UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

ESCOLA POLITÉCNICA

DEPARTAMENTO DE ELETRÔNICA E DE COMPUTAÇÃO

CODIFICADOR DE VOZ CELPS: VERSÃO ESTRUTURADA E DETECÇÃO DE SILÊNCIO.

Autor: _____
Alexandre Guazzi Gomes

Orientador: _____
Prof. Sérgio Lima Netto, Ph.D.

Co-orientador: _____
Vagner Luis Latsch, M.Sc.

Examinador: _____
Prof. Jose Gabriel Rodriguez Carneiro Gomes, Ph.D.

Examinador: _____
Prof. Amaro Azevedo de Lima, Ph.D.

DEL
Rio de Janeiro, RJ – Brasil
Dezembro de 2008

À minha mãe, Rosângela Guazzi...

Agradecimentos

Meus sinceros agradecimentos:

- ao professor e orientador Sergio Lima Netto, por toda a ajuda e pela excelente orientação que recebi ao longo deste projeto final;

- ao meu co-orientador Vagner Luis Latsch, que também me ajudou com orientações sobre várias questões relativas ao projeto e sempre esteve disposto a ajudar;

- aos meus amigos e professores, que direta ou indiretamente me ajudaram com apoio e orientações durante toda a graduação;

- e à minha mãe, por toda ajuda e incentivo recebido.

Alexandre Guazzi Gomes

Resumo

O presente trabalho possui dois objetivos principais: a criação de uma biblioteca do codificador CELPS com o código reestruturado na Linguagem C++ que reúna, em um bloco único e auto-contido, todas as funcionalidades principais do codificador e o estudo e implementação de modificações com o intuito de melhorar a relação qualidade por banda do codificador. Além disso, este trabalho explica o funcionamento do codificador CELPS descrevendo cada etapa da codificação e decodificação e documenta o histórico do codificador descrevendo todas as versões anteriores e ressaltando as modificações implementadas por cada uma.

Para a otimização do codificador CELPS foram feitos estudos sobre a influência da resposta à entrada zero, gerada por um bloco de sinal de voz nos seus blocos vizinhos, e a influência de diferentes combinações de tamanhos dos dicionários fixo e adaptativo na qualidade da codificação e na taxa gerada. Além disso, foram feitos estudos sobre a detecção de silêncios nos sinais de voz. Estudou-se a influência dos parâmetros de detecção de silêncio, energia limite e taxa de cruzamentos por zero na qualidade da codificação e taxa gerada; foi implementada a técnica de reconstrução dos blocos do tipo silêncio através do envio dos coeficientes LSF e da geração de ruído como excitação e, por fim, foi feita a requantização separada dos coeficientes LSF para sinais ativos ou inativos (silêncios).

Índice

Capítulo 1

Introdução

Com os avanços da eletrônica e das tecnologias de telecomunicações tornaram-se realizáveis muitas invenções antes idealizadas apenas em ficção científica. O telefone fixo e o celular foram uma delas. A tecnologia digital permitiu que o número de linhas telefônicas e o de ligações simultâneas se multiplicassem. Inicialmente, com a tecnologia analógica, os aparelhos celulares eram caros, grandes e muito pouco eficientes em termos de consumo de energia, utilização de banda e qualidade das ligações. Com a implementação das tecnologias digitais, todos esses fatores melhoraram muito. Mas as melhorias não se restringiram apenas a estes fatores, outras novas funcionalidades também puderam ser agregadas. Isto se deve ao fato de que a tecnologia digital permite que se utilizem algoritmos matemáticos complexos para a compressão dos sinais de voz, diminuindo ainda mais o espaço ocupado pela voz num canal de comunicação, ou seja, diminuindo sua taxa de bits por segundo. Porém, estes algoritmos não necessariamente precisam ser complexos. A simples amostragem e quantização do sinal (técnica PCM), por exemplo, já permite que os canais de voz sejam multiplexados no tempo, aumentando assim o número de conversações por canal.

No entanto, existem técnicas mais complexas de codificação de voz que a simples amostragem e quantização dos sinais são estas as técnicas de compressão dos sinais de voz. Estas técnicas fazem o uso não somente das características temporais dos sinais, mas também das suas características espectrais. Estas técnicas, em geral, comprimem o sinal com perdas, ou seja, o sinal quando decodificado terá uma qualidade inferior ao sinal original. Quanto melhores forem as técnicas de compressão, melhor será a qualidade dos sinais decodificados e menor será a taxa de bits geradas para a representação dos mesmos. Desta forma, o codificador que consegue a menor taxa de codificação para uma mesma qualidade do sinal reproduzido é considerado o mais eficiente.

As técnicas de codificação de voz mais complexas se baseiam não somente na amostragem e quantização dos sinais, mas sim, na utilização de filtros para modelar outras características específicas do sinal de voz como por exemplo a sua distribuição espectral.

Estas técnicas permitem a compressão dos sinais de voz para taxas muito baixas quando comparadas aos sinais simplesmente amostrados e quantizados. Os codificadores de voz que mais se destacam são os baseados na técnica CELP (*Code Excited Linear Prediction*). Esta técnica faz uso de regressões lineares e dicionários que contém excitações, as quais são utilizadas para reconstrução do sinal de voz. Esse tipo de codificador apresenta boa qualidade e baixa taxa de codificação, o que faz com que seja largamente utilizado em telefonia digital.

O codificador CELPS (*Code Excited Linear Prediction* - LPS) é um codificador baseado na técnica CELP desenvolvido por professores e alunos do LPS (Laboratório de Processamento de Sinais) do departamento de Engenharia Eletrônica da UFRJ. Este codificador é objeto de estudo do presente trabalho.

1.1 Proposta do trabalho

Este projeto final tem como um de suas propostas a reestruturação do código do codificador CELPS desenvolvido em [5] na forma orientada a objeto. O objetivo é o de se reunir todas as suas funcionalidades principais em uma biblioteca que contenha somente a implementação das funções do codificador, desta forma, isolando em um bloco único e auto-contido, toda a parte de codificação e decodificação.

Além da criação da biblioteca CELPS, este trabalho realiza estudos sobre a influência da resposta a entrada zero gerada por um bloco de sinal de voz nos seus blocos vizinhos; a influência dos tamanhos dos dicionários de excitações na qualidade do sinal decodificado e sobre a influência dos parâmetros de energia limite e taxa de cruzamentos por zero na detecção de silêncio. Além desses estudos, foram feitas a separação e requantização dos coeficientes LSF dos sinais do tipo sonoro e silêncio de forma separada. Todos esses estudos e modificações foram realizados com o objetivo de entender melhor o comportamento do tipo de codificação empregada no CELPS e também da melhoria de sua razão qualidade por taxa.

Outro objetivo do presente trabalho é explicar o funcionamento do codificador CELPS descrevendo cada etapa da codificação e decodificação bem como as características dos métodos utilizados. Também é feito um histórico de todas as versões do codificador CELPS desde a sua primeira implementação visando a documentação de todas as melhorias já experimentadas e todas adaptações para diferentes interfaces já implementadas.

Com a criação de uma biblioteca em linguagem orientada a objeto contendo todas as melhorias já testadas anteriormente, com a documentação do histórico do codificador e com a descrição do seu funcionamento, este trabalho visa reunir em uma única documentação suas principais características e também a trajetória do codificador CELPS de forma a facilitar a continuação dos estudos do codificador por trabalhos futuros.

1.2 Organização da dissertação

O capítulo 2 fornece uma idéia geral do campo de codificadores de voz. São explicado os tipos de codificadores de voz mais comuns dando o enfoque ao codificador CELPS. São explicadas as técnicas utilizadas no CELPS bem como os métodos de qualificação dos sinais de voz mais comuns. Ao final é feito um resumo das etapas do funcionamento da codificação e decodificação do codificador CELPS.

No capítulo 3 é feito um resumo de todas as versões do codificador CELPS já implementadas. Nele é possível observar a trajetória do desenvolvimento do codificador e analisar todas as melhorias já propostas e testadas bem como todas as interfaces de uso já feitas. A leitura deste capítulo é importante para se entender a evolução do codificador CELPS e motivar as modificações propostas pelo presente trabalho.

No capítulo 4 é descrita a implementação da versão do codificador CELPS proposta pelo presente trabalho. Nele são mostradas todas as modificações propostas relativas à reestruturação do código e bem como todas as decisões de projeto. Este capítulo mostra todas as classes, métodos e funções explicando resumidamente a função de cada uma.

No capítulo 5 são mostrados os estudos e modificações feitas sobre o codificador CELPS. São explicados seus objetivos, suas implementações e os resultados gerados.

No capítulo 6 é feito um resumo do projeto ressaltando os principais resultados obtidos e analisando seus impactos no codificador. São descritas também as propostas para trabalhos futuros.

Capítulo 2

Codificadores de Voz

2.1 Introdução

Este capítulo tem como objetivo a apresentação dos tipos de codificadores voz tendo como enfoque principal o codificador CELPS desenvolvido no LPS (Laboratório de processamento de sinais) da UFRJ.

Na seção 2.2, são descritos os principais métodos e ferramentas utilizados para a avaliação da qualidade dos codificadores neste trabalho. Em particular, são descritos o avaliador de qualidade PESQ (*Perceptual Evaluation of Speech Quality*), a escala de classificação de qualidade MOS (*Mean Opinion Score*) e os bancos de voz para testes.

Na seção 2.3 são descritos os diferentes tipos de codificadores de voz abrangendo os codificadores de forma de onda, os codificadores paramétricos e os codificadores híbridos. São mostrados também exemplos de cada tipo de codificador ressaltando as vantagens e desvantagens de cada um e comparando seus desempenhos.

Na seção 2.4 é descrito o codificador híbrido CELPS resumindo-se suas principais características e seu funcionamento passo a passo.

2.2 Métodos de qualificação de codificadores de voz

Para que se possa estudar os tipos de codificadores e compará-los entre si é importante que se entenda primeiramente como é feita a avaliação dos mesmos. Nas subseções seguintes são mostradas a ferramenta de avaliação e a escala comumente utilizada.

2.2.1 MOS (*Mean Opinion Score*)

O MOS é descrito na recomendação P.800.1 [12] da ITU (*International Telecommunication Union*) que descreve métodos e procedimentos que permitem a avaliação subjetiva da qualidade de sinais de voz. O objetivo do MOS é classificar um sinal de voz quanto a sua qualidade. São atribuídas notas de 1 a 5, segundo a tabela 2.1.

Tabela 2.1: Escala MOS.

MOS	Qualidade do sinal de voz
5	Excelente
4	Bom
3	Regular
2	Ruim
1	Pobre

O MOS é calculado por um grupo de pessoas que comparam os sinais originais com os que foram codificados e então atribuem uma nota de 1 a 5 segundo a percepção de qualidade de cada pessoa. Este método, no entanto, é bastante custoso e demorado. Assim, foram desenvolvidos programas que tentam automatizar essa avaliação para que possa ser feita de forma objetiva. Isto reduz consideravelmente o tempo dedicado aos testes de novas melhorias nos algoritmos dos codificadores.

2.2.2 PESQ (*Perceptual Evaluation of Speech Quality*)

O PESQ é uma medida de qualidade de sinais de voz pertencente à recomendação P.862 da ITU [13]. É utilizado para estimar de forma automática a nota MOS de sinais de voz codificados. Seus resultados possuem uma precisão aceitável de cerca de 90% de correlação estatística com o MOS. É levada em consideração a clareza da voz quando afetada pelos seguintes processos ou parâmetros:

- Codificadores de forma de onda;
- Codificadores paramétricos e híbridos a partir de 4 kbps;
- Erros no canal de transmissão;
- Perdas de pacotes.

Para calcular o valor PESQ, compara-se o sinal de entrada com o sinal de saída do codificador, ambos no formato de arquivo wav. Os arquivos são passados ao programa quando chamado por linha de comando.

O resultado objetivo obtido pelo cálculo PESQ pode ser mapeado na avaliação subjetiva MOS através da seguinte equação [1]:

$$MOS = 0{,}999 + \frac{4}{1+e^{(-1.4945*PESQ+4{,}6607)}}.$$

2.2.3 Bancos de voz

O banco de voz é um conjunto de pequenos arquivos de voz que são utilizados para testar a qualidade do codificador. Os arquivos possuem uma média de 4 segundos de duração, e contém frases em diferentes línguas faladas por homens e mulheres. A tabela 2.2 resume as informações do banco de voz. A obtenção deste banco de voz está descrita em [7].

Tabela 2.2: Banco de voz.

Idioma	Legenda	Sexo	Número de frases
Chinês	CH	Masculino	20
Francês	FR	Masculino	104
Indiano	IN	Masculino	80
Inglês Britânico	UK	Masculino	152
Inglês EUA	US	Masculino	100
Inglês EUA	US	Feminino	140
Total:			596

Para que se possa obter um resultado abrangente e mais próximo da realidade, para cada modificação implementada no codificador CELPS todas as frases do banco de voz são submetidas à codificação, decodificação e têm seus valores MOS calculados pelo PESQ. Em seguida, é feita uma média dos valores MOS entre todas as 596 frases gerando assim o valor MOS final decorrente da modificação em teste.

2.3 Tipos de codificadores

Existem diferentes tipos de codificadores que fazem uso de diferentes características do sinal de voz. Estes codificadores podem ser classificados em três tipos: codificadores por forma de onda, codificadores paramétricos e codificadores híbridos. Estes codificadores são descritos nas subseções seguintes.

2.3.1 Codificadores por forma de onda

Os codificadores por forma de onda fazem uso das características temporais e espectrais do sinal de voz. As etapas da codificação são resumidas a seguir:

- Filtragem do sinal: o sinal de voz é filtrado para que seja limitado em banda, em geral, limitado à banda de voz telefônica 300 a 3400 Hz;
- Amostragem do sinal: o sinal é amostrado respeitando o limite da freqüência de Nyquist, no caso da voz em telefonia a freqüência utilizada é 8kHz.
- Quantização: os valores das amostras do sinal são então aproximados para os níveis quânticos mais próximos.

As etapas mostradas acima são características dos codificadores por forma de onda mais conhecidos:

- G.711 (PCM) 8000 Hz x 8 bits - 64 kbps - MOS 4,3
- G.721 (ADPCM) 8000 Hz x 4 bits - 32 kbps - MOS 3,8

A grande vantagem desses codificadores é a baixa complexidade computacional, uma vez que não é feita análise do sinal, e a alta qualidade que chega próxima à pontuação 4,5 na escala MOS no caso do G.711. No entanto, a alta taxa de bits é a principal desvantagem.

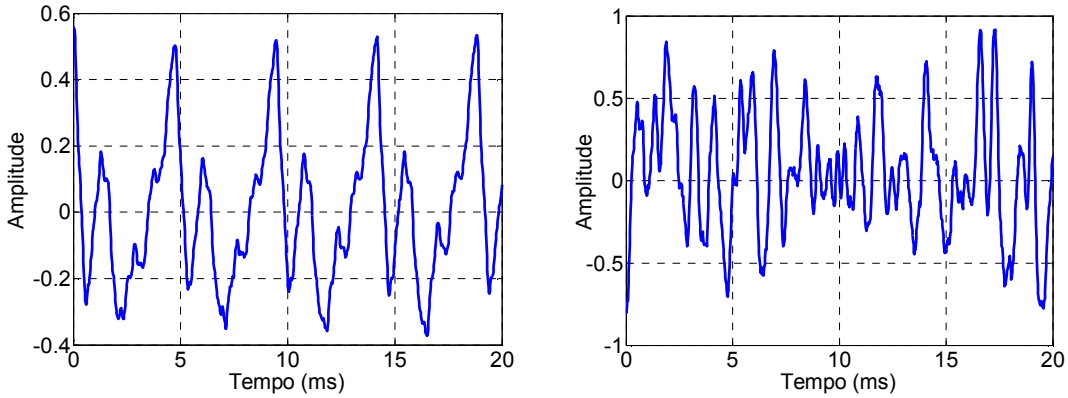
2.3.2 Codificadores paramétricos

Os codificadores paramétricos são codificadores específicos para cada tipo de sinal e procuram modelar matematicamente as características do trato vocal humano.

A voz humana pode ser modelada através do filtro de síntese cujas características são explicadas com mais detalhes na subseção 2.4.2. Resumidamente, o filtro de síntese é um filtro digital cuja entrada é uma excitação que modela o ar que sai dos pulmões e provoca a vibração das cordas vocais e cuja saída é a voz. Este filtro modela o trato vocal humano que é formado pela glote, epiglote, garganta, língua, dentes e lábios.

Os tipos de som mais comuns são os sonoros e os surdos. Os sons sonoros são aqueles que provocam excitação das cordas vocais. São sons semi-periódicos característicos da vibração das cordas vocais. Exemplos de sons sonoros são os sons das vogais e dos encontros vocálicos. A figura 2.1(a) mostra um exemplo de 20 ms de um sinal de voz sonoro da vogal “a”.

Já os sons surdos são sons de caráter ruidoso que não provocam vibração nas cordas vocais e, portanto, não possuem uma freqüência fundamental definida. Exemplos desses sons são os fonemas consonantais como “ch” em chuva, “s” em sorte, ou “f” em festa. A figura 2.1(b) mostra um exemplo de 20 ms de um sinal de voz surdo da consoante “f” da palavra foca.



(a) (b)

Figura 2.1: (a) Trecho de 20 ms de um sinal de voz sonoro da vogal “a”; (b) Trecho de 20 ms de um sinal de voz surdo da consoante “f” na palavra foca.

A voz humana é um sinal do tipo não estacionário e não periódico, no entanto, quando este sinal é dividido em pequenos blocos de duração, entre 10 ms e 30 ms , ele pode ser considerado estacionário e periódico por partes [15]. Devido a essas características, a subdivisão em blocos é utilizada para se modelar o filtro de síntese.

O LPC (*Linear Predictive Coding*) é o principal codificador paramétrico para a voz humana. Ele faz o uso dessas características de estacionaridade e periodicidade para modelar um filtro de síntese diferente para cada bloco de 10 ms a 30 ms de duração. O filtro gerado pela análise LPC é um filtro do tipo *all-pole* (figura 2.2), ou seja, só se varia a posição dos pólos, com os zeros permanecendo na origem do plano complexo. Os coeficientes desse filtro são determinados pela técnica de regressão linear. Descrita na subseção 2.4.3.

