(3,1,2);
44 plot(1:length(y),y);
45 legend('y');
46 title('Sinal recebido','FontSize',12);
47 subplot(3,1,3);
48 plot(1:length(x2),x2);
49 legend('x2');
50 title('Sinal recuperado','FontSize',12);
```

3.2 Implementação do filtro FIR em Linux

Após os experimentos com o projeto de filtros FIR para a equalização de eco, um projeto de filtro em MATLAB e C foi feito para anular o eco de um arquivo de som real. O cálculo dos coeficientes do filtro foi feito através do MATLAB, uma vez que não foi encontrada uma solução satisfatória para a análise e aproximação de um modelo AR na linguagem C. Assim, temos estes coeficientes exportados em arquivos de texto e posteriormente lidos pelo programa em C, que será responsável por apenas aplicar o filtro FIR e gravar os resultados na saída do dispositivo de áudio do sistema em que executa. Este segundo programa, em C, é implementado sob uma plataforma Linux e utiliza a biblioteca ALSA para lidar com os *drivers* de som, o que torna necessário explicar a razão do uso destas ferramentas nas seções 3.2.2 e 3.2.3.

3.2.1 Cálculo dos Coeficientes, em MATLAB

O arquivo de som em questão é o mesmo utilizado no segundo experimento, em MATLAB, nas seções anteriores. No entanto, neste caso os coeficientes são calculados pelo comando *ar*, do MATLAB, o que exige um tempo de processamento bastante longo.

Como temos um som no formato *stereo*, com dois canais de áudio, temos a necessidade de gerar um filtro inverso para cada canal. Assim, o algoritmo é aplicado duas vezes, e os coeficientes do filtro são gravados em dois arquivos, *h1.out* e *h2.out*.

Para tentar reduzir o tempo de processamento, foram feitas algumas tentativas. Utilizando a função *decimate*, do MATLAB, tentou-se reduzir o número de amostras do sinal em 50 vezes, e, já que o número de coeficientes do filtro agora pode ser menor, foi escolhido um número de $44100/dec = 882$ coeficientes. No entanto, várias amostras do sinal foram descartadas (temos um filtro de Chebyshev passa-baixas sendo aplicado), e com isso os sinais perderam várias de suas frequências mais altas, e com isso já é possível prever que o resultado não será satisfatório.

O algoritmo aplicado é dado abaixo. A explicação é dada nos próprios comentários do código.

```
% x e y: sinais sonoros com os dois canais
% order: ordem do modelo autoregressivo a ser usado na aproximação
% fa: tempo de amostragem
[y_in,fs,nbits,trash] = aiffread('virtual_echo21.aiff');
[x_in,fs,nbits,trash] = aiffread('virtual_off.aif');
```

```
6  y_in = double(y_in);
7  x_in = double(x_in);
8  fa = 44100;
9
10 % size indica a quantidade de amostras a serem levadas
11 % em consideração, para reduzir o tempo de processamento
12 size = 240000;
13 y = zeros(size,2);
14 x = zeros(size,2);
15 for i=1:size
16     y(i,1) = y_in(i,1);
17     y(i,2) = y_in(i,2);
18     x(i,1) = x_in(i,1);
19     x(i,2) = x_in(i,2);
20 end
21
22 % dec: fator de decimação
23 % x_d e y_d: sinais com a decimação
24 dec = 50;
25 y_d = zeros(size/dec,2);
26 x_d = zeros(size/dec,2);
27 x_d(:,1) = decimate(x(:,1),dec);
28 x_d(:,2) = decimate(x(:,2),dec);
29 y_d(:,1) = decimate(y(:,1),dec);
30 y_d(:,2) = decimate(y(:,2),dec);
31
32 % p é a ordem do modelo auto-regressivo
33 p = ceil((44100/dec));
34 poly1 = ar(y_d(:,1),p,'yw');
35 poly2 = ar(y_d(:,2),p,'yw');
36
37 % coeficientes do modelo auto-regressivo
38 A1 = polydata(poly1);
39 A2 = polydata(poly2);
40
41 % calcula a resposta ao impulso do modelo
42 [h1,t] = impz(A1, 1, p, fa/dec);
43 [h2,t] = impz(A2, 1, p, fa/dec);
```

```
44
45 % plota a resposta ao impulso do filtro inverso
46 [heco1,t] = impz(1, A1, p, fa/dec);
47 plot(heco1);
48 title('Resposta ao Impulso do Modelo AR Calculado','FontSize',12);
49
50 % grava a resposta ao impulso h em arquivos
51 out1 = fopen('h1.out','w');
52 out2 = fopen('h2.out','w');
53 fprintf(out1,'%0.5f\n',h1);
54 fprintf(out2,'%0.5f\n',h2);
55 fclose(out1);
56 fclose(out2);
57
58 % apenas para fim de testes, realiza a convolução
59 % com o sinal original
60 x1_d = conv(y_d(:,1),h1);
61 x1_d = x1_d(1:length(x_d(:,1)));
62 x2_d = conv(y_d(:,2),h1);
63 x2_d = x2_d(1:length(x_d(:,2)));
64
65 % comparação entre os sinais transmitido, recebido e recuperado
66 figure;
67 dist = length(x_d(:,1));
68 subplot(3,1,1);
69 plot(1:dist,x_d(1:dist));
70 legend('x');
71 title('Sinal transmitido','FontSize',12);
72 subplot(3,1,2);
73 plot(1:dist,x1_d(1:dist));
74 legend('x1');
75 title('Sinal recebido','FontSize',12);
76 subplot(3,1,3);
77 plot(1:dist,y_d(1:dist));
78 legend('y');
79 title('Sinal recuperado','FontSize',12);
80 figure;
81 plot(1:length(x),x2-x);
```

```
82 title('Erro entre os sinais transmitido e recuperado');
```

3.2.2 Linux Embarcado

Após a obtenção dos arquivos *h1.out* e *h2.out*, um programa em C é desenvolvido em Linux, com a proposta de se implementar o filtro em um ambiente de DSP.

Sabe-se que muitas plataformas de DSP atuais necessitam de tempo para serem estudadas e assimiladas, de forma que, dado o escopo do projeto, o estudo da plataforma e a posterior implementação seria inviável. Assim, o sistema operacional Linux aparece como uma solução satisfatória, já que apresenta um ambiente de programação bastante comum e uniforme, na linguagem C, e que além de funcionar em PCs convencionais, vem crescendo em seu uso em sistemas embarcados, inclusive em DSPs. Uma das distribuições mais comuns para Linux embarcado é a chamada Ångström [The Ångström Distribution 2012].