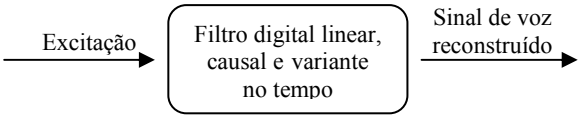


Figura 2.2: Processo de síntese da voz humana utilizado pelo codificador LPC

A principal vantagem dos codificadores paramétricos é que, ao invés de se ter que enviar informações de cada amostra do sinal, são enviados apenas alguns coeficientes. No

caso do codificador LPC, para cada bloco são enviados os coeficientes do filtro de síntese, o ganho do filtro e um *flag* (U/UV) que identifica se o som é do tipo sonoro ou surdo. Caso o som seja sonoro, também é enviado o valor do *pitch*, que é a percepção da freqüência fundamental ou periodicidade do sinal.

Para exemplificar a relação da taxa de transmissão entre o codificador LPC e o PCM considera-se um bloco de 20 ms do sinal de voz. Se este bloco for amostrado em 8 kHz são geradas 160 amostras. Utilizando o PCM seria necessário enviar todas as 160 amostras. Já no LPC seriam enviados, por exemplo, apenas 13 coeficientes (10 coeficientes LPC, o ganho, o *flag* U/UV e o valor do *pitch*). Portanto, o codificador LPC consegue reduzir consideravelmente a taxa de transmissão de 160 para 13 informações numéricas por bloco mas, no entanto, a qualidade do som reconstituído é baixa (aproximadamente 2,6 na escala MOS). A seguir a figura 2.3 mostra o diagrama em blocos das etapas da codificação LPC.

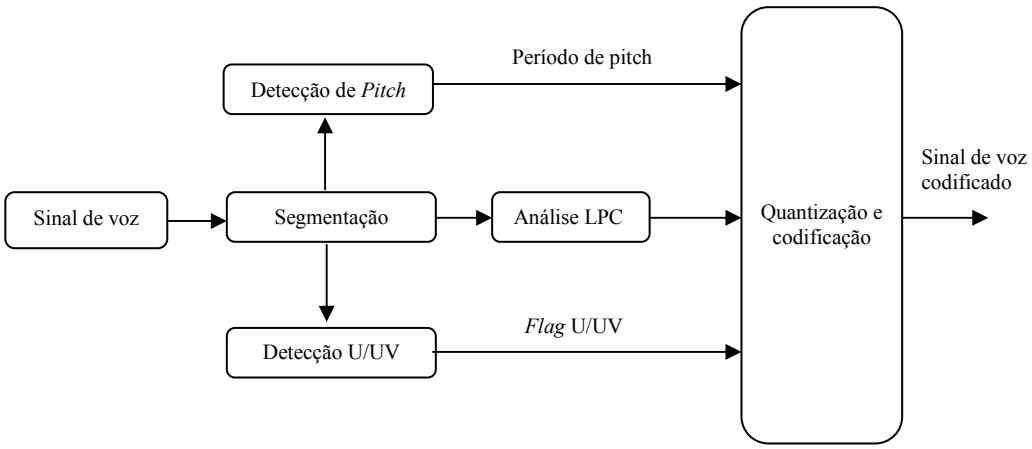


Figura 2.3: Esquema da codificação LPC

2.3.3 Codificadores híbridos

Os codificadores híbridos realizam a extração de parâmetros dos sinais de voz assim como os codificadores paramétricos, e ao mesmo tempo utilizam características temporais e espectrais dos sinais como os codificadores de forma de onda. Desta forma,

conseguem obter uma melhor qualidade de sinal reconstituído quando comparado aos outros tipos de codificadores, isso com taxas relativamente baixas, entre 2 kbps e 16 kbps.

Um dos tipos de codificação híbrida é a utilizada pelo codificador CELPS que é o objeto de estudo deste trabalho e é detalhado na seção 2.4.

2.3.4 Comparação entre os tipos de codificadores

Os codificadores de voz são, de modo geral, classificados por cinco fatores: qualidade, taxa de bits, complexidade computacional, atraso e robustez à perda de pacotes, sendo os dois primeiros os fatores principais. A complexidade computacional é um fator importante quando se deseja utilizar o codificador em dispositivos portáteis ou em outros dispositivos com limitação de bateria. Com o crescente aumento da velocidade dos computadores e DSPs (*Digital Signal Processors*) o que geralmente se procura fazer primeiro é melhorar a relação entre a qualidade e a taxa para depois tentar fazer aceleração do codificador diminuindo sua complexidade computacional. Com relação ao atraso, o máximo atraso em um sentido de transmissão, para usuários mais críticos e exigentes não deve ultrapassar 100 ms, enquanto que para usuários mais tolerantes, este valor pode chegar até a 400 ms [5]. Já a robustez à perda de pacotes é um fator crítico quando se deseja utilizar o codificador em sistemas onde seja comum a perda de pacotes, como é o caso da internet e da rede de telefonia celular. Para esses sistemas o codificador tem que ser capaz de impedir a propagação de erros quando acontece perda de pacotes. A figura 2.4 mostra a relação entre qualidade e taxa dos diferentes tipos de codificadores [16].

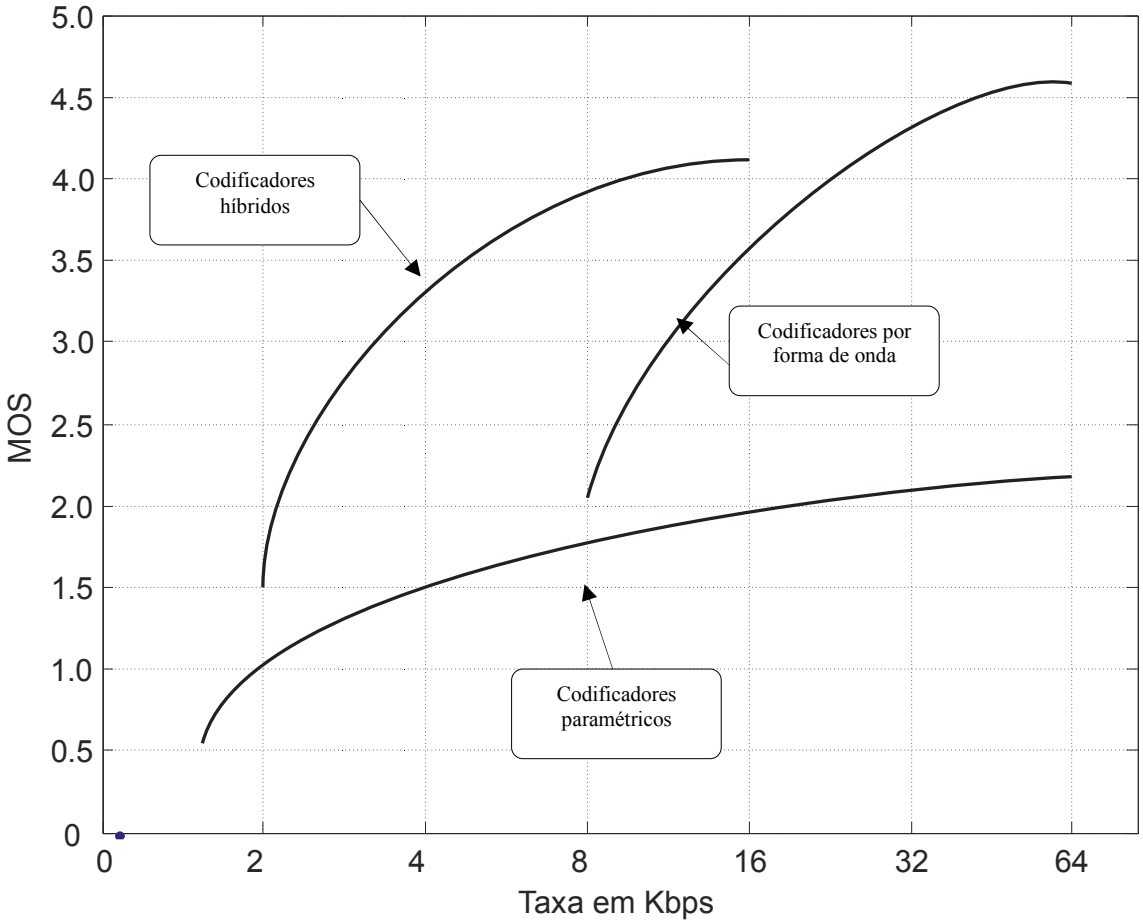


Figura 2.4: Relação entre qualidade e taxa dos tipos de codificadores

A partir do gráfico pode-se observar que os codificadores por forma de onda produzem uma qualidade muito boa, mas, a custo de taxas de bits muito altas. Em contrapartida, os codificadores paramétricos conseguem taxas de bits bem menores, mas, a custo de uma perda significativa na qualidade. Já os codificadores híbridos, uma vez que reúnem as características dos codificadores paramétricos e híbridos, conseguem produzir uma boa qualidade com taxas de bits pequenas, entre 2 e 16 kbps.

2.4 Codificador CELPS

O codificador CELPS (*Code Excited Linear Prediction* - LPS) desenvolvido por professores e alunos do LPS (Laboratório de Processamento de Sinais) do departamento de Engenharia Eletrônica da UFRJ é um codificador de voz do tipo híbrido que tem como base o codificador LPC. O CELPS utiliza dois dicionários de excitações para a entrada do

filtro de síntese visando diminuir o erro da reconstrução do sinal na saída do filtro. O CELPS, além do LPC, faz o uso de outros recursos para melhorar a qualidade da codificação e diminuir a taxa de bits. Esses recursos bem como o cálculo de alguns parâmetros estão listados abaixo e são descritos nas subseções seguintes:

- Janelamento do sinal de voz ;
- Filtro de síntese;
- Cálculo dos coeficientes LPC;
- Interpolação dos coeficientes LSF;
- Filtro perceptivo;
- Dicionários;
- Análise por síntese;

O Codificador CELPS se destaca do codificador LPC porque consegue gerar um sinal de voz reconstruído com qualidade consideravelmente superior à do LPC, no entanto, a um custo de uma taxa maior.

2.4.1 Janelamento do sinal de voz

Como dito anteriormente, o sinal de voz é um sinal do tipo não estacionário e não periódico. No entanto, quando considerado em blocos de pequenos intervalos entre 10 ms e 30 ms o sinal pode ser considerado estacionário dentro desses intervalos. Essa característica leva o codificador CELPS a ter que recortar o sinal em pedaços. O tamanho padrão dos blocos utilizado pelo CELPS é de 20 ms. No entanto, ao se recortar o sinal simplesmente copiando as amostras dentro do intervalo desejado pode acontecer de as amostras do início e/ou fim do intervalo não serem zero. Essas descontinuidades geradas pelas primeiras e últimas amostras serão modeladas no cálculo dos coeficientes LPC (ver seção 2.4.3), o que é indesejado. O janelamento, portanto, serve para atenuar o começo e o fim dos blocos a fim de reduzir tais descontinuidades e garantir que os coeficientes representarão de forma mais precisa o sinal original. A janela utilizada pelo CELPS é a janela Hamming, descrita por [9]:

$$w_h(n) = \begin{cases} \alpha + (1-\alpha)\cos\left(\frac{2\pi n}{M}\right), & |n| \leq \frac{M}{2} \\ 0, & |n| > \frac{M}{2} \end{cases},$$

onde $\alpha = 0{,}54$ [9] e M é a ordem do filtro.

2.4.2 Filtro de síntese

O filtro de síntese é um filtro tipo *all-pole* caracterizado por uma relação entrada e saída da forma:

$$H(z) = \frac{1}{A(z)} = \frac{1}{1 - \sum_{i=1}^{p} a_i z^{-i}},$$

onde $a_i = \{a_1, a_2, a_3, ..., a_p\}$ são os coeficientes de predição linear de um bloco de sinal.

O modelo LPC aproxima o espectro de freqüência do sinal. Portanto, quanto maior for o número de coeficientes melhor será a modelagem do sinal. As figuras 2.5(b), 2.5(c) e 2.5(d) mostram respectivamente os espectros de freqüências dos filtros modelados com p=3, p=10 e p=30 para 20 ms do sinal de voz da vogal "a" com 160 amostras cujo espectro é mostrado na figura 2.5(a). Nesta figura, observa-se que o aumento do número de coeficientes leva o filtro a conseguir acompanhar cada vez mais as variações rápidas nos módulos das freqüências. O codificador CELPS utiliza apenas 10 coeficientes o que já proporciona uma boa relação entre qualidade e taxa de transmissão.

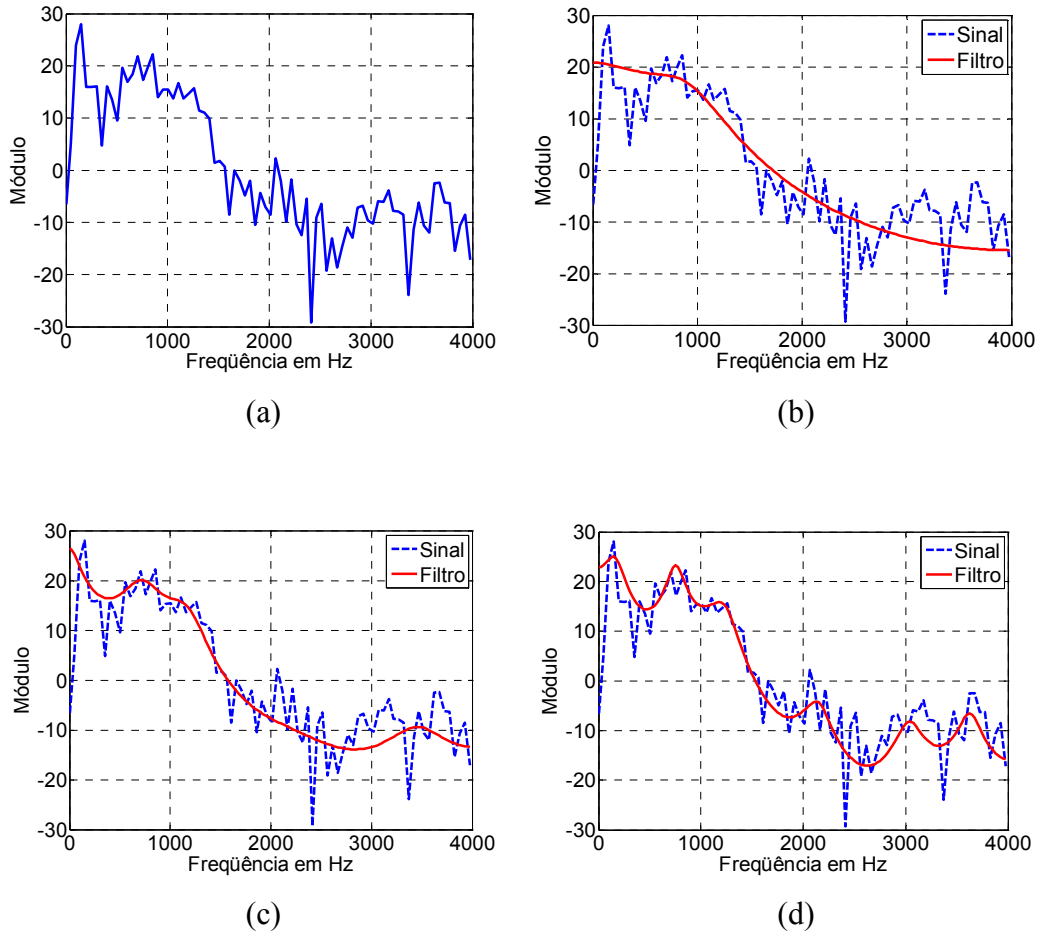


Figura 2.5: (a) Espectro do sinal de voz de 20 ms da vogal “a”; (b) Filtro de síntese de 3ª ordem (p=3); (c) Filtro de síntese de 10ª ordem (p=10); (d) Filtro de síntese de 15ª ordem (p=15).

O filtro de síntese, contudo, gera uma reposta de tamanho maior que a excitação de entrada. A parte do sinal da resposta do filtro que ultrapassa o tamanho da excitação de entrada são os resquícios de sinal cuja energia pode interferir nos sub-blocos seguintes. Estes resquícios são removidos do sinal original antes da codificação, pois, o mesmo resquício será gerado pelo filtro de síntese no decodificador e somado de volta ao sinal decodificado. Estes resquícios são conhecidos como resposta à entrada zero. No capítulo 5 é mostrado o estudo feito sobre a influência desses resquícios nos sub-blocos seguintes.

2.4.3 Cálculo dos coeficientes LPC

O cálculo dos coeficientes LPC que formam o filtro de síntese é composto das seguintes etapas:

1. Calcula-se a matriz de auto-correlação do bloco do sinal de voz:

$$R_s = \begin{bmatrix} r_s(0) & r_s(1) & r_s(2) & \cdots & r_s(M-1) \\ r_s(1) & r_s(0) & r_s(1) & \cdots & r_s(M-2) \\ r_s(2) & r_s(1) & r_s(0) & \cdots & r_s(M-3) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ r_s(M-1) & r_s(M-2) & r_s(M-3) & \cdots & r_s(0) \end{bmatrix},$$

onde

$$r_n(j) = \frac{1}{N}\sum_{n}^{N-1} x(n)x(n-j).$$

2. Resolve-se a equação matricial pelo método de Levinson-Durbin [14]:

$$R_s * \begin{bmatrix} a(1) \\ a(2) \\ a(3) \\ \vdots \\ a(M) \end{bmatrix} = \begin{bmatrix} r_s(1) \\ r_s(2) \\ r_s(3) \\ \vdots \\ r_s(M) \end{bmatrix},$$

onde $\{a(1), a(2), a(3), ..., a(M)\}$ são os coeficientes do filtro de síntese.

No entanto, a resposta em freqüência do filtro de síntese é muito sensível a erros de quantização dos coeficientes LPC. Um erro em um único coeficiente LPC, por exemplo, significa a mudança de todos os pólos da função de transferência correspondente. Para diminuir esse problema os coeficientes LPC são transformados em outros tipos de

coeficientes cuja quantização interfere menos na resposta em freqüência do filtro de síntese. Essa transformação é descrita na seguinte subseção.

2.4.4 Cálculo dos coeficientes LSF

Para diminuir o problema da sensibilidade da resposta em freqüência do filtro de síntese aos erros de quantização dos coeficientes LPC, estes são transformados em coeficientes LSF (*Line Spectral Frequency*) cuja quantização gera menor erro por razões que serão vistas a seguir. Os coeficientes LSF são calculados da seguinte maneira:

1. Calculam-se os polinômios $P(z)$, simétrico, e $Q(z)$, anti-simétrico, a partir de $A(z)$:

$$P(z) = A(z) + z^{-p-1}A(z^{-1})$$
$$Q(z) = A(z) - z^{-p-1}A(z^{-1})$$

   onde p é o número de coeficientes preditores. No CELPS são usados p = 10 coeficientes preditores.

2. Definem-se o polinômio $P_1(z)$ como o polinômio $P(z)$ sem a raiz -1, e o polinômio $Q_1(z)$ como o polinômio $Q(z)$ sem a raiz +1, ou seja:

$$P_1(z) = \frac{P(z)}{1+z^{-1}} \; e \; Q_1(z) = \frac{Q(z)}{1-z^{-1}}, \text{ para } p \text{ par e}$$
$$P_1(z) = P(z) \; e \; Q_1(z) = \frac{Q(z)}{1-z^{-2}}, \text{ para } p \text{ ímpar.}$$

3. Calculam-se as raízes dos polinômios $P_1(z)$ e $Q_1(z)$. Como estes polinômios têm a característica de serem simétricos de ordem par, e como as suas raízes são pares de números complexos conjugados, apenas metade delas precisa ser determinada. Ou seja, para p par, $p/2$ raízes de $P_1(z)$ e $p/2$ raízes de $Q_1(z)$, totalizando p

raízes, conseguem representar os polinômios $P_1(z)$ e $Q_1(z)$, e portanto, o filtro de síntese *H(z)*.

Uma característica importante dos coeficientes LSF é que, como as raízes de $P(z)$ e $Q(z)$ estão sobre o círculo unitário, é necessária apenas a determinação de seus ângulos na representação polar complexa para a representação de *H(z)*. Tais valores são os chamados coeficientes LSF.

Para diminuir ainda mais o número de bits para a representação dos coeficientes LSF, são enviadas as diferenças entre os coeficientes consecutivos. Essas diferenças são chamadas de DLSF.

Quando se calcula o filtro de síntese para dois blocos consecutivos do sinal de voz e depois se faz a síntese dos mesmos, a união dos sinais gerados pelos dois filtros pode não ser suave. Isto acontece, porque o trato vocal se movimenta de forma contínua fazendo com que de um bloco para outro a voz já possa ter variado bastante gerando dois filtros bem diferentes. Este problema é amenizado subdividindo o bloco de sinal em quatro sub-blocos de mesmo tamanho e aplicando em cada sub-bloco um filtro cujos coeficientes são uma combinação entre os coeficientes do bloco completo atual e do anterior só que com pesos diferentes para os coeficientes de cada bloco. Esta soma e ponderação dos coeficientes é conhecida como interpolação dos coeficientes LSF. Esta interpolação é feita da seguinte maneira:

$$w_i^n = (1 - q_n)w_i^a + q_n w_i^c,$$

onde w_i^n são os coeficientes do n-ésimo sub-bloco, w_i^a são os coeficientes do bloco anterior, w_i^c são os coeficientes do bloco corrente e q_n= {0,5; 1,0; 1,0; 1,0}[11].

2.4.5 Filtro perceptivo

Uma forma de melhorar a qualidade do codificador é se considerar as características psico-acústicas do ouvido humano. Sons com componentes espectrais de pouca energia são percebidos com mais detalhes do que os sons com alta amplitude. Assim, erros e ruídos em componentes espectrais de pouca energia são mais percebidos do que os em componentes espectrais de alta energia. Tal característica deve ser levada em consideração no cálculo do erro do sinal gerado pelo filtro de síntese quando comparado ao sinal original.

Portanto, o que se faz na prática é aplicar um filtro que é capaz de atenuar as componentes espectrais de alta energia e amplificar as de baixa. Esse filtro é chamado de filtro perceptivo e possui a seguinte equação:

$$W(z) = \frac{A(z)}{A\left(\frac{z}{\gamma}\right)},$$

onde $\gamma \in (0,1)$ é o coeficiente de ponderação o qual indica o grau de mudança no espectro.

Abaixo, a figura 2.6 mostra o filtro perceptivo aplicado ao filtro de síntese da figura 2.4(c) para alguns valores de γ. Nesta figura é possível observar que para γ perto do valor 0.9 é onde se observa o espectro de freqüências mais plano. No entanto, o valor de γ que obteve melhor resultado no codificador CELPS foi 0,75 [7].

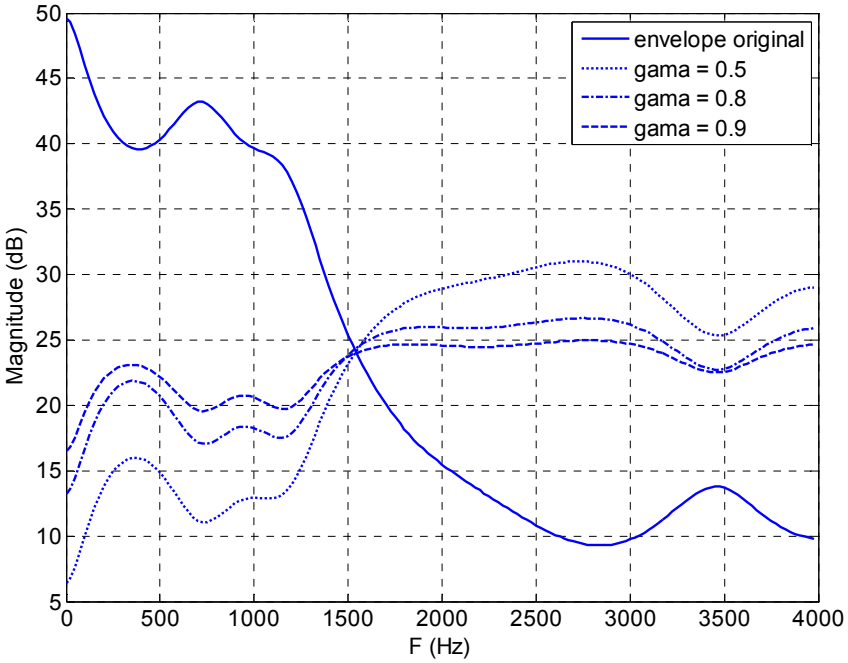


Figura 2.6: Efeito do filtro perceptivo para diferentes valores de γ [7].

2.4.6 Dicionários

Os dicionários são conjuntos de diferentes excitações que são usadas como possíveis entradas para o filtro de síntese. No codificador CELPS, os dicionários armazenam as excitações em um vetor de forma seqüencial tendo como endereço de cada excitação os índices do vetor. Os dicionários podem ser representados como a seguir:

$$C = \{[x_0(n)], [x_1(n)], \cdots, [x_{k-1}(n)]\},$$

onde k representa o número de excitações do dicionário e n o índice de sequência de amostras de cada excitação. O codificador CELPS utiliza dois tipos de dicionários adaptativo e o fixo.

O dicionário adaptativo tem como objetivo estimar com certa precisão tanto a parte sonora quanto a parte surda do sinal de voz. É um dicionário que inicialmente não contém nenhuma excitação. Ele vai sendo preenchido dinamicamente com a soma das melhores excitações escolhidas dos dicionários fixo e adaptativo para os blocos anteriores. O

dicionário adaptativo é implementado como uma fila do tipo FIFO (*first in first out*), ou seja, quando o dicionário fica cheio as primeiras excitações que entraram são as primeiras a serem descartadas. O dicionário é organizado dessa forma, pois, a probabilidade de um sinal de um bloco ser parecido com o do seu bloco anterior em um sinal de voz é muito grande, principalmente em trechos sonoros do sinal de voz.

O dicionário fixo, como o próprio nome sugere, contém excitações fixas que não mudam ao longo do tempo. Essas excitações são escolhidas previamente à criação do codificador. São escolhidas de tal forma que consigam gerar com menor erro possível os blocos de sinal do tipo voz. O dicionário fixo é conhecido tanto pelo codificador quanto pelo decodificador. De modo geral, dicionário fixo é o responsável por estimar a parte surda que o dicionário adaptativo não conseguiu.

Além de se simplesmente usar as excitações dos dicionários, aplicam-se ganhos diferentes em cada excitação de cada dicionário para que, quando combinados, estes sinais possam melhor representar a forma do sinal de voz original bem como sua energia. Observa-se também que, com a combinação dos dois dicionários utilizando diferentes ganhos para cada um, o número possível de combinações entre as excitações aumenta bastante. Isto permite gerar representações mais precisas do sinal de voz original. O ganho de cada dicionário é calculado da seguinte forma:

$$G = \frac{R_{s,d}}{R_{d,d}},$$

onde $R_{s,d}$ é a correlação do sinal de voz original com as respostas correspondentes a cada excitação contida no dicionário em questão e $R_{d,d}$ é a autocorrelação entre as respostas correspondentes à cada excitação contida no dicionário em questão.

2.4.7 Análise por síntese

Depois que o bloco do sinal de voz é separado, janelado e são calculados seus coeficientes LPC, ele é dividido em quatro sub-blocos de tamanho idênticos. Para cada sub-bloco devem ser escolhidas a excitação do dicionário fixo e a excitação do dicionário

adaptativo que gerem, através do filtro de síntese, o sinal com menor erro quando comparado com sinal original, também chamado de sinal alvo.

As etapas da análise acontecem para ambos os dicionários de forma independente. As melhores excitações são depois combinadas. As etapas da análise por síntese são descritas abaixo:

1. Para cada sub-bloco, todas as excitações contidas nos dicionários são submetidas ao filtro de síntese gerado para o sub-bloco gerando assim uma resposta;
2. À essa resposta é aplicado um ganho (subseção 2.4.6) e ela é subtraída do sinal alvo gerando um erro que é guardado;
3. É feita então a busca por todo o dicionário sendo escolhida a excitação que gerar o menor erro.

Para acelerar a busca pelas melhores excitações, os dicionários são inteiramente filtrados pelo filtro de síntese de uma única vez. Assim, as etapas de segmentação e comparação podem ser realizadas com o dicionário já filtrado.

2.4.8 Resumo do codificador CELPS

Na seção 2.4 foram descritos separadamente os componentes utilizados pelo codificador CELPS. Nesta seção é descrito um resumo passo a passo das etapas de funcionamento da codificação e decodificação. O sinal a ser codificado é considerado já digitalizado no formato PCM com freqüência de amostragem de 8 kHz.

Codificador:

1. O sinal de voz é segmentado em blocos de 20 ms (160 amostras) e cada bloco é segmentado em 4 sub-blocos de 5 ms (40 amostras) cada;
2. Os coeficientes LPC do bloco atual são calculados;
3. Os coeficientes LPC são convertidos em coeficientes LSF e depois em DLSF (Differential LSF) para serem trasmitidos;

4. É feita a interpolação dos coeficientes LSF;
5. Os coeficientes LSF do sub-bloco atual são convertidos de volta para coeficientes LPC;
6. É gerado o filtro de síntese do sub-bloco atual a partir dos coeficientes LPC;
7. De posse dos coeficientes LPC, é gerado o filtro perceptivo e este é aplicado ao sub-bloco atual;
8. A resposta à entrada zero do sub-bloco anterior é subtraída do sub-bloco atual e assim é gerado o sinal alvo inicial;
9. Inicia-se o processo de análise por síntese com a procura no dicionário adaptativo. Todas as excitações do dicionário são sintetizadas pelo filtro de síntese e passadas pelo filtro perceptivo;
10. Calcula-se o ganho correspondente à primeira excitação do dicionário adaptativo e gera-se o sinal sintetizado;
11. Subtrai-se o sinal sintetizado do sinal alvo obtido no passo 7 gerando um sinal de erro;
12. Varre-se todo o dicionário adaptativo repetindo os passos 9 e 10 para todas as excitações em busca da excitação que gere o menor EMQ (erro médio quadrático);
13. Quando a melhor excitação for encontrada o sinal alvo é substituído pelo sinal de erro resultante da diferença entre o sinal sintetizado da melhor excitação e o sinal alvo gerado no passo 7. O índice da melhor excitação e seu ganho são guardados para serem enviados;
14. Inicia-se o processo de análise por síntese com a procura no dicionário fixo. Todas as excitações do dicionário são sintetizadas pelo filtro de síntese e passadas pelo filtro perceptivo;
15. Repetem-se os itens 9, 10 e 11 considerando agora o dicionário fixo;
16. Quando a melhor excitação do dicionário fixo for encontrada, seu índice e ganho são guardados para serem enviados. São somadas as melhores excitações dos dois dicionários multiplicadas pelos seus respectivos ganhos gerando a resposta ótima completa;
17. O dicionário adaptativo é atualizado colocando-se a resposta ótima completa no final do dicionário;
18. Por fim, são transmitidos os coeficientes DLSF, os índices das excitações de ambos os dicionários e os respectivos ganhos.

A figura 2.7 abaixo mostra um esquemático simplificado do funcionamento da codificação no sistema CELPS.

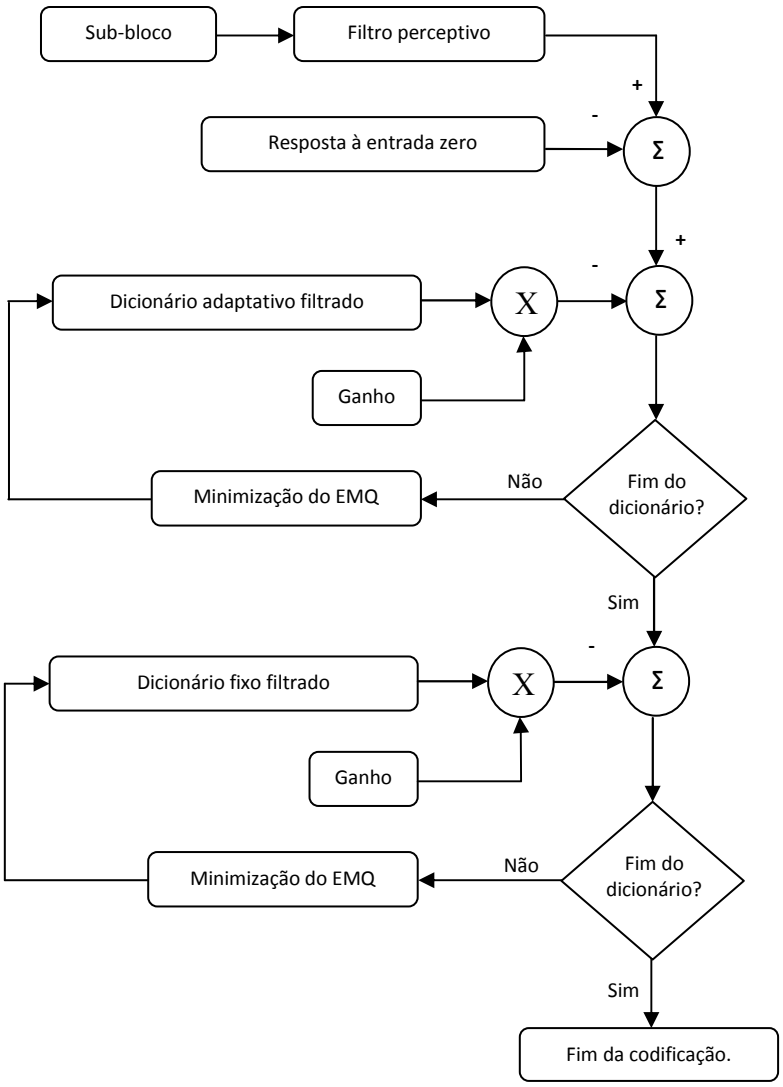


Figura 2.7: Esquemático simplificado do funcionamento da codificação CELPS.

Decodificador:

1. Transformam-se os coeficientes DLSF de volta em coeficientes LPC;
2. Com os índices dos dicionários monta-se a excitação completa a partir das excitações de ambos os dicionários multiplicadas pelos respectivos ganhos;

3. Aplica-se o filtro de síntese à excitação completa gerando a estimativa do sinal original, o sinal decodificado.

A figura 2.8 abaixo mostra um esquemático simplificado do funcionamento da decodificação no sistema CELPS.

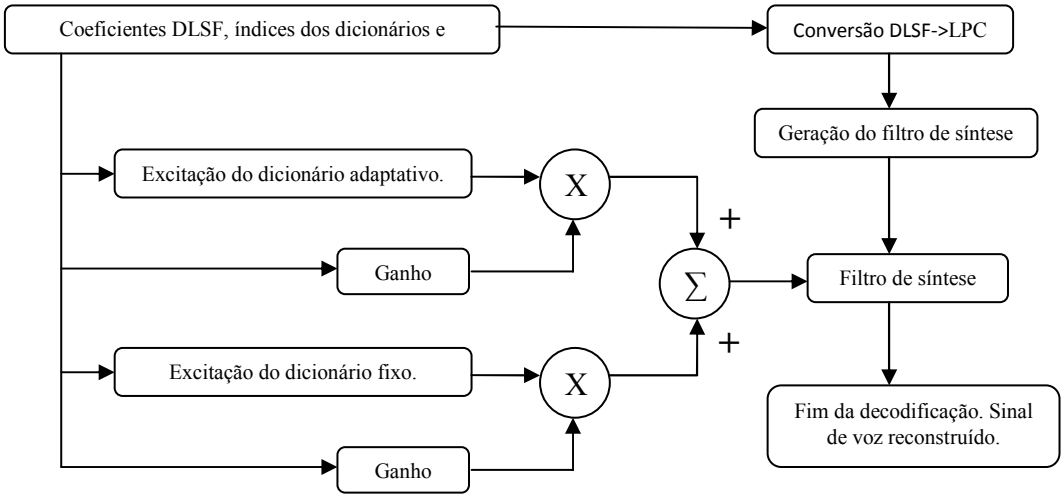


Figura 2.8: Esquemático simplificado do funcionamento da decodificação CELPS.

2.5 Conclusão

Este capítulo apresentou uma visão geral sobre os tipos de codificadores de voz. Foram explicados os codificadores de forma de onda, os codificadores paramétricos e os codificadores híbridos fazendo-se uma comparação entre eles.

O codificador híbrido CELPS foi explicado com mais detalhes por fazer parte do estudo do presente trabalho. A leitura deste capítulo é importante para se entender as melhorias propostas no capítulo 5.

Capítulo 3

Histórico do codificador CELPS

3.1 Introdução

Um dos objetivos deste trabalho foi o de se fazer uma versão do codificador CELPS na linguagem C++ que reunisse todas as melhorias implementas em suas versões anteriores. No entanto, antes de falar sobre esta nova versão, será feito um resumo sobre as versões anteriores. O conhecimento destes trabalhos é importante para se entender a evolução do codificador CELPS e motivar as melhorias propostas pelo presente trabalho.

Antes da proposta de se fazer essa unificação do codificador, havia sete versões as quais serão descritas na seguinte seção. Algumas destas foram desenvolvidas na linguagem C e outras em C++. Cada uma produziu melhorias diferentes que foram sendo somadas nas versões futuras. Tais projetos e suas contribuições também serão comentados na seção seguinte.