Linux, portanto, apresenta várias vantagens que contribuíram para a escolha da solução do presente projeto. Estas vantagens também são a causa do amplo crescimento do uso de Linux embarcado nos dias atuais, e algumas delas são listadas por [Hallinan 2010], entre as quais podemos citar:

- Linux se tornou uma alterativa madura, estável, e com alto desempenho em relação aos sistemas operacionais embarcados proprietários, além de não precisar de *royalties* para ser distribuído;
- Linux é escalável, e funciona desde dispositivos pequenos voltados para consumidores até sistemas de processamento pesado (como DSPs de ponto flutuante em tempo real).
- Linux tem atraído um enorme número de desenvolvedores, possibilitando um suporte rápido para novas arquiteturas de hardware, plataformas e dispositivos (constante desenvolvimento de *drivers* para dispositivos de áudio).

Além disso, Linux embarcado é um sistema operacional, e retira a necessidade de uma compreensão profunda de cada sistema onde se deseja implementar um programa. Os detalhes de *hardware*, incluindo os *drivers* de dispositivos (abstração de *software* que permite operar os dispositivos periféricos presentes no sistema), são tratados diretamente no sistema operacional. Com isso, pode-se desenvolver vários aplicativos em PCs convencionais, e transportá-los para uma aplicação embarcada de uma maneira rápida e simples.

Assim, embora o programa desenvolvido no presente trabalho não tenha sido portado para um sistema de Linux embarcado específico, pouco esforço seria necessário para isso. O procedimento necessário para portar uma aplicação desenvolvida em PCs para um ambiente Linux consistiria apenas na compilação cruzada para o microcontrolador ou DSP em questão. Arquiteturas modernas *multi-core* até mesmo oferecem compiladores capazes de dividir o código em trechos de código críticos, executados no DSP, e trechos de controle e I/O, executados no microcontrolador. Exemplo disso é o compilador *C6EZRun* [Texas Instruments 2012], desenvolvido pela Texas Instruments, que produz aplicações para a família de SoCs OMAP [Texas Instruments 2012], também da Texas Instruments.

3.2.3 Utilização da API ALSA

Diante da necessidade de desenvolver programas que lidam com os *drivers* de áudio em sistemas Linux, também fez-se necessário utilizar uma API chamada ALSA. ALSA [ALSA Project 2012] significa *Advanced Linux Sound Architecture*, e consiste em uma API e uma série de *drivers* que permitem o suporte a som em programas desenvolvidos sob plataforma Linux. Através de funções de baixo nível, a API oferece um controle amplo das funcionalidades suportadas pelos drivers de audio de um dispositivo [Tranter 2004]. ALSA também é suportado em muitas plataformas de Linux embarcado, e tem se tornado cada vez mais comuns em aplicações embarcadas de processamento de áudio.

[Tranter 2004] fornece vários programas simples, como exemplo, utilizando a API ALSA. Um deles ilustra a execução de um arquivo de som, no formato *.raw*, no dispositivo padrão de áudio aceito no sistema onde o programa executa. Este programa de exemplo, então, é tomado como base para o programa em C desenvolvido, e é modificado pelo autor de forma a satisfazer os objetivos do presente projeto, abrindo o arquivo de som, aplicando o filtro FIR, e executando o resultado no dispositivo de som.

3.2.4 Aplicação do Filtro FIR, em C

O código desenvolvido em C, que utiliza Linux e ALSA para a aplicação do filtro FIR, é apresentado no Apêndice A.

O código utiliza o *frame buffer* do *driver* de áudio do dispositivo, através da biblioteca ALSA, para enviar os dados processados em tempo real ao dispositivo de som, à medida que lê os arquivos de entrada. *Frame buffer* é um espaço de dados alocado no dispositivo de áudio onde o sistema operacional pode gravar os dados de som, sendo estes dados lidos pelo

dispositivo de som, que os converte em sinais sonoros. O programa lê, portanto, blocos de dados do tamanho do *frame buffer* alocado, aplica o filtro FIR com uma convolução simples, e grava os resultados no *frame buffer*.

A razão da leitura e processamento em blocos (enquanto temos a opção de ler o arquivo de som inteiro e aplicar o filtro a todo o sinal), existe pois a leitura do arquivo pode ser trocada a qualquer momento por código da API ALSA capaz de amostrar o som recebido no microfone (*frame buffer* de entrada), o que forneceria uma aplicação do filtro em tempo real. Nota-se também que a aplicação do filtro em blocos de dados faz com que o sinal de saída não seja uma convolução completa do sinal de entrada, porém ainda assim conta com uma boa aproximação (quanto maior o tamanho do bloco de dados, melhor a aproximação).

Algumas tentativas foram feitas no sentido de se portar o programa para uma plataforma de Linux embarcado em DSP. O programa foi compilado através da ferramenta *C6EZRun* [Texas Instruments 2012], e transferido para o sistema de arquivos de um sistema Linux, da distribuição Ångström, executando numa placa *BeagleBoard*, uma plataforma de desenvolvimento embarcado *open-source* bastante popular nos dias atuais.

A placa BeagleBoard [Coley 2012] apresenta um core OMAP3530, saída de vídeo (HDMI ou S-Vídeo), entrada/saída de áudio, *slot* para cartão SD, serial RS-232, *headers* de JTAG e USB *on-the-go*. Seu baixo consumo de energia e o tamanho reduzido ($7,2 \times 7,2cm$) também contribuem para que este seja um sistema com bastante potencial para desenvolvimento de aplicações de Linux embarcado e processamento digital de sinais. O processador OMAP3530 [Texas Instruments 2012], da Texas Instruments, é um processador *multi-core* híbrido, com um processador ARM Cortex-A8 de $720MHz$, um processador digital de sinal de ponto fixo TMS320C64x+ de $520MHz$ e um processador de vídeo POWERVR SGX. A ferramenta *C6EZRun* permite compilar trechos de código para executar em cada um destes cores, otimizando o desempenho.

4 Resultados e Discussão

A presente seção apresentará os resultados obtidos nos experimentos em MATLAB e na implementação do filtro inverso em Linux, com o cálculo dos coeficientes em MATLAB. Uma vez com estes resultados, também será feita uma discussão e análise dos mesmos à luz dos objetivos do trabalho.