3.2 Versões anteriores

Nesta seção será descrita a evolução do codificador CELPS desde a sua idealização até o último trabalho realizado anterior a este. Foram desenvolvidos um total de sete trabalhos voltados para o estudo, melhoria e adaptações do codificador. Sendo estes: uma tese de mestrado, cinco projetos finais de graduação e um projeto de consultoria. Abaixo seguem as descrições de cada trabalho, bem como seus objetivos e resultados, a partir de sua idealização.

3.2.1 Ranniery da Silva Maia, "Codificação CELP e análise espectral de voz" MSc Poli/UFRJ, Março de 2000. [1]

A tese de mestrado defendida por Raniery da Silva Maia em março de 2000 foi a primeira implementação do codificador de voz no LPS baseado no sistema CELP e foi portanto, o início da criação do CELPS. A proposta da tese era a de se implementar um sistema CELP para codificação de voz que operasse a uma taxa de 4,67 Kbps. O sistema proposto incorporava as seguintes implementações:

- Quantização escalar dos parâmetros do filtro de síntese;
- Dicionário adaptativo com atrasos fracionários;
- Dicionário fixo obtido a partir de amostras de ruído branco gaussiano e
- Método de busca seqüencial com otimização de ganhos.

Foram avaliados dois tipos de quantizadores escalares para os parâmetros do filtro de síntese, o LAR (*Log Area Ratio*) e o LSF (*Line Espectral Frequencies*), levando em conta o compromisso entre taxa de bits e distorção harmônica introduzida. Como resultado, o estudo mostrou que:

- A quantização individual de cada LSF é superior à quantização dos LAR no quesito distorção espectral e inferior no quesito estabilidades do filtro quantizado;
- A quantização diferencial das LSF é bem superior à quantização individual porque obteve valores menores de distorção e eliminou o problema de instabilidade do filtro após a quantização.

O sistema foi avaliado com e sem o método de busca rápida no dicionário adaptativo e os resultados foram:

- A avaliação subjetiva informal mostrou que a qualidade dos sinais processados pelo sistema com e sem o procedimento de busca rápida é indistinguível;
- O procedimento de busca rápida proposto reduz de 28% a 32% o tempo de codificação de um sinal de fala pelo sistema proposto.

O sistema foi avaliado através de medidas objetivas de qualidade e testes informais de escuta, sendo comparado ao DoD-CELP que opera a 4,8 kbps. Os resultados obtidos levaram a concluir que, de modo geral, o sistema proposto apresentou desempenho superior com diminuição do tempo de processamento de 28% a 32% com diferença imperceptível na qualidade.

Este trabalho foi a base para as seguintes versões do CELPS. Nele, foram reunidos todos os algoritmos de codificação e foi feita a primeira versão funcional do codificador. Foi escrito na linguagem C. No entanto, por ser uma primeira versão, a implementação do código estava pouco otimizada. Os trabalhos futuros tentaram melhorar essa organização e implementaram novas melhorias.

3.2.2 Bruno de Barros Oliveira, "Análise e teste de um codificador CELP", BSc. Poli/UFRJ, Abril de 2001. [2]

O projeto final de graduação de Bruno de Barros Oliveira deu segmento à tese de mestrado de Ranniery descrita na subseção 3.2.1. O objetivo do projeto do Bruno foi o de descrever detalhadamente o sistema CELPS de codificação de voz desenvolvido em [1]. Também foram aplicadas técnicas de aceleração com perda mínima para se possibilitar a implementação do codificador em tempo real.

Foram realizados os seguintes testes:

- Estudo da influência dos dicionários fixo e adaptativo no sinal obtido;
- Suavização espectral;
- Treinamento do dicionário fixo.

Foram feitas as seguintes modificações na configuração e estrutura do codificador para adaptá-lo para execução em tempo real:

- Foram retirados os atrasos fracionários do dicionário adaptativo;
- Reduziu-se o número de amostras tanto no dicionário fixo quanto no dicionário adaptativo;

- Alterou-se a ordem estrutural, sendo aplicado um novo método de busca no dicionário adaptativo;
- Alterou-se a estrutura do dicionário fixo, realizando-se a superposição dos vetores de código

Este trabalho produziu uma versão acelerada do programa na linguagem C e descreveu o funcionamento do algoritmo do CELPS.

3.2.3 Filipe Castello da Costa Beltrão Diniz, "Implementação de um Codificador de Voz CELP em Tempo Real", Poli/UFRJ, Maio de 2003. [3]

A proposta do projeto final de Filipe Castello da Costa Beltrão Diniz foi a de implementar um sistema de codificação CELP em tempo real e avaliar o desempenho do mesmo. Foram tomados como base os trabalhos descritos nas subseções 3.2.1 e 3.2.2. O codificador foi reestruturado segundo a lógica de programação orientada a objetos. O programa resultante do trabalho foi desenvolvido em ambiente Linux utilizando a linguagem C++. Este trabalho resultou nas seguintes contribuições:

- Desenvolvimento de uma classe para manipular dispositivos de áudio em Linux através do OSS;
- Desenvolvimento de uma classe para manipular vetores de áudio;
- Desenvolvimento de uma classe para obter medidas objetivas da qualidade de um sinal de voz;
- Documentação de todas essas classes em forma de um manual de instruções;
- Fragmentação do código do codificador;
- Reestruturação do codificador segundo a lógica de Orientação a Objetos;
- Desenvolvimento de um codificador em tempo real para Linux, obtendo-se uma Razão sinal por ruído média de 15,77 dB e máxima de 15,91 dB para a saída de voz decodificada do sistema.

Este codificador funciona em uma aplicação em tempo real, ou seja, o som produzido é capturado em um microfone, codificado, decodificado e reproduzido em um espaço muito curto de tempo, dando a sensação de reprodução simultânea.

3.2.4 Eduardo Fonseca Brasil, "Adaptação do codificador CELP à transmissão de voz sobre IP", Poli/UFRJ, Agosto de 2006. [4]

Este trabalho foi motivado inicialmente pela disciplina de Projeto Integrado oferecida no curso de graduação do DEL (Departamento de Eletrônica) da UFRJ. O projeto realizado foi o de um sistema de comunicação de voz sobe IP (VoIP) por uma rede local.

Precisava-se de um codificador para comprimir o sinal de voz. Foi assim que surgiu a necessidade de se fazer uma versão adaptada para este fim. Esta versão então, foi adaptada do codificador de voz CELPS em tempo real desenvolvido em [3]. Para que o sistema ficasse mais enxuto e se adaptasse ao contexto do projeto integrado, o codificador foi transcrito para a linguagem C.

Em seguida, o codificador foi utilizado como base para a realização de um projeto final defendido em agosto de 2006, data esta, posterior aos dois trabalhos citados em seguida nas seções 3.2.5 e 3.2.6 a seguir. A proposta do projeto final foi a de se adaptar o código do codificador do projeto integrado para viabilizar a transmissão de voz pela Internet.

Dentre as modificações realizadas pode-se citar a adição da quantização dos ganhos e coeficientes LSF e a utilização da biblioteca ALSA (*Advanced Linux Sound Architecture*) para aquisição e reprodução de áudio. Toda a infra-estrutura necessária para a transmissão de dados pela Internet, incluindo as principais características dos protocolos básicos da Internet (IP, TCP e UDP), foi desenvolvida e detalhada.

O sistema VoIP foi implementado, porém, está aquém dos sistemas comerciais, por ser apenas uma prova de conceito da tecnologia VoIP. Algumas das pendências críticas do sistema são citadas abaixo:

- Adição do cancelador de eco ao sistema VoIP;
- Adaptar o sistema para redes com perdas como a Internet;
- Detecção de silêncio.

3.2.5 Bruno Catarino Bispo, "Otimização do Codificador CELPS", Poli/UFRJ, Dezembro de 2005. [5]

Este projeto final teve por objetivo a otimização do sistema desenvolvido em [3], seção 3.2.3, na linguagem C++, acrescentando melhorias. Como melhoria, foi incorporado ao procedimento de busca nos dicionários, o duplo ciclo[5] e a inclusão da quantização dos ganhos relativos às excitações dos dicionários, sendo realizada com 3, 4 e 5 bits. Além disso, foi refeita a quantização dos coeficientes LPC, que formam o filtro que modela o trato vocal humano, sendo esta quantização realizada com 32, 36 e 40 bits, gerando versões com diferentes características de taxa e qualidade.

Foram desenvolvidos o CELP 30 ms e CELP Banda Larga, tendo o primeiro o intuito de funcionar em sistemas com largura de banda restrita, enquanto que o segundo, em banda larga (taxa de amostragem de 16 kHz).

E por fim, foi desenvolvida uma interface gráfica utilizando a biblioteca multi-plataforma wxWindows. Nesta interface, o usuário pode escolher qual dos codificadores irá usar e qual das três opções de quantização LPC e de quantização dos ganhos o codificador utilizará, além dos tamanhos dos dicionários e do tempo de execução do codificador escolhido.

Os resultados alcançados levaram às seguintes conclusões:

- A utilização do duplo ciclo proporciona um aumento considerável na qualidade do sinal reconstituído, devendo ser integrado ao codificador de forma definitiva;

- O aumento no número de bits utilizados na quantização LPC não apresenta uma grande melhoria na qualidade do sinal reconstituído;
- O aumento no número de bits utilizados na quantização dos ganhos proporciona um aumento considerável na qualidade do sinal reconstituído;
- O codificador CELP 20 ms apresentou uma boa qualidade na reconstrução do sinal de voz. Qualidade esta que melhorou ao passo que aumentamos a taxa de transmissão, atingindo o valor máximo de 3,4095 na escala MOS com 8,8 kbps de taxa de transmissão;
- O codificador CELP 30 ms apresentou uma qualidade bem inferior em comparação ao CELP 20 ms, atingindo o valor máximo de 2,8167 com 5,867 kbpps de taxa, mas mantendo o entendimento do sinal de voz reconstituído. Essa queda na qualidade foi provocada pela diminuição de 33% na taxa de transmissão em relação ao CELP 20 ms;
- O codificador CELP Banda Larga não apresentou uma qualidade esperada para um codificador que possui o intuito de trabalhar em banda larga. Isso foi devido à utilização da mesma taxa de transmissão do CELP 20 ms, sendo essa taxa muito pequena em termos de banda larga. Porém, isto pode ser contornado com a utilização de quantizadores mais apropriados para o CELP Banda Larga;

A versão descrita nesta subseção ainda não foi utilizada por nenhuma versão futura até o presente momento.

3.2.6 V. L. Latsch, "Projeto Maritaca", COPPE/UFRJ, Janeiro de 2006 [6]

Este projeto foi um trabalho independente realizado por Sergio Lima Netto, Vagner Latsch e Bruno Bispo pela COPPE/UFRJ para uma aplicação comercial. Este teve por objetivo a implementação de um sistema de *software* para voz por protocolo de internet (VoIP).

Neste trabalho foi feita uma versão do codificador CELPS na linguagem C a partir da versão feita para um projeto integrado 3.2.4 [6]. Existia a possibilidade do codificador

ser inserido em *hardware* dedicado e ao mesmo tempo funcionar em ambiente Windows. Portanto, foi optado na ocasião manter o código na linguagem C.

Tendo em vista a aplicação, foram implementadas as seguintes melhorias:

- Cancelador de eco acústico e de linha;
- Detector de silêncio;
- Componentes de gravação de áudio e conferência.

Criou-se nesse momento uma versão do codificador bem estruturada na linguagem C. Esta versão é multi-plataforma e pode ser executada tanto em ambiente Linux quanto em Windows. Esta versão incluía todas as melhorias implementadas por todos os projetos descritos nas subseções anteriores.

3.2.7 Thiago de Moura Prego, "Aperfeiçoamento do codificador de voz CELP", Poli/UFRJ, Agosto de 2007. [7]

Este trabalho teve como objetivo otimizar o sistema de codificação de voz CELPS do projeto descrito na seções 3.2.4 [4] e 3.2.6 [6] .

Para a otimização do sistema foram feitas as seguintes modificações:

- Incorporação do bloco de detecção de silêncio;
- Requantização dos coeficientes do filtro de síntese e dos ganhos dos dicionários adaptativo e fixo;
- Modificação do processo de interpolação dos coeficientes do filtro de síntese;
- Análise do duplo ciclo na busca das melhores excitações do dicionário adaptativo e fixo;
- Interpolação dos coeficientes LSF;
- Requantização dos coeficientes DSLF;
- Reavaliação do coeficiente de ponderação γ (melhor resultado para γ =0,75);

- Foram inseridos o pré-processamento e pós-processamento das amostras de áudio que realizam respectivamente uma pré-ênfase e deênfase no sinal;
- Foi feita uma rotina para diminuir a taxa média de transmissão e a complexidade computacional do algoritmo;

Os resultados foram todos satisfatórios, pois melhoraram ora a qualidade de transmissão, ora a complexidade computacional, ora a taxa média de transmissão. A melhora na qualidade de codificação foi significativa.

3.2.8 Resumo das versões do CELP

Seguem abaixo todas versões do codificador de voz CELP em ordem cronológica:

- Primeira versão (3.2.1 [1]): Foi a primeira implementação do codificador CELP funcional. Foi escrita na linguagem C e apresentava um processamento muito lento;
- Segunda versão (3.2.2 [2]): O processamento foi sensivelmente acelerado e foi adaptado para a versão off-line, incorporando um extenso código voltado para este fim;
- Terceira versão (3.2.3 [3]): Foi feita a reestruturação do codificador na linguagem C++ voltada para o funcionamento de forma *on-line*;
- Quarta versão (3.2.4 [6]): Foi feita uma adaptação da versão descrita em [3] e resumida na seção 3.2.3 na linguagem C para viabilizar a transmissão de voz pela Internet;
- Quinta versão (3.2.5 [4]): Aperfeiçoou o código da versão [3] (seção 3.2.3) escrito em C++ e criou uma interface gráfica para facilitar a utilização do codificador
- Sexta versão (3.2.6 [5]): Foi feita uma versão do codificador CELP na linguagem C a partir da versão descrita em 3.2.3 [3] voltada para a utilização em VoIP;
- Sétima versão (3.2.7 [7]): Implementou outras melhorias no código anterior;

3.3 Conclusão

Este capítulo apresentou um histórico resumido sobre todas as versões já desenvolvidas do codificador CELPS. Foram mostradas as características de cada versão, bem como as melhorias propostas e os resultados. O estudo destas versões motivou a criação da versão proposta pelo presente trabalho. Foi feita uma versão na linguagem C++ a partir da sétima versão descrita na subseção 3.2.7 e da versão descrita na subseção 3.2.6. Este resumo ajudará na compreensão da versão proposta pelo presente trabalho.

Capítulo 4

Versão atual do CELPS

4.1 Introdução

Este capítulo descreve a versão proposta pelo presente trabalho detalhando todos os seus componentes. Esta versão não tem como objetivo único incorporar apenas as novas modificações propostas, mas sim, reestruturar o código do codificador CELPS na forma orientada a objeto e isolar todas as suas funcionalidades principais em uma biblioteca.

Na seção 4.2, são descritos os motivos que levaram a criação desta nova versão. Na seção 4.3, são discutidos os problemas encontrados e as decisões tomadas. Na seção 4.4, é descrito como foi reestruturado o codificador e é feita uma descrição, e breve resumo, de todas as classes, métodos e funções auxiliares da biblioteca criada.

4.2 Motivação

Como visto no capítulo anterior desde a idealização do codificador CELPS algumas versões com diferentes funcionalidades e características foram desenvolvidas. Estas foram criadas para atender diferentes finalidades, como para trabalhar nos modos *on-line* e *off-line* e com voz sobre IP (VoIP). Este trabalho se propõe a fazer uma versão do codificador CELPS que reúna todas as melhorias implementadas nos trabalhos anteriores e as implementadas pelo presente trabalho.

Para permitir a portabilidade do codificador, surgiu a idéia de se fazer uma biblioteca em linguagem orientada a objeto que contivesse somente a implementação das funções do codificador, isolando em um bloco único e auto-contido toda a parte de codificação e decodificação. A biblioteca deveria ter como interface a entrada e saída das amostras de áudio codificadas e decodificadas e algumas funções para passagem e recebimento de parâmetros. Assim, para se fazer uma implementação *on-line, off-line*, em voz sobre IP ou em qualquer outra interface, seria necessário apenas construir o programa

que utilizaria a biblioteca sem a necessidade de se conhecer o interior do codificador. Desta forma, essa biblioteca permitiria a utilização do codificador como uma parte de projetos maiores onde o programador não precisaria necessariamente conhecer sobre seu funcionamento detalhado para utilizá-lo.

Com relação à documentação do código do codificador CELPS, havia alguns trabalhos anteriores que continham uma boa descrição sobre o mesmo mas que se ocuparam em documentar com mais detalhes as partes relativas às suas respectivas melhorias. Portanto, este trabalho visa fazer uma descrição mais completa de todas as classes, métodos e funções auxiliares da biblioteca para facilitar o estudo do codificador em trabalhos futuros.

4.3 Decisões de projeto

Uma vez aceita a proposta da nova versão, foi preciso escolher uma linguagem que atendesse bem aos objetivos. O CELPS é um codificador que está em constante estudo e desenvolvimento pelos professores e alunos do DEL. Então, seria interessante que fosse escolhida uma linguagem de programação que permitisse boa estruturação e portabilidade para facilitar o entendimento e uso por futuros pesquisadores. A linguagem C++, portanto, foi a escolhida por apresentar as seguintes características:

- É uma linguagem bem difundida e com amplo suporte disponível na internet;
- Dá suporte à programação orientada a objeto;
- Permite melhor organização estrutural;
- É amplamente utilizada na programação de DSPs (*Digital Signal Processors*).

Todas essas características facilitam a programação e portabilidade do código. A linguagem C apesar de também ser bem difundida foi deixada de lado uma vez que já existe uma forma de programação mais avançada orientada a objeto proporcionada pela linguagem C++.

Havia, no entanto, a dúvida sobre qual versão deveria ser tomada como base para a criação da nova biblioteca. Duas versões se mostraram mais favoráveis: a versão descrita na seção 3.2.5 e na seção 3.2.6. A segunda, era a versão mais atual do codificador e conseqüentemente com um maior número de melhorias implementadas. No entanto, ela estava escrita na linguagem C. A segunda, era uma versão um pouco mais antiga, mas já estava estruturada segundo uma organização de programação orientada a objeto. Estava escrita na Linguagem C++. Depois de uma análise elaborada junto aos autores dessas duas versões anteriores, optou-se por escrever a biblioteca a partir do código da versão descrita na seção 3.2.6, escrita na linguagem C.

O projeto foi desenvolvido em ambiente Windows utilizando o programa Visual Studio 2008 da Microsoft. No entanto, nada impede a sua utilização em ambiente Linux uma vez que não está sendo utilizada nenhuma biblioteca específica para o ambiente Windows. Esta escolha foi feita devido às facilidades de organização, programação e depuração dos códigos proporcionadas pelas fermentas do programa Visual Studio 2008.

Uma vez escolhida a versão em C foi preciso estudar como fazer a migração para linguagem C++. Uma opção seria partir do zero e implementar todos os métodos desde o começo. Outra, seria a de aproveitar as funções que já estavam escritas em C e apenas organizá-las como métodos de classes que seriam criadas. Esta segunda opção se mostrou mais viável. De modo geral, foram mantidos os nomes da maioria das funções e variáveis e, as que foram mudadas, o foram para nomes similares, que melhor refletissem o objetivo de cada uma, até mesmo para manter uma compatibilidade com a literatura do codificador CELPS composta pelos trabalhos anteriores, facilitando a compreensão dos mesmos.

4.4 Implementação da biblioteca CELPS

No código original, como as funções estavam escritas na forma procedural elas podiam ser chamadas em qualquer parte do programa e a qualquer momento. Agora em C++, foi necessário reorganizar as chamadas dos métodos, uma vez que os mesmos só podem ser chamados quando um objeto da classe a qual pertencem for instanciado no mesmo escopo.

Essa reorganização foi feita resultando em quatro classes:

- CELPS: classe principal do codificador;
- Ccoder: classe responsável pela codificação;
- Cdecoder: classe responsável pela decodificação;
- VAD_FRAME classe responsável pela detecção de silêncio.

Estas classes se relacionam segundo o diagrama da figura 4.1.

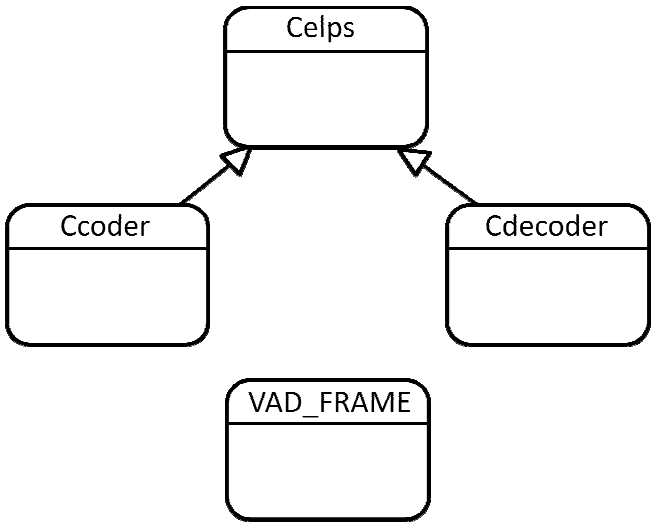


Figura 4.1: Diagrama das classes principais do CELPS.

A classe Ccoder e Cdecoder estabelecem uma relação de herança com a classe CELPS e compartilham os atributos da mesma.

Para composição o módulo de detecção de silêncio, foi implementada a classe de nome VadFrame que neste trabalho foi incorporada ao codificador por ser considerada de alta importância para redução da taxa de bits. Esta não contém nenhuma relação com as demais. Além dessas, o codificador conta com mais algumas funções auxiliares não pertencentes a nenhuma classe.

A seguir serão explicadas as funções de cada classe bem como as das funções auxiliares.

4.4.1 Classe CELPS

A classe CELPS é a classe principal do codificador. Ela contém como atributos todos os parâmetros do codificador tanto para codificação quanto para decodificação. Esta é a classe base para as classes Ccoder e Cdecoder. Seguem abaixo os métodos com suas respectivas definições e sucinta explicação do funcionamento.

.

Métodos:

- `Celps()`: este método é o construtor da classe. Por definição não recebe nenhum argumento e retorna void. Sua função é a de iniciar os parâmetros do objeto da classe Celps.

- `virtual ~Celps()`: este é o método destrutor da classe. Por definição não recebe nenhum argumento e retorna void. Sua função é a de apagar alocações de memória que foram criadas na execução dos métodos da classe Celps.

- `unsigned int celp_quant_fc_gain ()`: este método realiza a quantização dos ganhos do dicionário fixo. Não recebe nenhum argumento e retorna ERROR_OK ou outro código de erro.

- `unsigned int celp_quant_ac_gain ()`:este método realiza a quantização dos ganhos do dicionário adaptativo. Não recebe nenhum argumento e retorna ERROR_OK ou outro código de erro.

- `unsigned int celp_dequant_gains (CELPS_WIN *)`: este método realiza a desquantização dos ganhos do dicionário fixo e adaptativo.

Recebe como argumento um objeto do tipo PCELPS_WIN e retorna ERROR_OK ou outro código de erro.

- `unsigned int celp_apply_sfilter_to_ac ()`: este método realiza a filtragem de todo o dicionário adaptativo pelo filtro perceptual. Não recebe nenhum argumento e retorna ERROR_OK ou outro código de erro.

- `unsigned int celp_apply_sfilter_to_fc ()`: este método realiza a filtragem de todo o dicionário fixo pelo filtro perceptual. Não recebe nenhum argumento e retorna ERROR_OK ou outro código de erro.

- `unsigned int celp_set_complete_response ()`: este método monta a excitação final composta pelas excitações dos dicionários fixo e adaptativo multiplicadas pelos seus respectivos ganhos e aplica o filtro de sintese na excitação final. Não recebe nenhum argumento e retorna ERROR_OK ou outro código de erro.

- `unsigned int celp_ac_update ()`: este método atualiza o dicionário adaptativo colocando o sub-bloco mais recente no começo, deslocando os outros para traz e eliminando o último. Não recebe nenhum argumento e retorna ERROR_OK ou outro código de erro.

- `unsigned int celp_interp_lsf ()`: este método realiza a interpolação dos coeficientes LSF, e já retorna os coeficientes LPC equivalentes interpolados. Não recebe nenhum argumento.

- `unsigned int celp_quant_lsf ()`: este método realiza a quantização dos coeficientes LSF. Não recebe nenhum argumento e retorna ERROR_OK ou outro código de erro.

- `unsigned int celp_dequant_lsf (CELPS_WIN *w)`: este método realiza a desquantização dos coeficientes LSF. Não recebe nenhum argumento e retorna ERROR_OK ou outro código de erro.

4.4.2 Classe Ccoder

Esta classe contém os métodos utilizados para realização da codificação. Ela herda da classe base CELPS os atributos que serão utilizados em seus métodos. Seguem abaixo os métodos com suas respectivas definições e sucinta explicação do funcionamento.

Métodos:

- `Ccoder()`: este método é o construtor da classe. Por definição não recebe nenhum argumento e retorna void. Sua função é a de iniciar os parâmetros do objeto da classe Ccoder.

- `virtual ~Ccoder()`: este é o método destrutor da classe. Por definição não recebe nenhum argumento e retorna void. Sua função é a de apagar alocações de memória que por ventura possam ter sido criadas na execução dos métodos da classe Ccoder.

- `void ccoder_config(int)`: este método recebe como argumento a *flag* que indica utilização ou não da detecção de silêncio. Assim, configura o codificador para usar ou não compressão de silêncio.

- `unsigned int ccoder_set_samples ( short *samples, char *bitstream, int vactive)`: este método recebe no primeiro argumento as amostras de áudio relativas ao bloco de 20 ms na forma de um vetor de elementos do tipo *short*; no segundo, o *bitstream*, que é um vetor de elementos do tipo *char* com tamanho máximo de 22 bytes que devolve a saída codificada da amostras de entradas e o no terceiro, um *flag* do tipo int que indica se o segmento de voz é ativo ou silêncio (caso a detecção de silêncio esteja

desativada este *flag* é desconsiderado). Sua função é executar a codificação preenchendo o vetor *bitstream* e devolvendo o número de bytes usados. Retorna ERROR_OK ou outro código de erro.

- `int ccoder_pre_process(short *samples)`: este método recebe como argumento as amostras de áudio relativas ao bloco de 20 ms na forma de um vetor de elementos do tipo *short*. Ele prepara o bloco para ser codificado aplicando um filtro passa alta (pré-ênfase). Retorna ERROR_OK ou outro código de erro.

- `unsigned int ccoder_search_ac ()`: este método realiza a procura da seqüência do dicionário adaptativo que gera o menor EMQ (erro médio quadrático). Retorna ERROR_OK ou outro código de erro.

- `unsigned int ccoder_search_fc ()`: este método realiza a procura da seqüência do dicionário fixo que gera o menor EMQ. Retorna ERROR_OK ou outro código de erro.

- `unsigned int ccoder_ac_update_target_signal ()`: este método atualiza o sinal alvo removendo do mesmo a resposta da excitação selecionada do dicionário adaptativo. Retorna ERROR_OK ou outro código de erro.

- `unsigned int ccoder_set_pfilter ()`: este método calcula o valor dos coeficientes do numerador e denominador do filtro perceptual. Retorna ERROR_OK ou outro código de erro.

- `Unsigned int ccoder_apply_pfilter_to_sub_block ()`: este método aplica o filtro perceptual ao sub-bloco corrente. Retorna ERROR_OK ou outro código de erro.

- `unsigned int ccoder_remove_zinput_response ()`: este método remove a resposta à entrada zero do sub-bloco passado pelo filtro perceptual. Retorna ERROR_OK ou outro código de erro.

- `unsigned int ccoder_set_impulse_response ()`: este método calcula a resposta ao impulso do filtro perceptual. Retorna ERROR_OK ou outro código de erro.

- `unsigned int ccoder_sub_block_analysis ()`: este método gera os coeficientes LSF interpolados do sub-bloco corrente, calcula o filtro perceptual e aplica-o ao sub-bloco, remove a resposta à entrada zero, filtra os dicionários com o filtro perceptual, procura a melhor frase e ganho dos dicionários adaptativo e fixo, quantiza os ganhos, forma a resposta completa e atualiza o dicionário adaptativo com as melhores frases escolhidas. Retorna ERROR_OK ou outro código de erro.

- `unsigned int ccoder_sub_block_sid_analysis()`: este método gera os coeficientes LSF interpolados do sub-bloco corrente. Retorna ERROR_OK ou outro código de erro.

4.4.3 Classe Cdecoder

Esta classe contém os métodos utilizados para realização da decodificação. Ela herda da classe base CELPS os atributos que serão utilizados em seus métodos. Seguem abaixo os métodos dessa classe, bem como as respectivas definições e sucinta explicação do funcionamento.

Métodos:

- `Cdecoder()`: este método é o construtor da classe. Por definição não recebe nenhum argumento e retorna *void*. Sua função é a de inicia os parâmetros do objeto da classe Cdecoder.

- `virtual~Cdecoder()`: este é o método destrutor da classe. Por definição não recebe nenhum argumento e retorna void. Sua função é a de apagar alocações de memória que possam ter sido criadas na execução dos métodos da classe Cdecoder.

- `unsigned int cdecoder_set_bitstream ( char *bitstream, short *samples)`: este método recebe no primeiro argumento o bitstream, que é um vetor de elementos do tipo *char* com tamanho máximo de 22 bytes contendo as amostras codificadas e no segundo, devolve as amostras de áudio do segmento decodificadas na forma de um vetor de elementos do tipo *short.* Sua função é executar a decodificação preenchendo o vetor *samples* com 160 bytes e retorna o número de bytes escritos.

- `unsigned int cdecoder_get_sub_block ()`: este método não recebe nenhum argumento. Ele calcula os coeficientes do numerador do filtro de síntese e aplica o filtro à excitação decodificada gerando o bloco de voz decodificada e atualiza o dicionário adaptativo com as excitação formada pelas excitações do bloco recebido. Retorna ERROR_OK ou outro código de erro.

- `int cdecoder_pos_process(short *samples)`: este método recebe as amostras de áudio relativas ao bloco de 20 ms na forma de um vetor de elementos do tipo *short*. Ele desfaz a pré-ênfase aplicada ao bloco no início da codificação aplicando um filtro passa-baixas (de-ênfase). Retorna ERROR_OK ou outro código de erro.

- `unsigned int cdecoder_get_sub_block_sid ()`: este método gera os coeficientes LSF interpolados do sub-bloco corrente, gera a excitação com a energia do bloco e faz a síntese do sinal de silêncio. Retorna ERROR_OK ou outro código de erro.

4.4.4 VAD_FRAME

Esta classe contém os métodos e atributos responsáveis pela detecção de silêncios no sinal de voz. Ela não possui nenhuma relação com outras classes. Seguem abaixo os métodos dessa classe, bem como as respectivas definições e sucinta explicação do funcionamento.

Métodos:

- `VAD_FRAME ():` este método é o construtor da classe. Por definição não recebe nenhum argumento e retorna *void*. Sua função é a de inicia os parâmetros do objeto da classe `VAD_FRAME`.

- `virtual ~ VAD_FRAME ():` este é o método destrutor da classe. Por definição não recebe nenhum argumento e retorna void. Sua função é a de apagar alocações de memória que por ventura possam ter sido criadas na execução dos métodos da classe `VAD_FRAME`.

- `unsigned int vad(short *samples,int *value):` este método recebe no primeiro argumento as amostras de áudio relativas ao bloco de 20 ms na forma de um vetor de elementos do tipo *short* e no segundo, devolve a *flag* que indica se o bloco é do tipo ativo ou silêncio. Ele avalia o bloco recebido e detecta se ele é do tipo ativo ou inativo (silêncio). Retorna ERROR_OK ou outro código de erro.

- `int zero_cross():` este método não recebe nenhum argumento. Ele calcula a taxa de cruzamentos por zero do bloco corrente. Retorna ERROR_OK ou outro código de erro.

4.4.5 Funções auxiliares

As funções auxiliares são as responsáveis pela aplicação de filtros e algoritmos matemáticos, além de empacotamento de dados. Tais funções são utilizadas pelos diversos métodos de todas as classes. Seguem abaixo essas funções bem como as respectivas definições e sucinta explicação do funcionamento.

Funções:

- `int energy_calc(double * data)`: esta função recebe como argumento o sinal a ser analisado. Ela calcula a energia do bloco corrente. Retorna ERROR_OK ou outro código de erro.

- `int quant_energy(double energy)`: esta função recebe como argumento o valor da energia a ser quantizada. Sua função é realizar a quantização da energia recebida. Retorna o índice da energia quantizada.

- `double dequant_energy(int index)`: esta função recebe como argumento o índice da energia quantizada. Sua função é realizar a desquantização do índice recebido. Retorna o valor da energia desquantizada.

- `unsigned int hamming (double *, int)`: esta função recebe no primeiro argumento um vetor de elementos do tipo *double* cujo tamanho é determinado pelo segundo argumento. Sua função é calcular a janela do tipo *hamming* do tamanho especificado pelo segundo argumento. Esta janela é devolvida no primeiro argumento. Retorna ERROR_OK ou outro código de erro.

- `void cng(double *data,int length, double energy)`: esta função devolve no primeiro argumento a excitação criada, no segundo, recebe o tamanho da excitação a ser criada e no terceiro, o valor da energia com a qual a excitação deve ser gerada. Sua função é gerar um sinal tipo

ruído do tamanho especificado no segundo argumento e com a energia recebida no terceiro argumento. Esta função não retorna nada.

- `void polimulti (double poli1[], double poli2[], double polir[],int ordem1, int ordem2)`: esta função recebe nos dois primeiros argumentos os coeficientes dos dois polinômios a serem multiplicados, no terceiro, devolve o resultado da multiplicação e nos dois últimos, recebe as ordens dos dois polinômios a serem multiplicados. Sua função é calcular a multiplicação de dois polinômios. Esta função não retorna nada.

- `void filt_pz (double s_input[], double s_output[], double num_filter[], double den_filter[], double buffer_e[], double buffer_s[],int signal_size, int buffer_size)`: esta função recebe no primeiro argumento o sinal a ser filtrado, no segundo, devolve o sinal já filtrado, nos terceiro e quarto recebem respectivamente o numerador e o denominador do filtro, no quarto e quinto, recebem um *buffer* com as amostras passadas, no sexto, o tamanho do sinal e no sétimo, o tamanho do buffer. Sua função é aplicar ao sinal de entrada o filtro formado pelos coeficientes recebidos. Esta função não retorna nada.

- `void filt_sp1(double s_input[], double s_output[], double den_filter[], double buffer[], int signal_size, int buffer_size)`: esta função recebe no primeiro argumento o sinal a ser filtrado, no segundo, devolve o sinal já filtrado, no terceiro, o denominador do filtro, no quarto, o buffer contendo as amostras passadas, no quinto, o tamanho do sinal e no sexto, o tamanho do buffer. Sua função é aplicar ao sinal de entrada o filtro formado pelos coeficientes recebidos. Esta função não retorna nada.

- `unsigned int filt_sp2 (double * s_input, double * s_output, double * den_filter, int signal_size, int buffer_size)`: esta função recebe no primeiro argumento o sinal a ser

filtrado, no segundo, devolve o sinal já filtrado, no terceiro, o denominador do filtro, no quarto, o tamanho do sinal e no quinto, o tamanho do buffer. Sua função é aplicar ao sinal de entrada o filtro formado pelos coeficientes recebidos. Esta função não retorna nada.

- `double inner_prod (double *vec1, double *vec2, int size)`: esta função recebe nos dois primeiros argumentos dois sinais e no terceiro, o tamanho do resultado. Sua função é calcular o produto interno entre os sinais de entrada. Esta função retorna o valor do produto.

- `double quant_esc (double, double [], double [], int, int *)`: esta função recebe no primeiro argumento o coeficiente a ser quantizado, no segundo, a partição, no terceiro, o dicionário , no quarto, o tamanho do dicionário, e no quinto, o índice do coeficiente quantizado. Sua função é realizar a quantização do coeficiente recebido. Esta função retorna o índice do dicionário correpondente ao coeficiente quantizado.

- `double cheb_poly(double *coef,double x,int m) *)`: esta função recebe no primeiro argumento os coeficientes do polinômio, no segundo, o ponto onde o polinômio será avaliado e no terceiro, a ordem do polinômio. Sua função é calcular o valor do ponto no polinômio desejado através da formulação de Chebyshev. Esta função retorna o valor do cálculo do polinômio.

- `unsigned int lpc2lsf (double * coef_lpc, double * coef_lsf, int num_coef)`: esta função recebe no primeiro argumento o vetor de coeficientes LPC, no segundo, devolve o vetor com os coeficientes LSF e no terceiro, o número de coeficientes. Sua função é fazer a transformação dos coeficientes LPC para LSF. Retorna ERROR_OK ou outro código de erro.