4.1 Experimentos realizados em MATLAB

4.1.1 Filtro FIR em um Sinal de Impulso Unitário com Eco

A execução do algoritmo descrito na seção 3.1.1, do primeiro experimento, contendo um sinal senoidal na entrada e a inserção de um atraso de cinco amostras atenuado em 0.5, gerou a resposta ao impulso inversa apresentada na equação 4.1:

$$H^{-1}(z) = \frac{1}{1 - 0.5z^{-5}} \tag{4.1}$$

(com ordem 5, no modelo auto-regressivo, e o tempo de amostragem $T_a = 0,25s$, amostras de $t = 0...20s$).

A figura 6 mostra o diagrama de pólos e zeros da resposta acima. Pode-se observar que todos os pólos estão no interior do círculo unitário, indicando que temos uma resposta estável.

A figura 7 mostra uma comparação entre o sinal transmitido e o sinal recuperado. A figura 8 apresenta o erro entre o sinal transmitido (x) e o recuperado com filtro inverso ($x2$), e mostra que o erro é praticamente inexistente, da ordem de 10^{-10}, o que indica talvez apenas um erro de aproximação nos cálculos do MATLAB. O filtro inverso, portanto, conseguiu recuperar com precisão o sinal transmitido.

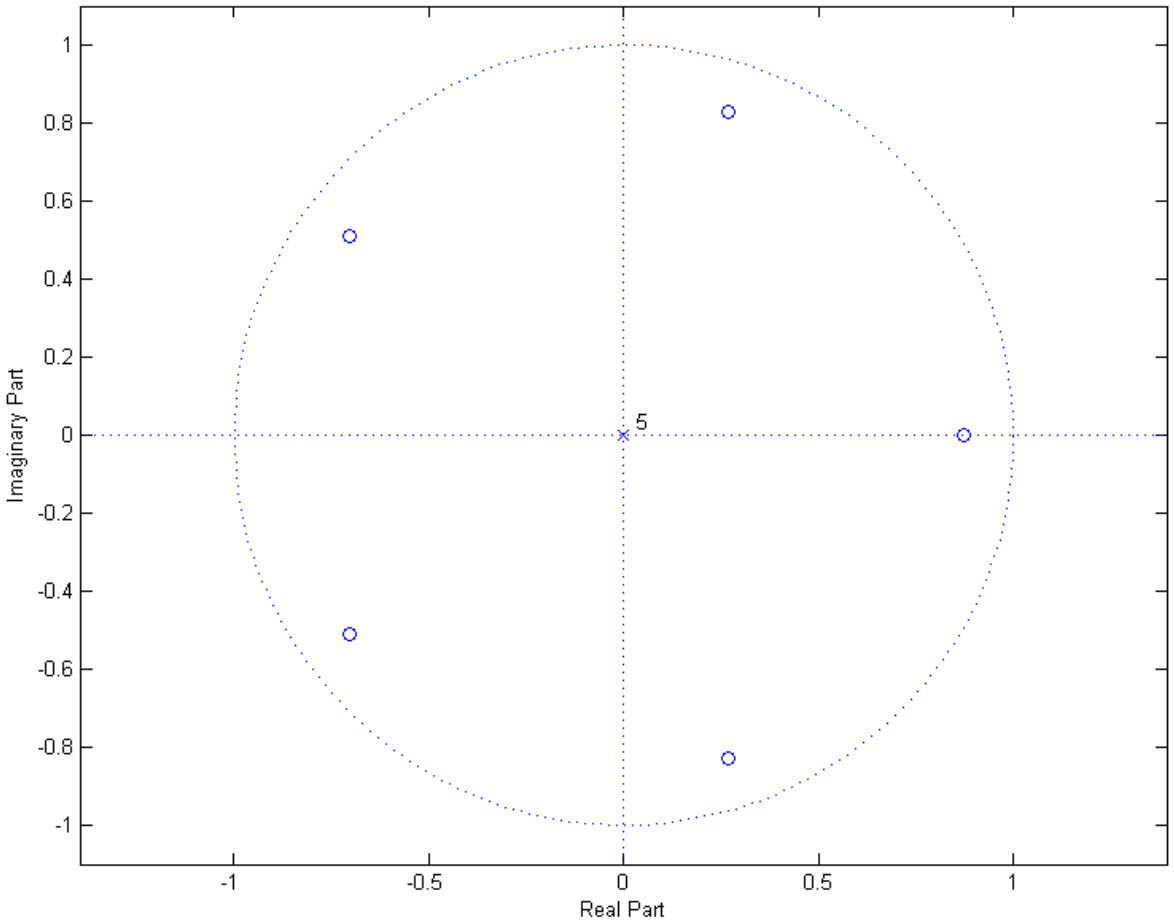

Figura 6: Diagrama de pólos e zeros para a resposta ao impulso do filtro inverso obtido no primeiro experimento.

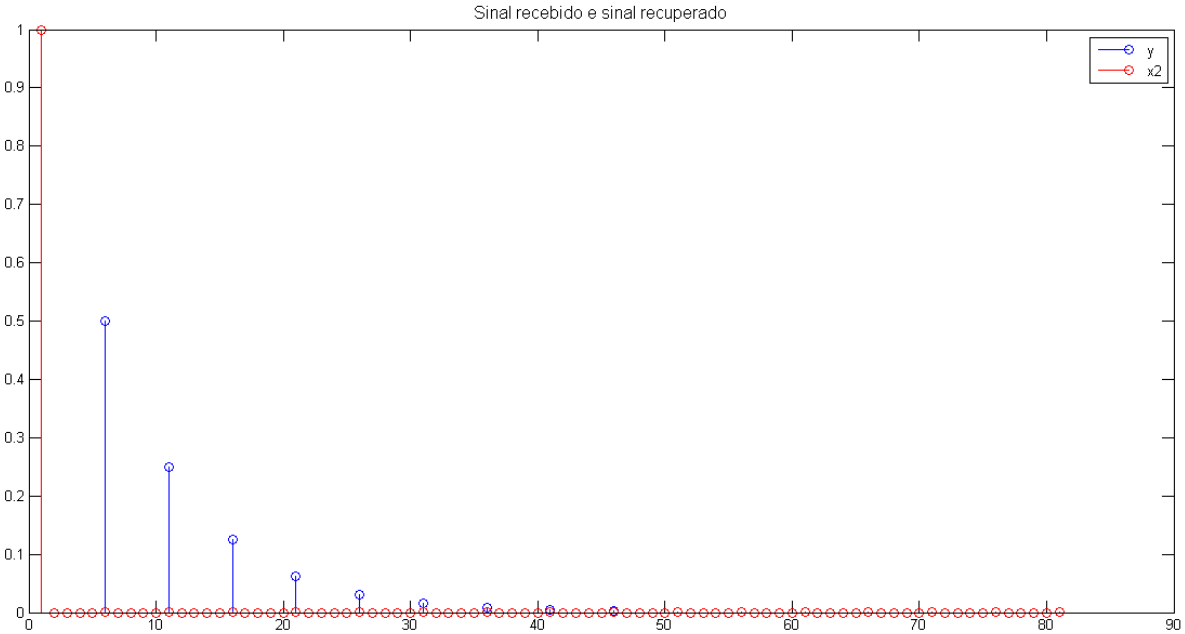

Figura 7: Comparação entre o sinal transmitido e o sinal recuperado (primeiro experimento).