- `unsigned int lsf2lpc (double *coef_lsf, double *coef_lpc, int num_coef)`: esta função recebe no primeiro argumento o vetor de coeficientes LSF, no segundo, devolve o vetor com os coeficientes LPC e no terceiro, o número de coeficientes. Sua função é fazer a transformação dos coeficientes LSF para LPC. Retorna ERROR_OK ou outro código de erro.

- `unsigned int levinsonDurbin(double *r, double *A, double *e, int num)`: esta função recebe no primeiro argumento o sinal a ser analisado, no segundo, devolve os coeficientes LPC calculados, no terceiro, devolve a energia residual e no quarto, recebe o número de coeficientes. Sua função é calcular os coeficientes LPC do sinal analisado. Retorna ERROR_OK ou outro código de erro.

- `unsigned int apply_binomial_win(double *in, int length)`: esta função recebe no primeiro argumento o sinal a ser analisado e no segundo, o tamanho da janela. Sua função é aplicar a janela binomial ao sinal analisado. Retorna ERROR_OK ou outro código de erro.

- `unsigned int autocorr(double *s, int length, int lag, double *corr)`: esta função recebe no primeiro argumento o sinal a ser utilizado, no segundo, recebe o tamanho do sinal, no terceiro, o tamanho do *lag* e no quarto, devolve o resultado da auto-correlação. Sua função é calcular a auto-correlação do sinal de entrada. Retorna ERROR_OK ou outro código de erro.

- `void dopack ( char **, int, int, int *)`: esta função recebe no primeiro argumento o ponteiro para o *bitstream* a ser preenchido, no segundo, o conteúdo ou índice a ser guardado no *bitstream,* no terceiro, o número de bits utilizados para se guardar o índice e no quarto, a posição no *bitstream* onde o índice deve ser guardado. Sua função é empacotar o parâmetro recebido no *bitstream* de saída. Esta função não retorna nada.

- `void unpack ( char **, int *, int, int *)`: esta função recebe no primeiro argumento o ponteiro para o *bitstream* a ser lido, no segundo, devolve o índice lido do *bitstream,* no terceiro, o número de bits utilizados para se guarda o índice e no quarto, a posição no *bitstream* onde o índice deve ser lido. Sua função é desempacotar o parâmetro recebido no *bitstream* de entrada. Esta função não retorna nada.

- `void pckd2win (CELPS_WIN *, char *)`: esta função devolve no primeiro argumento uma janela do tipo CELPS e no segundo, recebe o *bitstream* a ser lido. Sua função é desempacotar o *bitstream* recebido e carregá-lo na janela CELPS. Esta função não retorna nada.

- `void win2pckd ( char *, CELPS_WIN *)`: esta função recebe no primeiro argumento uma janela do tipo CELPS e no segundo, devolve o *bitstream* empacotado. Sua função é empacotar o *bitstream* com os dados carregados da janela CELPS. Esta função não retorna nada.

- `int sign(double x)`: esta função recebe um numero como argumento. Sua função é verificar o sinal do número. Esta função retorna 1 se o número for maior que zero, -1 se for menor que zero ou 0 se o número for igual a zero.

4.5 Conclusão

Foram descritos os motivos que levaram a elaboração de uma nova versão do CELPS bem como as dificuldades encontradas para sua realização. Em seguida, foi descrito o novo codificador na linguagem C++, a estrutura desta nova versão organizada em classes e a função dos métodos e funções.

A versão do codificador CELPS descrita nesse capítulo é a versão final deste projeto final e já incorpora todas as modificações propostas pelo mesmo. Estão modificações são descritas no capítulo 5.

Capítulo 5

Modificações no codificador CELPS

5.1 Introdução

O objetivo deste capítulo é descrever os estudos realizados sobre o codificador CELPS e as modificações propostas pelo presente trabalho. Para a realização dos testes das modificações, foram utilizadas as 596 frases do banco de treinamento citado na seção 2.2.3. Para cada teste, todas as frases do banco de treinamento foram codificadas, decodificadas e tiveram seus valores na escala MOS calculado através do programa PESQ. Também foram guardadas a taxa média de bits gerada por cada arquivo codificado. Em seguida, foi feita uma nova média entre as taxas médias de cada arquivo e também a média dos valores MOS gerados por cada arquivo. Desta forma, pôde-se obter uma taxa média de bits por segundo e a qualidade média na escala MOS para cada modificação testada.

5.2 Resposta à entrada zero

A resposta à entrada zero é o sinal gerado na saída do filtro de síntese quando a excitação de entrada é nula. Essa resposta do filtro é um sinal indesejado que se não for compensado na codificação causa interferência no sinal codificado. Quando no decodificador a excitação de cada sub-bloco de 40 amostras é montada, esta passa pelo filtro de síntese gerando um sinal com um número maior de amostras do que o da excitação de entrada. Essas amostras que ultrapassam as 40 originais caracterizam a resposta a entrada zero e interferem no sub-bloco seguinte. Conhecendo-se esse problema, o codificador realiza a mesma síntese do sinal gerado no decodificador e então subtrai do sinal do sub-bloco atual a resposta à entrada zero do anterior gerando um novo sinal alvo modificado. Dessa forma, a procura das melhores excitações é feita para o novo sinal alvo, pois, no decodificador a resposta a entrada zero será gerada e somada ao sinal reconstruído compensando assim esse problema.

O codificador CELPS considera apenas a influência das amostras de um determinado sub-bloco no primeiro sub-bloco seguinte. O objetivo do presente estudo foi o de se avaliar a influência dos resquícios gerados por um sub-bloco também no segundo e terceiro sub-blocos mais próximos. Para isso, foram coletados os coeficientes LPC e as respectivas excitações de cada sub-bloco de um sinal de voz de teste qualquer de 4 segundos de duração. Esses dados foram coletados modificando-se o código fonte do decodificador de forma a gerar dois arquivos contendo tais informações. Um sinal de voz de 4 segundos gera 800 sub-blocos o que já permite fazer uma avaliação geral dessa influência de forma satisfatória.

Após a coleta dos dados, os mesmos foram carregados no programa MatLab. Inicialmente, foram feitas as sínteses de algumas excitações com seus respectivos filtros de síntese gerados pelos coeficientes LPC de cada excitação. No entanto, em cada excitação foram adicionadas mais 120 amostras de valor nulo totalizando 160 amostras por excitação, de forma a tornar a excitação do tamanho de 4 sub-blocos. Essa alteração foi feita para se pudessem observar as últimas 120 amostras do sinal na saída do filtro. As figuras 5.1(a), 5.1(b), 5.1(c) e 5.1(d) mostram a saída do filtro de síntese para 4 sub-blocos escolhidos aleatoriamente do sinal de voz teste.

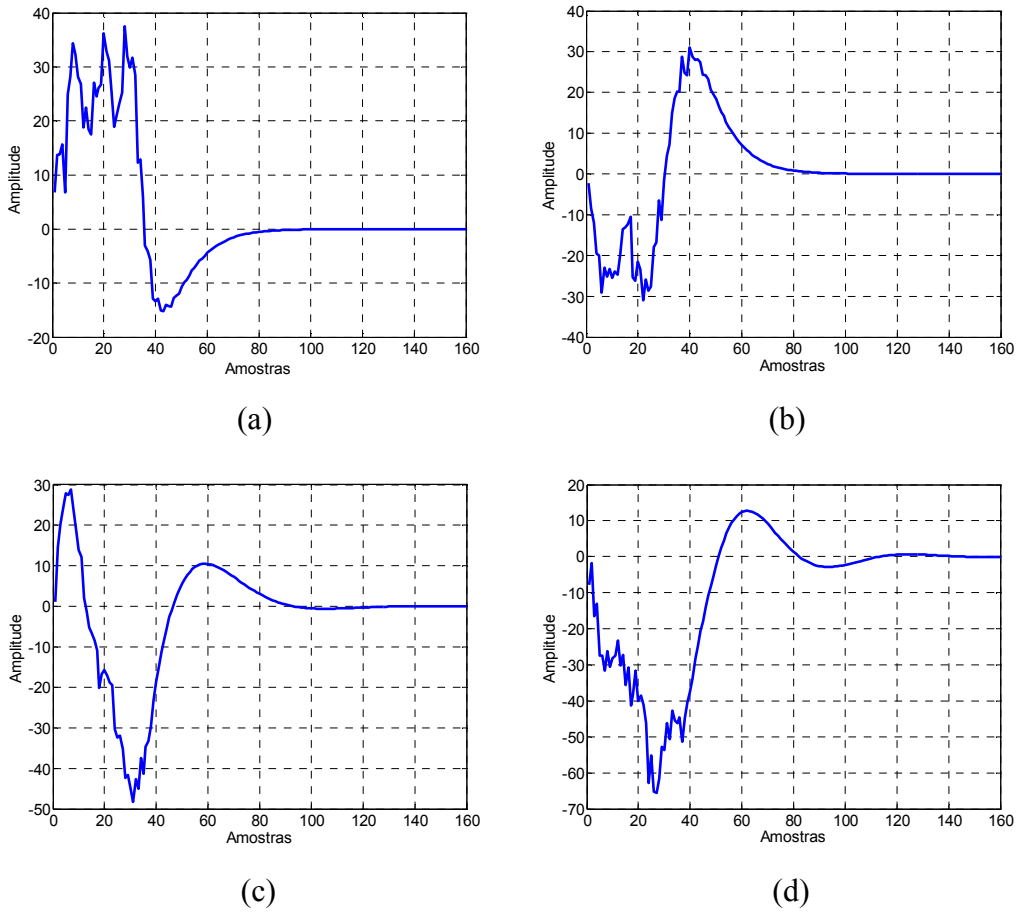


Figura 5.1: (a), (b), (c) e (d) Respostas à entrada zero de quatro sub-blocos de um sinal de voz.

Analisando-se a figura 5.1, pode-se observar que de fato a energia do sinal de voz reconstruído concentra-se em sua maioria nos dois primeiros grupos de 40 amostras, ou seja, nos dois primeiros sub-blocos. Escolhendo-se mais sub-blocos aleatórios em diferentes arquivos de áudio também se observou que a energia se concentra nos dois primeiro sub-blocos. Para se obter um resultado mais geral, foram calculadas as energias dos primeiros, segundos, terceiros e quartos sub-blocos gerados pelo filtro de síntese a partir de cada um dos 800 sub-blocos do sinal teste. Em seguida, foi feita a média das energias de todos os primeiros, segundos, terceiros e quartos sub-blocos do sinal teste gerando uma média da energia que se concentra em cada sub-bloco gerado na síntese. A figura 5.2 mostra o resultado desse teste representando de forma percentual a concentração de energia média em cada sub-bloco da resposta do filtro de síntese. Pode-se observar na figura 5.2 que, a energia gerada no terceiro e quarto sub-bloco seguintes são praticamente nulas.

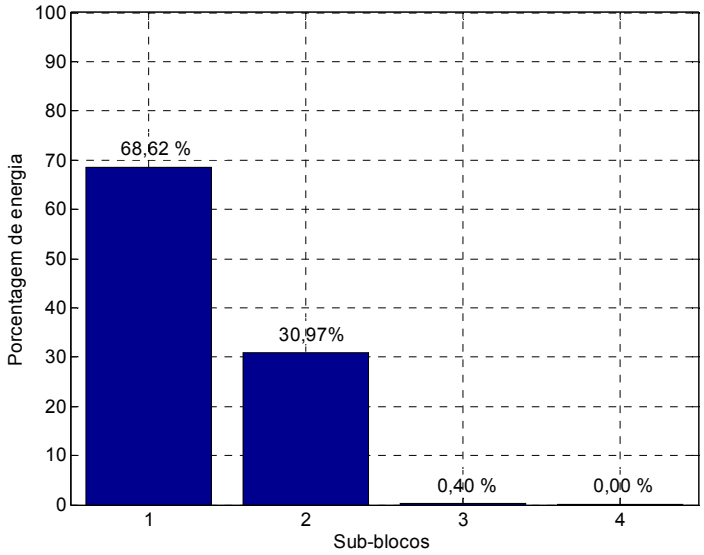


Figura 5.2: Porcentagem de energia média por sub-bloco.

Portanto, a conclusão deste estudo foi a de que não faz diferença considerar as interferências geradas pela resposta à entrada zero de sub-blocos vizinhos mais afastados. A interferência gerada no primeiro sub-bloco vizinho, no entanto, é significativa e então deve ser considerada, como já é feito no codificador CELPS atual. Portanto, não foi realizada nenhuma mudança relativa a este aspecto.

5.3 Tamanho dos dicionários

O codificador CELPS antes das modificações propostas pelo presente trabalho, implementava o dicionário adaptativo com tamanho fixo de 512 excitações e o adaptativo com tamanho máximo de 512 excitações. Não se havia feito ainda um estudo mais amplo sobre como diferentes combinações de tamanhos dos dicionários fixo e adaptativo poderiam afetar a qualidade da codificação.

Neste trabalho então, foram testadas diversas combinações. Os valores de tamanho testados para cada dicionário foram de 256, 512, 1024 e 2048 excitações gerando um total de 16 combinações. Como os dicionários são endereçados através de uma palavra binária, eles são usados com tamanhos de potência de 2 de forma a aproveitar todos os endereços

gerados pelos bits de endereçamento. Os gráficos da figura 5.3 mostram a nota MOS em função da taxa de bits gerada para os quatro valores possíveis de tamanho do dicionário fixo variando-se apenas o tamanho do dicionário adaptativo. O dicionário fixo foi escolhido como referência por nenhuma razão especial, analogamente poderia ter-se utilizado o dicionário adaptativo como referência. A tabela 5.1 mostra todos os dados obtidos no teste.

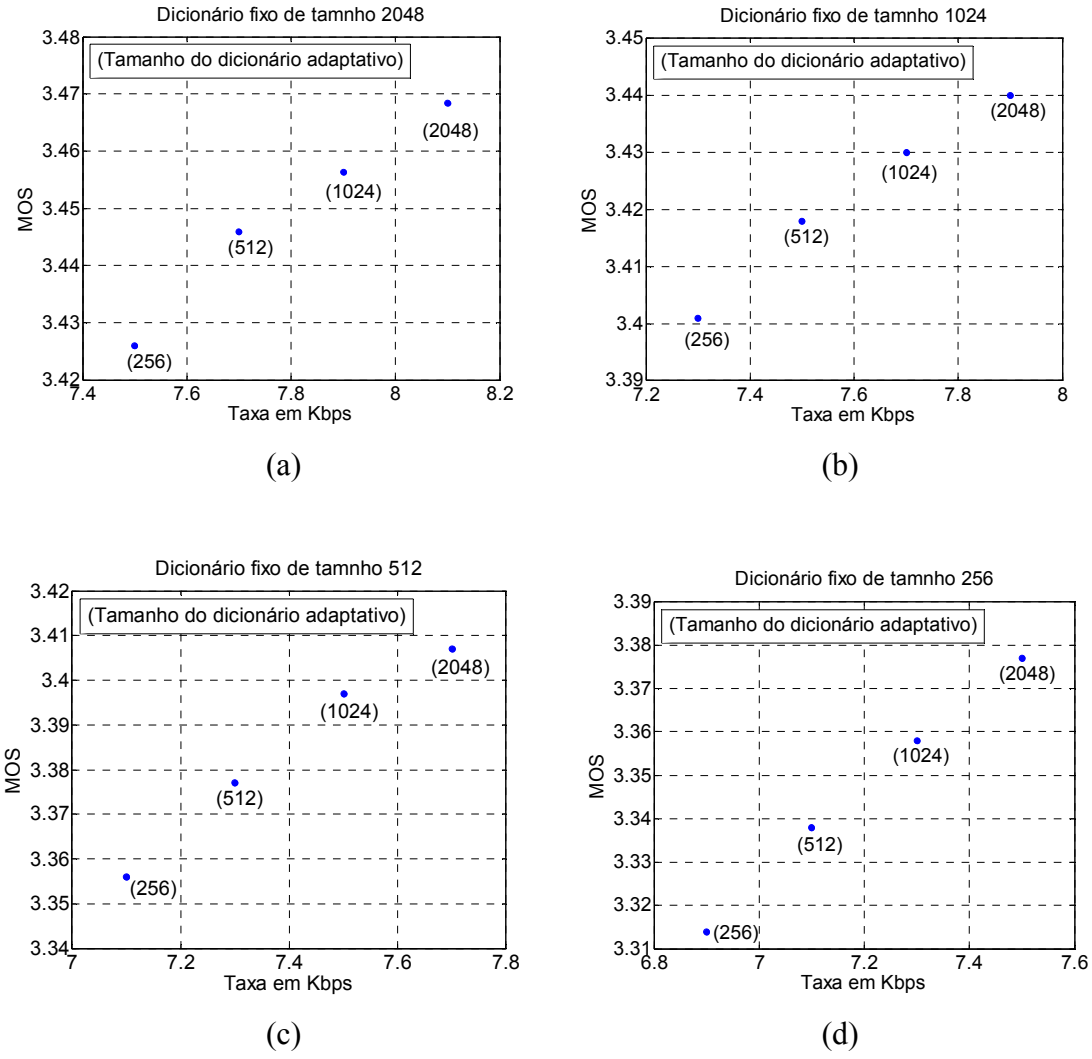


Figura 5.3: (a), (b), (c) e (d) Relação MOS por taxa para todas as combinações de tamanhos de dicionário fixo e adaptativo. O tamanho do dicionário adaptativo é mostrado ao lado de cada ponto nos gráficos.

Tabela 5.1: Notas MOS e taxas gerados para diferentes combinações de tamanhos de dicionários.

Tamanho do dicionário fixo	Tamanho do dicionário adaptativo	MOS	Taxa em Kbps
2048	2048	3,468	8,1
2048	1024	3,456	7,9
2048	512	3,446	7,7
2048	256	3,426	7,5
1024	2048	3,440	7,9
1024	1024	3,430	7,7
1024	512	3,418	7,5
1024	256	3,402	7,3
512	2048	3,407	7,7
512	1024	3,397	7,5
512	512	3,377	7,3
512	256	3,356	7,1
256	2048	3,377	7,5
256	1024	3,358	7,3
256	512	3,338	7,1
256	256	3,314	6,9

A partir dos gráficos da figura 5.3 é possível observar que, à medida que se aumenta o tamanho dos dicionários, a taxa de bits e a qualidade também aumentam, o que era de se esperar. No entanto, o gráfico também mostra que existem alguns pontos onde para uma mesma taxa são gerados arquivos com valores de MOS diferentes. Este é um resultado interessante, pois, revela a combinação de tamanhos dos dicionários fixo e adaptativos que geram melhor qualidade para uma mesma taxa. A tabela 5.2 mostra as combinações de tamanhos de ambos os dicionários que geram uma melhor nota MOS para uma mesma taxa. Os valores desta tabela são mostrados na figura 5.4 que constitui uma curva de possibilidades de escolha da melhor qualidade para uma taxa desejada.