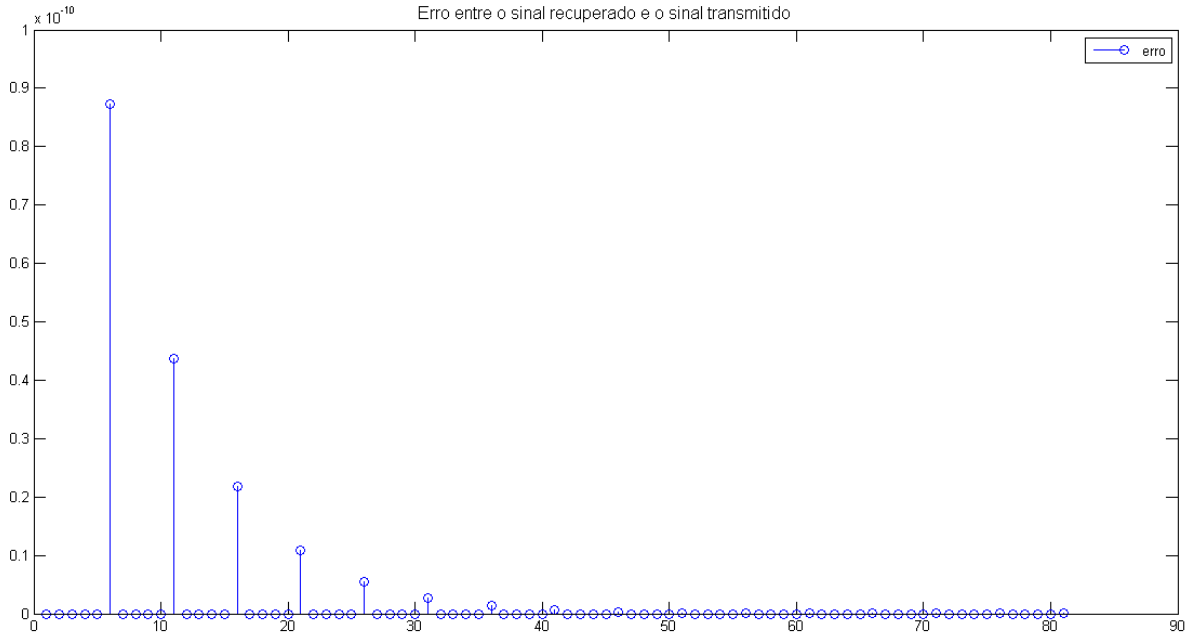

Figura 8: Erro entre o sinal recuperado e o sinal transmitido (primeiro experimento).

4.1.2 Filtro FIR em um Sinal de Som Real com Eco

O modelo que foi estimado na seção 3.1.2, do segundo experimento, gerou o gráfico de resposta ao impulso da figura 9, através da função $stem(t, h\_eco)$, na linha 22.

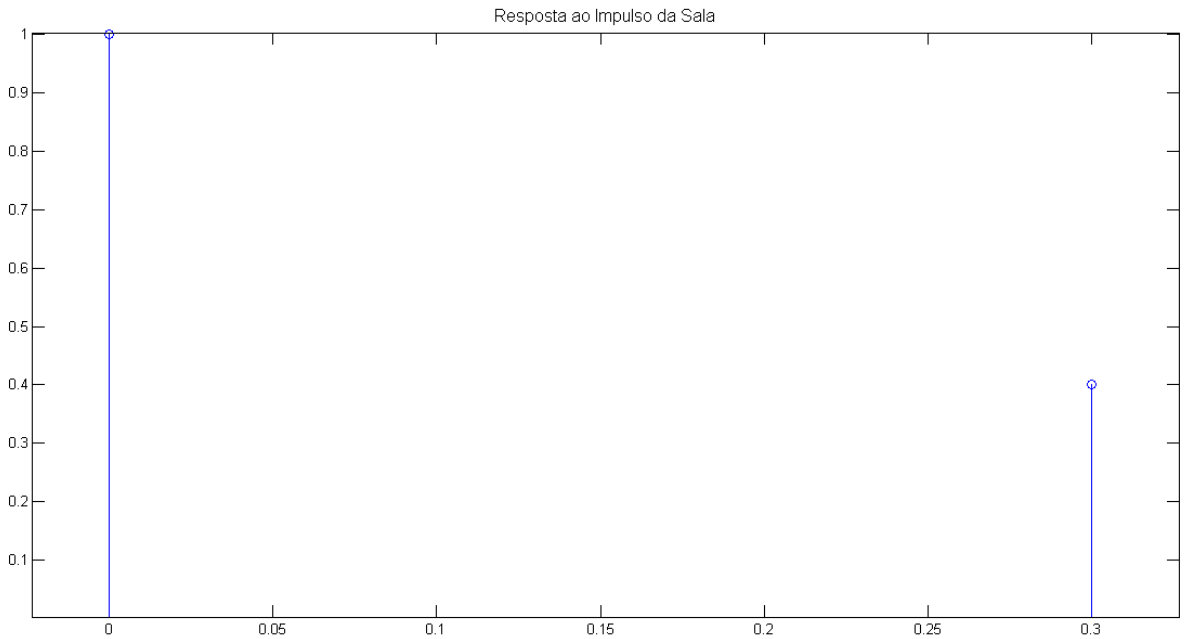

Figura 9: Resposta ao impulso do modelo da sala (antes da inversão) (segundo experimento).

Com a inversão deste modelo e aplicação do filtro no sinal y, temos um sinal $x2$, que

deverá ser próximo do sinal x. A semelhança é facilmente notada (perceptível ao) e o erro é mínimo. A figura 10 apresenta o erro entre os sinais $x2$ e x, que não passa do valor de 10. Uma vez que os valores em questão estão no intervalo de -32768 a 32768 (tipo *"signed short"*, de 16 bits) este erro não ultrapassa $0,01\%$. Não obstante, faz-se o cálculo do erro médio quadrático (ou erro RMS) entre os sinais, com base na equação 4.2, e o resultado equivale a $2,18$ (erro de $0,003\%$).

$$RMS = \sqrt{\frac{\sum_{i=1}^{n}(x_2(i) - x(i))^2}{n}} \tag{4.2}$$

.

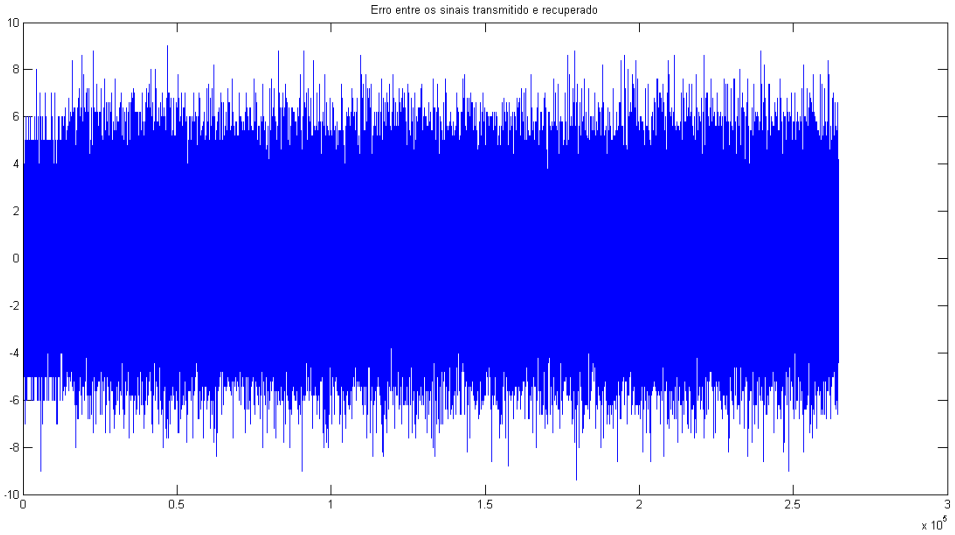

Figura 10: Erro entre os sinais recuperado ($x2$) e o sinal transmitido (x)

4.2 Aplicação do Filtro FIR, em C

4.2.1 Coeficientes do filtro FIR em MATLAB

O método de Yule-Walker aplicado para estimar o modelo AR da sala gerou uma resposta ao impulso da sala na forma da figura 11.

Como pode-se observar, o modelo não foi capaz de estimar exatamente o efeito do eco, como feito no experimento anterior, e isso ocorre uma vez que o modelo tem uma ordem muito alta, fazendo com que o método de Yule-Walker se baseie muito mais nas primeiras amostras do que nas últimas. Apesar disso, o modelo foi suficiente para retirar o eco no sinal decimado, embora também tenha eliminado, junto com o eco, partes do sinal sonoro. A figura 12 mostra o sinal recuperado.

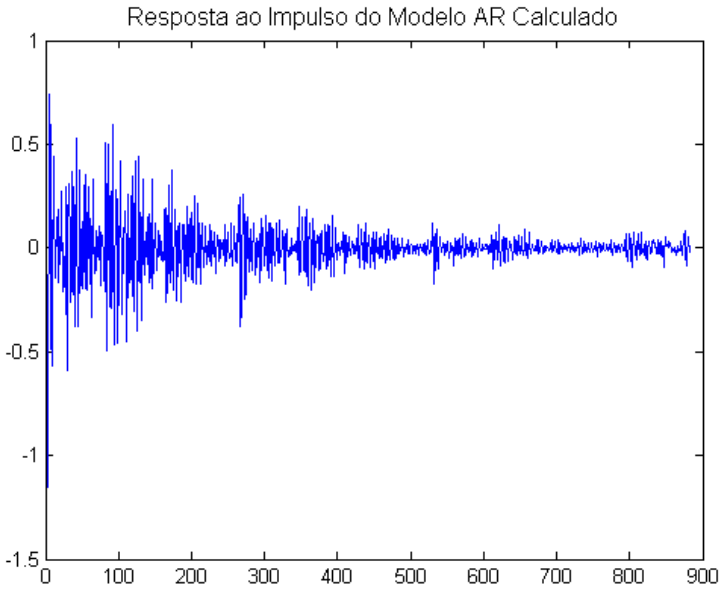

Figura 11: Resposta ao impulso do modelo da sala estimado pelo método Yule-Walker (aplicação do filtro).

Nota-se que o filtro eliminou não só o eco, mas também partes do sinal. Isso ocorre devido à já comentada imprecisão do modelo AR em descrever o efeito da sala, pois só possui informação do sinal y e nenhuma informação sobre o sinal x. O método de Yule-Walker com muitos coeficientes também fez com que a estimativa dos coeficientes fosse ruim, e isso causou uma descrição imprecisa do efeito da sala.

A aplicação do filtro FIR no sinal completo, sem decimação, também se torna impossível, uma vez que este sinal tem uma frequência de amostragem 50 vezes maior. Algumas técnicas podem ser exploradas para tentar aproximar os coeficientes de um filtro para uma amostragem maior com base apenas nestes 882 coeficientes, no entanto abordagens como essa fogem ao escopo do trabalho e podem ser pesquisadas posteriormente.

Apenas para fins de uma implementação correta, podemos utilizar o modelo AR indicado no experimento anterior, indicado na equação 4.1. Este modelo foi exportado nos arquivos *h1.out* e *h2.out* para ser utilizado na implementação em C.