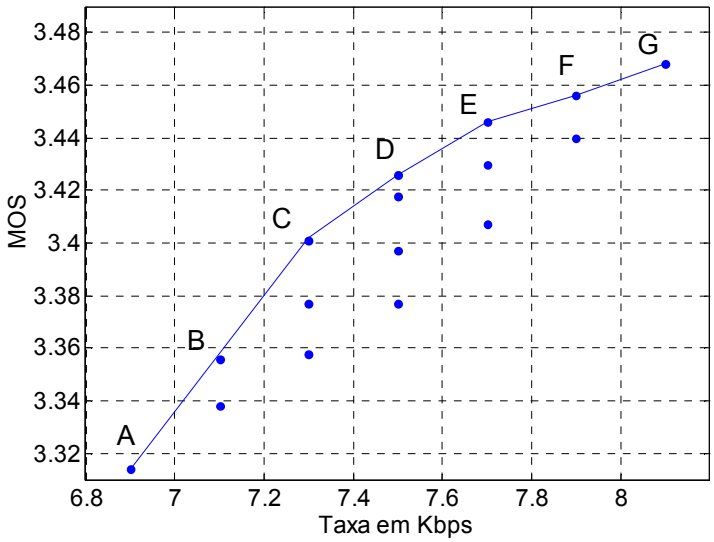


Figura 5.4: Curva de possibilidades de escolha de nota MOS por taxa de bits.

Tabela 5.2: Combinações de tamanhos de ambos os dicionários que geram uma melhor nota MOS para uma mesma taxa.

Ponto	Tamanho do dicionário fixo	Tamanho do dicionário adaptativo	MOS	Taxa em Kbps
A	256	256	3,314	6,9
B	512	256	3,356	7,1
C	1024	256	3,402	7,3
D	2048	256	3,426	7,5
E	2048	512	3,446	7,7
F	2048	1024	3,456	7,9
G	2048	2048	3,468	8,1

A partir da tabela 5.2 é possível observar que para uma taxa de 7,3 Kbps é possível conseguir uma nota MOS de 3,402 combinando-se o dicionário fixo de tamanho 1024 com o adaptativo de tamanho 256. O codificador CELPS utilizava o dicionário fixo e o adaptativo de tamanho 512 gerando uma taxa de 7,3 Kbps e uma nota MOS igual a 3,377. Esta diferença na nota MOS é pequena e perceptivelmente pode não fazer diferença. No

entanto, ao se somar este pequeno aumento a outros gerados por outras modificações pode-se chegar a um aumento perceptível na qualidade.

Portanto, os resultados deste trabalho mostram que é possível melhorar a qualidade do codificador CELPS mantendo a sua taxa original apenas alterando o tamanho do dicionário fixo de 512 para 1024 e diminuindo o adaptativo de 512 para 256. Esta alteração foi incluída na versão final codificador CELPS gerada pelo presente trabalho. Outra contribuição importante deste estudo é que foi gerada uma curva de possibilidades onde é possível escolher a melhor qualidade possível para uma dada taxa escolhida em função dos tamanhos dos dicionários.

5.4 Detecção de silêncio

Ao se analisar a voz humana, é possível perceber claramente que momentos de ausência de voz, ou silêncio, ocorrem freqüentemente, pois, o silêncio faz parte da comunicação por voz. Não existe, portanto, a necessidade de se codificar os sinais considerados como silêncio com a mesma taxa de bits de sinais de voz ativa. O codificador CELPS já possui um algoritmo de detecção de silêncio. O presente trabalho propõe mudanças nesse algoritmo e realiza o estudo de seus parâmetros.

5.4.1 Reconstrução dos blocos de silêncio

Na versão anterior do codificador CELPS [7] os blocos classificados como silêncio eram substituídos no decodificador por um sinal de silêncio pré-gravado. O objetivo deste estudo é modificar a forma de decodificação dos blocos do tipo silêncio para uma forma que gere sinais mais parecidos com os blocos de silêncio originais.

Para tal, foi proposto o envio dos coeficientes DLSF dos blocos classificados como silêncio e também o valor de suas energias. Assim, no decodificador seria gerado um ruído com a mesma energia do bloco original que serviria de excitação para o filtro gerado a

partir dos coeficientes LSF/LPC recebidos. Esse tipo de reconstrução permite uma aproximação mais precisa do sinal original. A tabela 5.3 mostra a organização do *bitstream* dos blocos do tipo ativo e do tipo silêncio.

Tabela 5.3: Organização do *bitstream* para blocos do tipo ativo e tipo silêncio;

Conteúdo do *bitstream*	Bloco ativo	Bloco silêncio
	Número de bits	Número de bits
Tipo do frame	2	2
Energia	-	4
Coeficientes DLSF	32	32
Índice do dicionário adaptativo	32	-
Índice do dicionário fixo	40	-
Ganho do dicionário adaptativo	24	-
Ganho do dicionário fixo	20	-
Total	150	38
Taxa em Kbps	7,5	1,9

A figura 5.5 mostra o resultado da comparação de qualidade e taxa entre a técnica proposta e a técnica utilizada em [7], chamada a partir de agora de técnica antiga, para quatro combinações entre os parâmetros energia limite e taxa de cruzamentos por zero que geraram os melhores resultados na técnica antiga e na nova técnica proposta. A tabela 5.4 mostra os dados da figura 5.5.

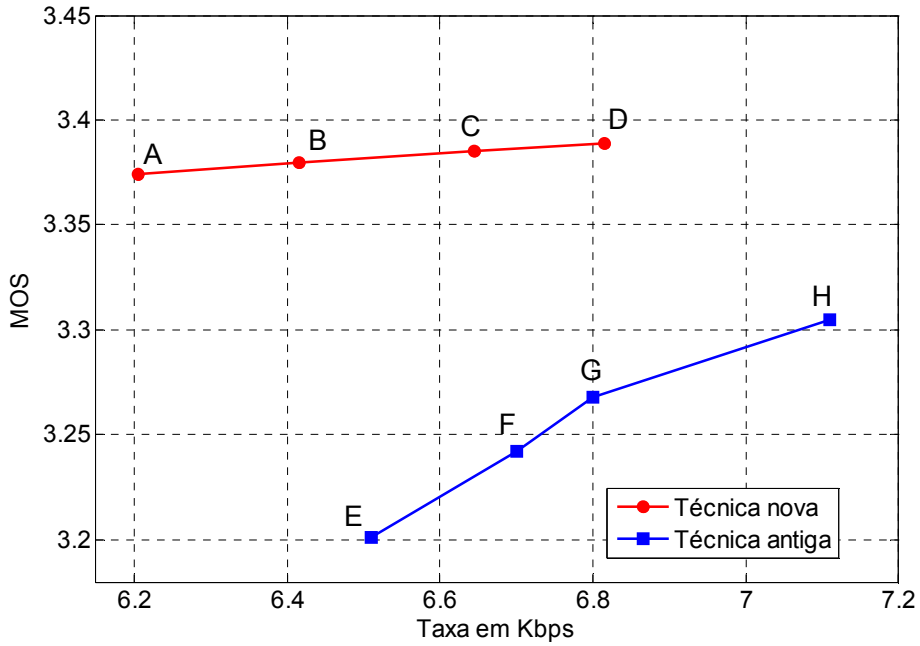


Figura 5.5: Comparação entre nota MOS e taxa para o método antigo e o novo método proposto de reconstrução de blocos do tipo silêncio.

Tabela 5.4: Combinações de parâmetros energia limite e taxa de cruzamentos por zero que geraram as melhores notas MOS na técnica antiga e na técnica nova.

Ponto	Energia limite	Taxa de cruzamentos por zero limite	MOS	Taxa em Kbps
Técnica nova				
A	35,0	52	3,374	6,204
B	33,0	61	3,380	6,416
C	31,5	55	3,385	6,645
D	30,5	52	3,389	6,815
Técnica antiga				
E	40,0	57	3,201	6,510
F	39,5	57	3,242	6,700
G	39,0	57	3,268	6,800
H	38,0	57	3,305	7,110

Comparando-se os resultados da qualidade por taxa gerados pelos dois métodos de reconstrução de blocos de silêncio, é possível observar que a nova técnica proposta obteve

uma melhora de mais de 0,1 na escala MOS gerando taxas ainda menores que a técnica antiga.

5.4.2 Análise dos parâmetros de detecção de silêncio

O codificador CELPS implementa a detecção de silêncio da seguinte forma[7]:

1. Uma vez tendo o sinal dividido em blocos, é calculada a energia do bloco corrente;
2. Se a energia calculada for maior que uma energia limite especificada o sinal é classificado como sonoro. Caso contrário é calculado a taxa de cruzamentos por zero do bloco;
3. Se a taxa de cruzamentos for maior que uma taxa de cruzamentos limite especificada o bloco é considerado como sendo do tipo surdo. Caso contrário, como do tipo silêncio;
4. Caso seja classificado como do tipo silêncio é gerado apenas um *flag* informando que o bloco é do tipo silêncio. No decodificador o bloco é substituído por um sinal de silêncio pré-gravado.

O valor de energia limite utilizado é de 39,5 dB e o valor da taxa de cruzamentos por zero limite é de 57 cruzamentos por zero por bloco de 20 ms. O codificador CELPS com esses valores de parâmetros consegue uma nota MOS de 3,242 com taxa de 6,6 Kbps [7]. No estudo anterior descrito em [7] foi avaliado apenas a qualidade de codificação para alguns poucos níveis de energia limite. O presente trabalho faz a avaliação da qualidade de codificação produzida por um número maior de combinações entre valores de energia limite e taxa de cruzamentos por zero. Isto, com o objetivo de determinar as combinações de parâmetros que geram as melhores notas na escala MOS.

Foram então testadas diferentes combinações de valores de energia limite e taxa de cruzamentos por zero. O número total de combinações possíveis desses valores é muito grande. Portanto, foram feitos testes pontuais com valores extremos e foi verificado que valeria apena apenas considerar certos valores limites para cada parâmetro, pois, os

melhores resultados provavelmente estariam dentro desse intervalo. A tabela 5.5 mostra os limites de valores testados, o passo de incremento para cada parâmetro e o número de combinações testadas.

Tabela 5.5: Limites dos parâmetros testados, passos de incremento e número de combinações.

	Mínimo	Máximo	Passo	Total de combinações por parâmetro
Energia limite	30	42	0,5	25
Taxa de cruzamentos por zero limite	52	62	1	11
Total de combinações entre os parâmetros testadas				275

A figura 5.6 mostra o valor na escala MOS e a taxa em Kbps correspondente a cada combinação entre os valores dos parâmetros da tabela 5.5 e destaca os melhores resultados que são mostrados na tabela 5.6.

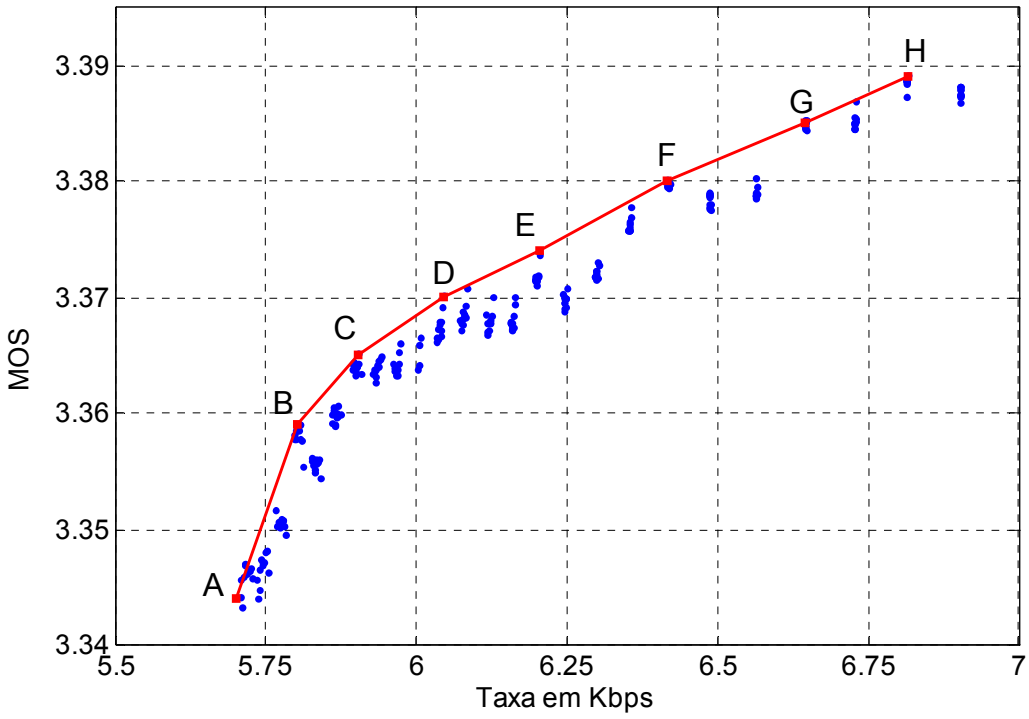


Figura 5.6: Curva de possibilidades gerada a partir das combinações descritas na tabela 5.5.

Através da figura 5.6 pode-se traçar uma curva de possibilidades ótimas, ou seja, uma curva que contém as combinações de parâmetros que geram os melhores resultados. Dessa forma, torna-se possível a escolha de uma combinação ótima para uma determinada taxa ou qualidade desejada. As combinações que produziram os melhores resultados são mostradas na tabela 5.6.

Tabela 5.6: Combinações de parâmetros energia limite e taxa de cruzamentos por zero que geraram melhores resultados de nota MOS e taxa.

Ponto	MOS	Taxa em Kbps	Energia limite	Taxa de cruzamentos por zero limite
A	3,344	5,709	42,0	61
B	3,359	5,806	40,5	55
C	3,365	5,904	39,0	53
D	3,370	6,045	37,0	52
E	3,374	6,204	35,0	52
F	3,380	6,416	33,0	61
G	3,385	6,645	31,5	55
H	3,389	6,815	30,5	52

5.4.2 Requantizacão dos coeficientes DLSF de blocos ativos e inativos

Com a inclusão da modificação da forma de reconstrução dos blocos de silêncio descrita na subseção anterior, torna-se interessante realizar uma quantização dos coeficientes LSF de forma separada para blocos do tipo ativo e do tipo silêncio. Isto porque, os coeficientes dos blocos do tipo ativo podem ser muito diferentes dos coeficientes dos blocos do tipo silêncio. Quando esses coeficientes são quantizados juntos, geram níveis de quantização sub-otimizados.

Este trabalho propõe o uso dos coeficientes DLSF quantizados separadamente para blocos do tipo ativo e silêncio não somente para o caso onde a compressão da taxa por detecção de silêncio estiver ativada. É proposto também, para o caso em que a detecção de silêncio é utilizada mas sem haver diminuição da taxa, ou seja, os blocos do tipo silêncio são reconstruídos da mesma forma que os blocos do tipo ativos. Nesse caso, é

esperada uma melhora da qualidade do codificador mesmo quando a compressão por detecção de silêncio não estiver ativada, pois, desta forma, os coeficientes DLSF estarão otimizados para cada tipo de bloco. A requantização destes coeficientes não gera nenhum aumento de taxa ou complexidade computacional, portanto, é uma proposta interessante de ser testada.

Foi realizada, portanto, a requantização dos coeficientes DLSF separadamente para blocos do tipo ativo e blocos do tipo silêncio. Para isso, o programa do codificador foi modificado de forma a copiar num arquivo do tipo texto os coeficientes LSF dos blocos do tipo ativo e do tipo silêncio separadamente, isto para cada combinação de parâmetros da tabela 5.4. Em seguida, através de um script do programa MatLab que utiliza a função lloyds, foram calculados os dicionários ótimos para os coeficientes LSF de cada tipo de bloco. Para que fosse possível fazer uma comparação de forma mais precisa entre os resultados obtidos com a requantização e os anteriores mostrados na tabela 5.6, foi feita a requantização dos coeficientes DLSF de blocos ativos e de silêncios para cada combinação de energia limite e taxa de cruzamentos por zero da tabela 5.6.

Inicialmente, foi feito o teste com a compressão por detecção de silêncio ativada. Os resultados obtidos após a requantização estão listados na tabela 5.7. A figura 5.7 mostra a comparação entre os resultados obtidos antes e depois da requantização.

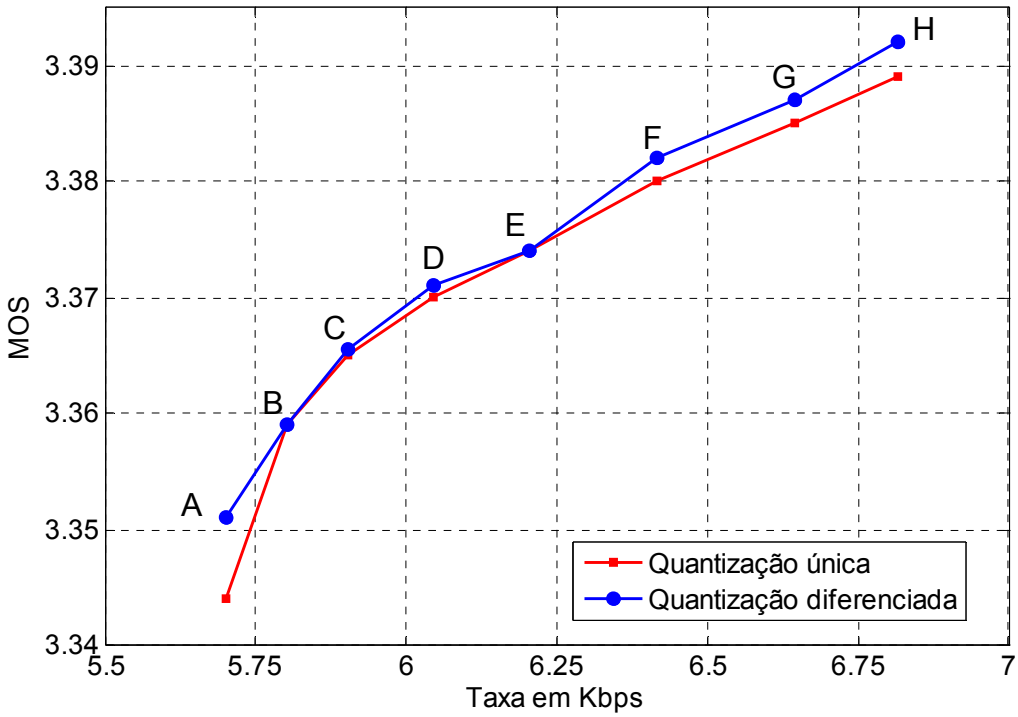


Figura 5.7: Comparação entre os resultados obtidos antes e depois da requantização

Tabela 5.7: Resultados de MOS e taxa obtidos para as mesmas combinações de parâmetros depois da requantização.