4.2.2 Implementação em C

Como já previsto nas seções anteriores, reduzir a quantidade de coeficientes do filtro FIR através da técnica da decimação faz com que o filtro só funcione corretamente para os próprios sinais decimados. Assim, o filtro aplicado não foi estimado pelo método de Yule-Walker, mas sim, previsto pela equação 4.1, uma vez que já se conhece o eco que

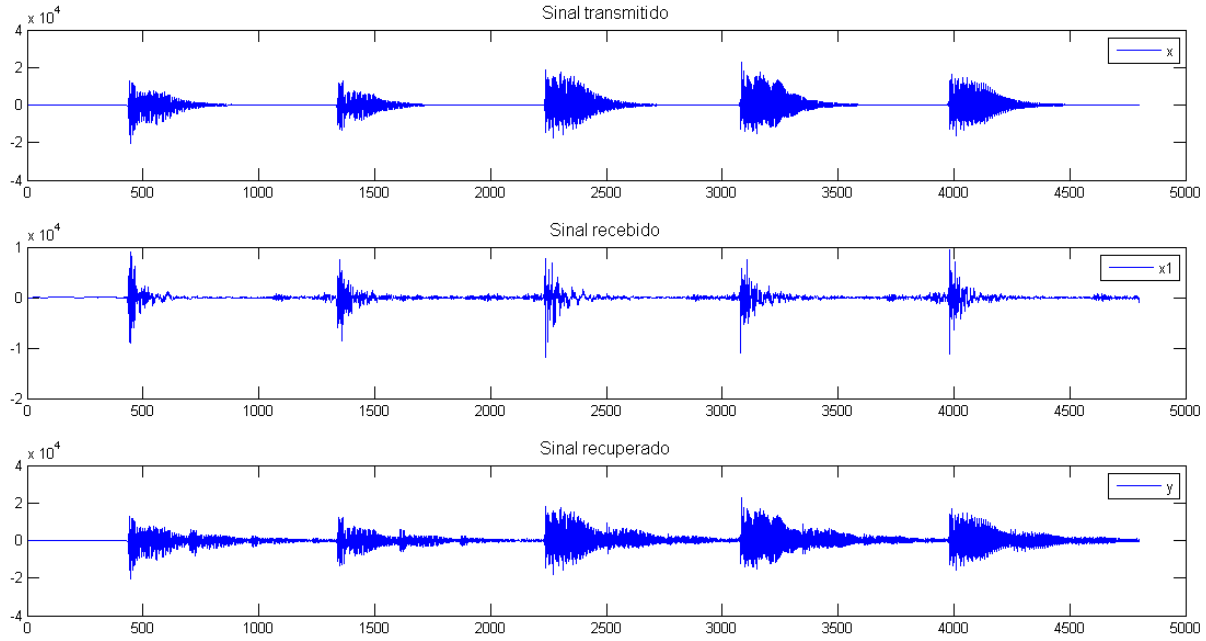

Figura 12: Comparação entre os sinais x (transmitido), y (com o efeito da sala) e $x1$ (recuperado). Nota-se que o filtro inverso anulou não só o eco, mas também partes do sinal, uma vez que não se baseou no sinal de entrada x, e sim, em uma aproximação arbitrária obtida pelo método de Yule-Walker. (aplicação do filtro)

existe no sinal.

O programa, que está indicado no Apêndice A, foi compilado e executado corretamente em uma plataforma Linux, distribuição Fedora 15 em um PC com arquitetura x86, e pode-se perceber que o eco foi removido do arquivo de som.

No entanto, na tentativa de se portar o programa para a placa BeagleBoard, houve problemas em relação ao *driver* de áudio fornecido para a placa. Uma mensagem de erro mostrou que o *driver* não era capaz de suportar um tamanho de quadro muito grande para receber e enviar dados para o dispositivo de áudio (que, no caso do código, era de tamanho 50000). A tentativa de reduzir para 13250 quadros, uma quantidade menor e que seria suficiente para anular o efeito de eco também não foi aceita pelo *hardware*. Desta forma, a melhor maneira de aplicar o filtro no ambiente embarcado seria desenvolver algoritmos que possam lidar com este gargalo do *frame buffer* encontrado no *driver* de áudio da placa em questão.

5 Conclusão

O presente trabalho contou com muitas análises de métodos e técnicas utilizadas em processamento digitais de sinais aplicado a equalização sonora, e conforme demonstrado, muitas abordagens se mostraram inadequadas para os objetivos propostos, enquanto outras não só se mostraram adequadas como também serviram para indicar os próximos passos a serem tomados em trabalhos futuros.

Exemplos de métodos descartados foram todos aqueles relacionadas à analise de sinais no domínio da frequência. Conforme já comentado na seção 2.1, estimar uma resposta ao impulso com base na divisão entre a saída e a entrada é uma alternativa inviável, dada a possibilidade de haver divisão por zero ou valores muito próximos de zero, ocasionando *overflow*. Outro método foi descartado durante o projeto do filtro FIR, onde pode-se passar a função de transferência da função para o domínio da frequência, $e^{j\omega}$, e aplicar o filtro nos coeficientes de frequência. Esta última abordagem, embora correta, é pouco eficiente dadas as otimizações dos DSPs relacionadas ao domínio do tempo, de forma que operações neste domínio ocorrem de maneira muito mais rápida.

Uma abordagem importante explorada durante o projeto se refere à utilização de sinais para o teste do efeito da sala. Uma entrada de impulso unitário, no sistema caracterizado pela sala e o filtro inverso, se mostrou muito mais útil para a análise dos sinais do que qualquer outro sinal de entrada. Sinais senoidais, por exemplo, são mais adequados para projetos no domínio analógico, e não digital.

Entre os métodos utilizados que podem ser explorados em trabalhos futuros, temos a análise conjunta dos modelos MA e AR, que mostrou que o primeiro é adequado para uma modelagem precisa de $h(n)$ em intervalos de tempo determinados, sendo uma resposta FIR, enquanto o segundo é adequado para a modelagem do eco introduzido por uma sala, sendo uma resposta IIR. Surge, então, a possibilidade de se unir estas duas características no modelo chamado ARMA - *Auto-regressive Moving Average*, um trabalho que pode ser desenvolvido posteriormente. A figura 13 apresenta um efeito de reverberação completo

causado por uma sala em um sinal de impulso unitário (o efeito é aplicado pela ferramenta de edição de áudio [Audacity 2012]).

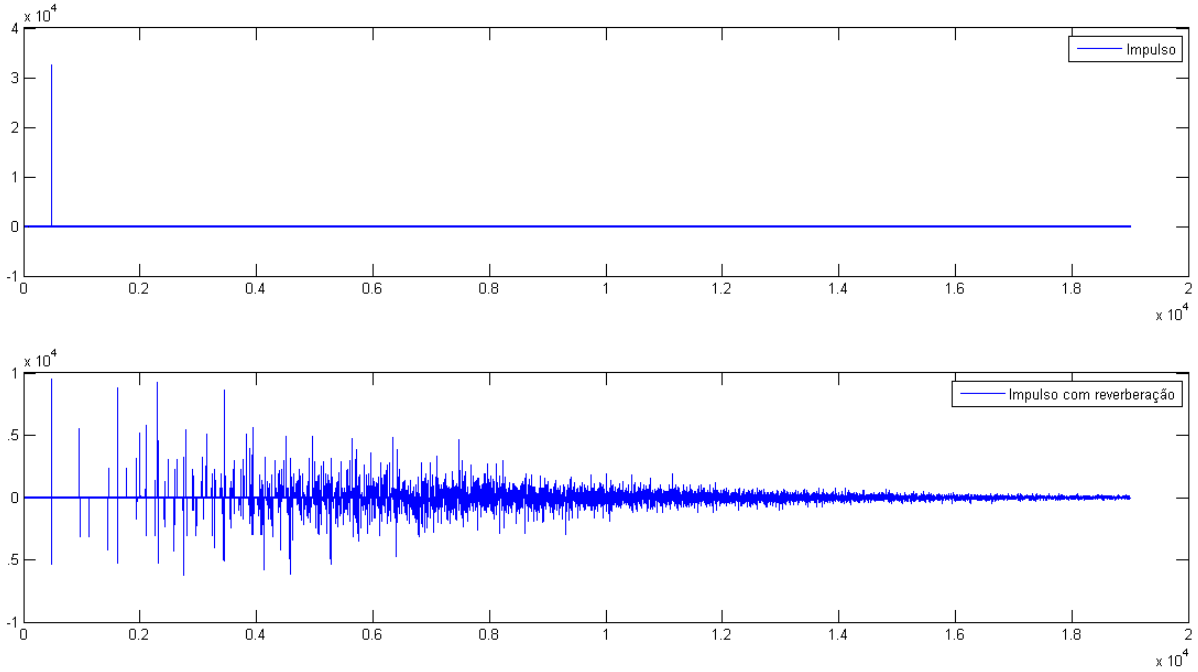

Figura 13: Efeito de reverberação completo de uma sala, aplicado em um sinal de impulso unitário na entrada.

Como pode-se observar na resposta ao impulso com reverberação, há um primeiro trecho, curto, que pode ser descrito com bastante precisão através de um modelo MA. Em seguida, devido a várias reflexões e interferência entre os sinais refletidos, a resposta começa a ficar mais indeterminada e é atenuada com o tempo, o que exige um modelo AR. Algumas pesquisas sugerem que o modelo de reverberação de uma sala, portanto, deve ser do tipo ARMA, com as características iniciais da resposta sendo tratadas por MA, e o eco gerado posteriormente por AR. Um filtro inverso a este modelo seria capaz de anular o efeito com bastante precisão.

Outro possível trabalho que pode ser desenvolvido em continuação a este consistiria na otimização do filtro FIR e sua aplicação em C. Pode-se buscar algumas técnicas capazes de reduzir o número dos coeficientes do filtro FIR, e mesmo técnicas que tornam o cálculo dos coeficientes do filtro FIR em tempo real, e na linguagem C, criando um sistema adaptativo. A alternativa de decimação no tempo para a redução dos coeficientes foi tentada durante o projeto e se mostrou insatisfatória, uma vez que distorceu várias frequências do sinal sonoro.

Considerando-se, portanto, os resultados obtidos e as presentes propostas de trabalhos futuros, temos ainda o fato de que o uso de DSPs e plataformas de Linux embarcado

vem crescendo muito rapidamente, tornando importantes as pesquisas nesta área. O crescimento de aplicações multimídia, principalmente em computação móvel e telecomunicações, tem constantemente exigido o uso de DSPs para processamento e equalização sonora, de forma a fornecer um conteúdo de qualidade usuários, tanto em quesitos de clareza no som como em possibilidade de interatividade (como em *software* de edição de áudio para músicos amadores). E, embora aplicações do tipo já existam em grande quantidade para plataformas de PCs convencionais, a possibilidade de portá-las para aplicações embarcadas abre um novo paradigma para usuários e desenvolvedores, apresentando possibilidades de aplicações relacionadas a processamento de sinais em tempo real, em qualquer lugar e em qualquer situação. E, como o presente trabalho demonstra, as técnicas de equalização sonora, os sistemas Linux embarcados e os compiladores com otimizações para arquiteturas de DSP tem desempenhado um papel importante nestes avanços.

Referências

[ALSA Project 2012]ALSA Project. *The ALSA Project Homepage*. May 2012. URL: http://www.alsa-project.org.

[Audacity 2012]Audacity. *Audacity*. June 2012. URL: http://audacity.sourceforge.net/.

[Bourke 1998]BOURKE, P. *Autoregression Analysis*. Nov 1998. URL: http://paulbourke.net/miscellaneous/ar/.

[Coley 2012]COLEY, G. *BeagleBoard System Reference Manual*. [S.l.], June 2012.

[Eaton 2009]EATON, K. *AIFFREAD, Matlab Central - File Exchange*. Mar 2009. URL: http://www.mathworks.com/matlabcentral/fileexchange/23328-aiffread.

[Hallinan 2010]HALLINAN, C. *Embedded Linux Primer: A Practical Real-World Approach*. [S.l.]: Prentice Hall, 2010.

[Hayes 96]HAYES, M. H. *Statistical Digital Signal Processing and Modeling*. [S.l.]: John Wiley and Sons, 96.

[Haykin e Widrow 2003]HAYKIN, S.; WIDROW, B. *Least-Mean Square Adaptative Filters*. [S.l.]: John Wiley and Sons, 2003.

[Joaquim 2006]JOAQUIM, M. B. *Processamento Digital de Sinais*. 2006.

[Oppenheim e Schafer 1998]OPPENHEIM, A. V.; SCHAFER, R. W. *Discrete-Time Signal Processing*. Second edition. [S.l.]: Prentice Hall, 1998.

[Paatero e Karjalainen 2002]PAATERO, T.; KARJALAINEN, M. New digital filter techniques for room response modeling. In: *Audio Engineering Society Conference*. [S.l.: s.n.], 2002.

[Takalo, Hytti e Ihalainen 2005]TAKALO, R.; HYTTI, H.; IHALAINEN, H. Tutorial on univariate autoregressive spectral analysis. *Journal of Clinical Monitoring and Computing*, 2005.

[Texas Instruments 2012]Texas Instruments. *C6EZRun*. June 2012. URL: http://processors.wiki.ti.com/index.php/C6EZRun/.

[Texas Instruments 2012]Texas Instruments. *OMAP Mobile Processors*. June 2012. Http://www.ti.com/general/docs/gencontent.tsp?contentId=46946/.

[The Ångström Distribution 2012]The Ångström Distribution. *The Ångström Distribution*. June 2012. Http://www.angstrom-distribution.org.

[Tranter 2004]TRANTER, J. *Introduction to Sound Programming with ALSA*. Set 2004. URL: http://www.linuxjournal.com/article/6735.

[Yin 2011]YIN, P. *Introduction to TMS320C6000 DSP Optimization*. [S.l.], 2011.