Ponto	MOS	Taxa em Kbps	Energia limite	Taxa de cruzamentos por zero limite
A	3,351	5,709	42,0	61
B	3,359	5,806	40,5	55
C	3,366	5,904	39,0	53
D	3,370	6,045	37,0	52
E	3,371	6,204	35,0	52
F	3,374	6,416	33,0	61
G	3,386	6,645	31,5	55
H	3,392	6,815	30,5	52

Na figura 5.7 é possível observar que a quantização dos coeficientes DLSF de forma separada gerou um pequeno aumento na qualidade de codificação para uma mesma taxa. Este aumento de qualidade tende a crescer com o aumento da precisão da técnica de detecção de silêncio, pois, quanto mais precisa for a discriminação dos blocos, mais

otimizada será a quantização e, portanto, mais fielmente os coeficientes DLSF representarão o sinal original. A detecção de silêncio utilizada pelo codificador CELPS implementa uma técnica simples, no entanto, já permite uma queda de taxa considerável ao preço de um pequena queda na qualidade.

Para tornar-se a combinação padrão de parâmetros de detecção de silêncio utilizada no codificador CELPS, foi escolhido o ponto G, pois, este apresenta uma boa relação entre nota MOS e taxa, e possui uma nota MOS aceitável. Os valores dos dicionários de quantização dos coeficientes DLSF para ambos os tipos de blocos para a combinação de parâmetros do ponto G encontram-se no apêndice A.

Uma vez escolhida a combinação de parâmetros ótimos de detecção de silêncio, foi feito então um teste com a detecção de silêncio ativada mas com a representação dos blocos de silêncio sendo feitas da mesma forma que os blocos ativos. A tabela 5.8 mostra o resultado dessa nova técnica comparado com os resultados da técnica antiga sem detecção de silêncio ativada onde todos os tipos de blocos eram quantizados juntos sem nenhum tipo de separação.

Tabela 5.8: Comparação de taxa e qualidade entre a versão anterior e a atual.

Codificador sem compressão por detecção de silêncio	MOS	Taxa em Kbps
Versão anterior	3,414	7,5
Versão atual	3,401	7,3

É possível observar que a quantização separada dos blocos do tipo ativo e silêncio gera uma melhoria muito pequena na qualidade. Da mesma forma para este caso, com a melhoria da técnica de detecção de silêncio essa quantização separada tende a melhorar, pois, os níveis de quantização gerados estariam representando um grupo mais concentrado de possíveis valores de coeficientes DLSF.

5.5 Conclusão

Comparando os resultados da modificação proposta com a versão anterior do codificador CELPS, observa-se que houve uma considerável melhoria na relação qualidade por taxa com as modificações implementadas na detecção de silêncio. A tabela 5.9 mostra um resumo comparativo entre a versão anterior do codificador CELPS e a desenvolvida neste projeto.

Tabela 5.9: Comparações entre a versão anterior e a versão atual

Codificador		MOS	Taxa média em Kbps
Versão anterior	Sem detecção de silêncio	3,414	7,5
	Com detecção de silêncio	3,242	6,7
Versão atual	Sem detecção de silêncio	3,401	7,3
	Com detecção de silêncio	3,386	6,6

O valor da taxa média na configuração sem detecção de silêncio da versão anterior consta na referência [7] como sendo de 7,4 Kbps. No entanto, essa taxa não levava em consideração os dois bits necessários em cada *bitstream* para representação do tipo do bloco. Uma vez considerando esses dois bits a taxa passa de 7,4 Kbps para 7,5 Kbps. O presente trabalho considera esses bits. Portanto, para que pudesse ser feita uma comparação exata entre os valores obtidos na versão antiga e na versão deste trabalho, os valores da taxa de bits do resultados de [7] foram acrescidos de 0,1 Kbps, ou seja, 2 bits a mais por cada bloco de 20 ms.

O estudo das diferentes combinações entre energia limite e taxa de cruzamentos por zero limite foi importante, pois, permitiu o conhecimento das combinações ótimas geradoras dos melhores resultados. A implementação da nova técnica de reconstrução dos blocos de silêncio e a requantização dos coeficientes DLSF também foram importantes para a melhoria da relação qualidade por taxa. Os resultados obtidos com as novas modificações na detecção de silêncio foram significativos, pois, permitiram uma queda de

quase 9% da taxa de bits quando comparada com a taxa sem compressão por detecção de silêncio, ou seja, de 7,5 Kbps para 6,6 Kbps.

Capítulo 6

Conclusão

6.1 Contribuições do trabalho

O presente trabalho descreveu o sistema CELPS de codificação de voz no capítulo 2 resumindo as etapas de funcionamento do mesmo. Foram apresentadas as características desse codificador como a análise LPC, tipos de dicionários de excitações e análise por síntese. Foi vista a descrição resumida do algoritmo de funcionamento da codificação e decodificação no CELPS. Foi feita também no capítulo 3, a documentação do histórico da evolução do codificador CELPS desde a sua primeira versão.

O presente trabalho contribuiu com a construção de uma biblioteca (capítulo 4), escrita na linguagem C++, contendo o núcleo do codificador CELPS isolando todas as suas funcionalidades de codificação e decodificação das possíveis interfaces de uso. A interface de teste utilizada foi a de linha de comando no sistema operacional Windows. Esta biblioteca visa facilitar o estudo do codificador por trabalhos futuros.

Outras contribuições do trabalho foram (capítulo 5):

- O estudo da influência da resposta à entrada zero gerada por um bloco de sinal de voz nos seus blocos vizinhos: concluiu-se que a resposta à entrada zero gerada por um bloco somente interfere significativamente no primeiro bloco seguinte não necessitando assim, ser considerada para mais de um bloco vizinho;

- O estudo da influência de diferentes combinações de tamanhos dos dicionários fixo e adaptativo na qualidade da codificação e na taxa gerada: concluiu-se que o dicionário fixo influencia mais na qualidade final de codificação do que o dicionário adaptativo sendo, portanto, mais vantajoso utilizar-se um dicionário fixo de tamanho maior e um adaptativo de tamanho menor para uma mesma taxa;

- O estudo da influência dos parâmetros de detecção de silêncio energia limite e taxa de cruzamentos por zero na qualidade da codificação e taxa gerada: foi gerada uma curva de possibilidades contendo as combinações entre esses dois parâmetros que geram a melhor qualidade para diferentes taxas desejadas;

- A inclusão da técnica de reconstrução dos blocos do tipo silêncio através do envio dos coeficientes LSF e da energia do sinal: esta técnica gerou uma melhora na qualidade de codificação quando comparada com a técnica anterior;

- Separação entre os coeficientes LSF dos sinais ativos e inativos (silêncios) e requantização dos mesmos: como os dois tipos de sinais possuem características diferentes, esta separação e requantização permitiram um aumento qualidade de codificação. Esta modificação combinada com a modificação no tamanho dos dicionários conseguiu uma nota MOS de 3,386 para uma taxa de 6,6 Kbps contra 3,242 para uma taxa de 6,7 Kbps no modo antigo .

Os estudos realizados contribuíram para o aperfeiçoamento do codificador CELPS. A qualidade de codificação com detecção de silêncio para um mesma taxa melhorou quando comparada com as versões anteriores [5] e [7]. A contribuição da criação da biblioteca na linguagem em C++ também facilitará a continuação dos estudos e desenvolvimento do codificador CELPS.

6.2 Propostas para trabalhos futuros

Existe um número grande de focos de estudo na área de codificação de voz. No codificador CELPS muitos algoritmos ainda podem ser melhorados e acelerados ou ainda podem ser criados novos. O tema é bastante amplo. A seguir estão algumas sugestões de continuação deste trabalho:

- Estudo mais sofisticado para melhorar o algoritmo de detecção de blocos de silêncio de forma a reduzir o número de interpretações erradas de blocos sonoros e surdos como silêncio;

- Estudo visando a melhoria da interpolação dos coeficientes LSF utilizando, por exemplo, um janela assimétrica no lugar da janela atual;

- Estudo da possibilidade de um codificador de taxa variável, controlada externamente;

Referências Bibliográficas

[1] Maia, R. da S., "Codificação CELP e Análise Espectral de Voz", Tese de M.Sc., PEE-COPPE/UFRJ, Março de 2000.

[2] Oliveira, B. de B., "Análise e Testes de um Codificador CELP", Projeto Final DEL-Poli / UFRJ, Rio de Janeiro, RJ, Brasil, Abril de 2001

[3] Diniz, F. C. da C. B., "Implementação de um Codificador de Voz CELP em Tempo Real", Projeto Final, Poli/UFRJ, Maio de 2003.

[4] Bispo, B. C., "Otimização do Codificador CELPS", Poli/UFRJ, Projeto Final, Dezembro de 2005.

[5] Latsch, V. L., "Projeto Maritaca, COPPE/UFRJ", Janeiro de 2006.

[6] Brasil, E. F., "Adaptação do Codificador CELP à Transmissão de Voz Sobre IP", Projeto Final, Poli/UFRJ, Agosto de 2006.

[7] Prego, T. de M., "Aperfeiçoamento do Codificador de voz CELP", Poli/UFRJ, Projeto Final, Agosto de 2007.

[8] Netto, Sergio Lima, "Codificação de voz para sistemas de telecomunicações", EPOLI-COPPE/UFRJ

[9] P. S. R. Diniz, E. A. B. da Silva, S. L. Netto, "Processamento Digital de Sinais: Projeto e Análise de Sistemas", Bookman, 2004.

[10] A. M. Kondoz, "Digital Speech: Coding for Low Bit Rate Communications Systems", Wiley, 1999.

[11] F. C. da C. B. Diniz, "Implementação de Codificador de Voz CELP em Tempo Real", Projeto Final DEL-EE/UFRJ, Maio de 2003.

[12] ITU-T Recommendation P.800: “Methods for subjective determination of transmission quality”,1996.

[13] ITU-T Recommendation P.862: “Perceptual Evaluation of Speech Quality (PESQ)”, version 1.2 - 2 August 2002

[14] J. R. Deller, Jr., J. H. L. Hansen, and J. G. Proakis, “Discrete-Time Processing of Speech Signals”, IEEE Press 2002.

[15] J. R. Deller, J. G. Proakis, J. H. L. Hansen, “Discrete-Time Processing of Speech Signals”, MacMillan Coll Div, 1995.

[16] Woodard, J. P., “Commonly speech codecs.”, 1995. http://www-mobile.ecs.soton.ac.uk/speech_codecs/common_classes.html

Apêndice A

Dicionários de quantização dos coeficientes DLSF para parâmetros de detecção de silêncio energia limite igual a 31,5 e taxa de cruzamentos por zero limite igual a 55. Os números de bits utilizados para representação de cada coeficiente DLSF são diferentes e se distribuem da seguinte forma [4 3 4 4 4 3 3 3 3 1]. Esta distribuição foi estudada em [7].

Para blocos ativos:

DLSF(0)	Dicionário	{0.0612, 0.0791, 0.0936, 0.1061, 0.1173, 0.1276, 0.1370, 0.1459, 0.1543, 0.1622, 0.1698, 0.1770, 0.1872, 0.2054, 0.2362, 0.2989}
	Partição	{0.0702, 0.0863, 0.0998, 0.1117, 0.1224, 0.1323, 0.1415, 0.1501, 0.1582, 0.1660, 0.1734, 0.1821, 0.1963, 0.2208, 0.2675}
DLSF(1)	Dicionário	{0.0480, 0.0779, 0.1089, 0.1426, 0.1839, 0.2383, 0.3138, 0.4392, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000}
	Partição	{0.0630, 0.0934, 0.1257, 0.1632, 0.2111, 0.2760, 0.3765, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000}
DLSF(2)	Dicionário	{0.0649, 0.0998, 0.1319, 0.1650, 0.1996, 0.2349, 0.2707, 0.3072, 0.3424, 0.3768, 0.4128, 0.4519, 0.4957, 0.5506, 0.6353, 0.7829}
	Partição	{0.0823, 0.1158, 0.1484, 0.1823, 0.2172, 0.2528, 0.2889, 0.3248, 0.3596, 0.3948, 0.4323, 0.4738, 0.5231, 0.5930, 0.7091}
DLSF(3)	Dicionário	{0.0845, 0.1395, 0.1877, 0.2291, 0.2663, 0.3013, 0.3367, 0.3741, 0.4160, 0.4633, 0.5168, 0.5778, 0.6502, 0.7373, 0.8434, 0.9888}
	Partição	{0.1120, 0.1636, 0.2084, 0.2477, 0.2838, 0.3190, 0.3554, 0.3951, 0.4397, 0.4901, 0.5473, 0.6140, 0.6937, 0.7904, 0.9161}
DLSF(4)	Dicionário	{0.0635, 0.1060, 0.1461, 0.1868, 0.2261, 0.2630, 0.2975, 0.3327, 0.3690, 0.4090, 0.4562, 0.5136, 0.5830, 0.6666, 0.7823, 0.9732}

	Partição	{0.0848, 0.1261, 0.1665, 0.2065, 0.2446, 0.2803, 0.3151, 0.3508, 0.3890, 0.4326, 0.4849, 0.5483, 0.6248, 0.7244, 0.8778}
DLSF(5)	Dicionário	{0.1027, 0.1883, 0.2631, 0.3318, 0.4092, 0.5123, 0.6575, 0.8883, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000}
	Partição	{0.1455, 0.2257, 0.2975, 0.3705, 0.4608, 0.5849, 0.7729, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000}
DLSF(6)	Dicionário	{0.1135, 0.2083, 0.2806, 0.3456, 0.4162, 0.5065, 0.6247, 0.8091, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000}
	Partição	{0.1609, 0.2445, 0.3131, 0.3809, 0.4613, 0.5656, 0.7169, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000}
DLSF(7)	Dicionário	{0.0899, 0.1669, 0.2316, 0.2894, 0.3479, 0.4199, 0.5266, 0.7099, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000}
	Partição	{0.1284, 0.1993, 0.2605, 0.3186, 0.3839, 0.4732, 0.6182, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000}
DLSF(8)	Dicionário	{0.1216, 0.2061, 0.2736, 0.3342, 0.4007, 0.4867, 0.6150, 0.8408, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000}
	Partição	{0.1639, 0.2399, 0.3039, 0.3675, 0.4437, 0.5508, 0.7279, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000}
DLSF(9)	Dicionário	{0.1981, 0.3805, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000}
	Partição	{0.2893, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000}

Para blocos de silêncio:

DTX_DLSF(0)	Dicionário	{0.0612, 0.0791, 0.0936, 0.1061, 0.1173, 0.1276, 0.1370, 0.1459, 0.1543, 0.1622, 0.1698, 0.1770, 0.1840, 0.1907, 0.2017, 0.2263}
	Partição	{0.0702, 0.0863, 0.0998, 0.1117, 0.1224, 0.1323, 0.1415, 0.1501, 0.1582, 0.1660, 0.1734, 0.1805, 0.1873, 0.1962, 0.2140}

DTX_DLSF(1)	Dicionário	{0.0821, 0.1065, 0.1292, 0.1529, 0.1780, 0.2064, 0.2420, 0.2938, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000}
	Partição	{0.0943, 0.1178, 0.1411, 0.1655, 0.1922, 0.2242, 0.2679, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000}
DTX_DLSF(2)	Dicionário	{0.0838, 0.1864, 0.2300, 0.2656, 0.2926, 0.3179, 0.3414, 0.3635, 0.3835, 0.4046, 0.4253, 0.4465, 0.4693, 0.4957, 0.5285, 0.5851}
	Partição	{0.1351, 0.2082, 0.2478, 0.2791, 0.3053, 0.3297, 0.3524, 0.3735, 0.3941, 0.4149, 0.4359, 0.4579, 0.4825, 0.5121, 0.5568}
DTX_DLSF(3)	Dicionário	{0.0954, 0.1353, 0.1646, 0.1873, 0.2072, 0.2249, 0.2417, 0.2572, 0.2729, 0.2895, 0.3064, 0.3252, 0.3453, 0.3704, 0.4030, 0.4550}
	Partição	{0.1153, 0.1499, 0.1759, 0.1973, 0.2161, 0.2333, 0.2495, 0.2650, 0.2812, 0.2979, 0.3158, 0.3352, 0.3578, 0.3867, 0.4290}
DTX_DLSF(4)	Dicionário	{0.0827, 0.1241, 0.1672, 0.2037, 0.2316, 0.2559, 0.2783, 0.2989, 0.3195, 0.3395, 0.3601, 0.3818, 0.4064, 0.4343, 0.4682, 0.5182}
	Partição	{0.1034, 0.1457, 0.1855, 0.2177, 0.2437, 0.2671, 0.2886, 0.3092, 0.3295, 0.3498, 0.3709, 0.3941, 0.4204, 0.4513, 0.4932}
DTX_DLSF(5)	Dicionário	{0.1554, 0.2043, 0.2412, 0.2740, 0.3075, 0.3417, 0.3860, 0.4577, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000}
	Partição	{0.1799, 0.2227, 0.2576, 0.2907, 0.3246, 0.3638, 0.4218, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000}
DTX_DLSF(6)	Dicionário	{0.1914, 0.2380, 0.2733, 0.3046, 0.3380, 0.3720, 0.4100, 0.4654, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000}
	Partição	{0.2147, 0.2556, 0.2889, 0.3213, 0.3550, 0.3910, 0.4377, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000}
DTX_DLSF(7)	Dicionário	{0.1705, 0.2156, 0.2491, 0.2800, 0.3110, 0.3443, 0.3852, 0.4470, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000}
	Partição	{0.1930, 0.2324, 0.2645, 0.2955, 0.3277, 0.3647, 0.4161, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000}
DTX_DLSF(8)	Dicionário	{0.1765, 0.2247, 0.2625, 0.2948, 0.3266, 0.3606, 0.3998, 0.4562, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000}
	Partição	{0.2006, 0.2436, 0.2786, 0.3107, 0.3436, 0.3802, 0.4280, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000}
DTX_DLSF(9)	Dicionário	{0.2430, 0.3436, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000}
	Partição	{0.2933, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000}