APÊNDICE A – Aplicação de um Filtro FIR em C

A seguir encontra-se o código desenvolvido em C que aplica um filtro FIR cujos coeficientes estão nos arquivos *h1.out* e *h2.out*, gerados na seção 3.2.1 do presente trabalho.

```
1  #define ALSA_PCM_NEW_HW_PARAMS_API
2
3  #include <alsa/asoundlib.h>
4  #include <math.h>
5
6  #define DEBUG 1
7  #define PLAY_RESULTS 1
8
9  #define FRAME_SIZE 50000
10
11 #define FIR_SIZE 44100
12
13 double* convolution(double *x, double *h, int n, int degree) {
14     double *y;
15     int i,k;
16
17     y = malloc(sizeof(double) * n);
18
19     double aux;
20     for (i=0;i<n;i++) {
21         aux = 0;
22         for (k=0;k<degree;k++) {
23             if((i-k)>=0) {
24                 aux += h[k]*x[i-k];
25             }
```

```
26          }
27          y[i] = aux;
28      }
29
30      return y;
31  }
32
33  int main() {
34    long loops;
35    int rc;
36    int size, window_size, i;
37    snd_pcm_t *handle;
38    snd_pcm_hw_params_t *params;
39    unsigned int val;
40    int dir;
41    snd_pcm_uframes_t frames;
42    char *buffer;
43    int play = PLAY_RESULTS;
44
45    char *out_buffer;
46    double *in_rev1, *in_rev2;
47    double *h1, *h2;
48    double *y1, *y2;
49
50    /* open files */
51    int writefd;
52    FILE *hf1, *hf2;
53    int fd1;
54    writefd = creat("out.raw",0);
55    fd1 = open("virtual_echo21.raw",0);
56
57    hf1 = fopen("h1.out","r");
58    hf2 = fopen("h2.out","r");
59
60    /* Open PCM device for playback. */
61    rc = snd_pcm_open(&handle, "default", SND_PCM_STREAM_PLAYBACK, 0);
62    if (rc < 0) {
```

```
63      fprintf(stderr,"unable to open pcm device: %s\n",snd_strerror(rc)
            );
64      exit(1);
65    }
66    /* Allocate a hardware parameters object. */
67    snd_pcm_hw_params_alloca(&params);
68    /* Fill it in with default values. */
69    snd_pcm_hw_params_any(handle, params);
70    /* Set the desired hardware parameters. */
71    /* Interleaved mode */
72    snd_pcm_hw_params_set_access(handle, params,
          SND_PCM_ACCESS_RW_INTERLEAVED);
73    /* Signed 16-bit little-endian format */
74    snd_pcm_hw_params_set_format(handle, params, SND_PCM_FORMAT_S16_LE)
          ;
75    /* Two channels (stereo) */
76    snd_pcm_hw_params_set_channels(handle, params, 2);
77    /* 44100 bits/second sampling rate (CD quality) */
78    val = 44100;
79    snd_pcm_hw_params_set_rate_near(handle, params, &val, &dir);
80    /* Set period size to 32 frames. */
81    frames = FRAME_SIZE;
82    snd_pcm_hw_params_set_period_size_near(handle, params, &frames, &
          dir);
83    /* Write the parameters to the driver */
84    rc = snd_pcm_hw_params(handle, params);
85    if (rc < 0) {
86      fprintf(stderr,"unable to set hw parameters: %s\n",snd_strerror(
            rc));
87      exit(1);
88    }
89    /* Use a buffer large enough to hold one period */
90    snd_pcm_hw_params_get_period_size(params, &frames, &dir);
91    size = frames * 4; /* 2 bytes/sample, 2 channels */
92     buffer = (char *) malloc(size);
93    /* set the window size */
94    window_size = frames;
95    /* We want to loop for 5 seconds */
```

```
96  snd_pcm_hw_params_get_period_time(params, &val, &dir);
97  /* 5 seconds in microseconds divided by period time */
98  loops = 10000000 / val;
99
100 in_rev1 = malloc(sizeof(double)*window_size);
101 in_rev2 = malloc(sizeof(double)*window_size);
102 out_buffer = (char *) malloc(size);
103 h1 = malloc(sizeof(double)*FIR_SIZE);
104 h2 = malloc(sizeof(double)*FIR_SIZE);
105
106 // reads the FIR coefficients
107 for (i=0;i<FIR_SIZE;i++) {
108   fscanf(h1,"%lf",&h1[i]);
109   fscanf(h2,"%lf",&h2[i]);
110 }
111
112 while (loops > 0) {
113
114   loops--;
115
116   /* read buffer */
117   if(rd_input)
118       rc = read(0, buffer, size);
119   else
120       rc = read(fd1, buffer, size);
121
122   if (rc == 0) {
123       fprintf(stderr, "end of file on input\n");
124       break;
125   } else if (rc != size) {
126       fprintf(stderr, "short read: read %d bytes\n", rc);
127   }
128
129   // load samples
130   for (i=0;i<window_size;i++) {
131       // channel 1
132       in_rev1[i] = (short) (buffer[4*i+1] << 8 | buffer[4*i]);
133       // channel 2
```

```
134             in_rev2[i] = (short) (buffer[4*i+3] << 8 | buffer[4*i+2]);
135         }
136
137         y1 = convolution(in_rev1,h1,window_size,FIR_SIZE);
138         y2 = convolution(in_rev2,h2,window_size,FIR_SIZE);
139
140         // load channels into out_buffer
141         for (i=0;i<window_size;i++) {
142             // little endian
143             short a, b;
144             a = y1[i];
145             b = y2[i];
146             out_buffer[4*i] =   (char) a;
147             out_buffer[4*i+1] = (char) (a >> 8);
148             out_buffer[4*i+2] = (char) b;
149             out_buffer[4*i+3] = (char) (b >> 8);
150         }
151
152         /* write into sound buffer */
153         if (play) {
154             rc = snd_pcm_writei(handle, out_buffer, frames);
155             if (rc == -EPIPE) {
156                 fprintf(stderr, "underrun occurred\n");
157                 snd_pcm_prepare(handle);
158             } else if (rc < 0) {
159                 fprintf(stderr,"error from writei: %s\n",snd_strerror(rc)
                        );
160             } else if (rc != (int)frames) {
161                 fprintf(stderr,"short write, write %d frames\n", rc);
162             }
163         }
164         else {
165             rc = write(writefd, out_buffer, size);
166         }
167     }
168
169     free(buffer); free(in_rev1); free(in_rev2); free(out_buffer); free(
        h1); free(h2);
```

```
170     snd_pcm_drain(handle);
171     snd_pcm_close(handle);
172     close(writefd);
173     return 0;
174 }
```