UNIVERSIDADE FEDERAL DE PERNAMBUCO

CENTRO DE TECNOLOGIA E GEOCIÊNCIAS

DEPARTAMENTO DE ELETRÔNICA E SISTEMAS

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

ROBERTO FERNANDO BATISTA SOTERO FILHO

NOVAS ABORDAGENS PARA CODIFICAÇÃO DE VOZ E RECONHECIMENTO AUTOMÁTICO DE LOCUTOR PROJETADAS VIA MASCARAMENTO PLENO EM FREQUÊNCIA POR OITAVA

RECIFE

2017

ROBERTO FERNANDO BATISTA SOTERO FILHO

NOVAS ABORDAGENS PARA CODIFICAÇÃO DE VOZ E RECONHECIMENTO AUTOMÁTICO DE LOCUTOR PROJETADAS VIA MASCARAMENTO PLENO EM FREQUÊNCIA POR OITAVA

Dissertação submetida ao curso de Pós-Graduação em ENGENHARIA ELÉTRICA da Universidade Federal de Pernambuco, como parte dos requisitos necessários à obtenção do grau de Mestre em ENGENHARIA ELÉTRICA.

Área de concentração: Comunicações
Linha de pesquisa: Processamento de voz

Orientador: Prof. *Docteur* Hélio Magalhães de Oliveira

RECIFE

2017

Catalogação na fonte
Bibliotecária Valdicéa Alves, CRB-4 / 1260

S717n Sotero Filho, Roberto Fernando Batista.

Novas abordagens para codificação de voz e reconhecimento automático de locutor projetadas via mascaramento pleno em frequência por oitava / Roberto Fernando Batista Sotero Filho - 2017.
260 folhas, Il.; Tabs; Abr.e Sigl.; e Simbs.

Orientador: Prof. Dr. Hélio Magalhães de Oliveira.

Dissertação (Mestrado) – Universidade Federal de Pernambuco. CTG. Programa de Pós-Graduação em Engenharia Elétrica, 2017.
Inclui Referência, Anexos e Apêndices.

1. Engenharia Elétrica. 2. *Vocoder*. 3. Reconhecimento automático de locutor. 4. Mascaramento em frequência. I. Oliveira, Hélio Magalhães (Orientador). II. Título.

UFPE

621.3 CDD (22. ed.) BCTG/2017 - 292

Universidade Federal de Pernambuco

Pós-Graduação em Engenharia Elétrica

PARECER DA COMISSÃO EXAMINADORA DE DEFESA DE DISSERTAÇÃO DO MESTRADO ACADÊMICO DE

ROBERTO FERNANDO BATISTA SOTERO FILHO

TÍTULO

"NOVAS ABORDAGENS PARA CODIFICAÇÃO DE VOZ E RECONHECIMENTO AUTOMÁTICO DE LOCUTOR PROJETADAS VIA MASCARAMENTO PLENO EM FREQUÊNCIA POR OITAVA"

A comissão examinadora composta pelos professores: HÉLIO MAGALHÃES DE OLIVEIRA, DES/UFPE, RICARDO MENEZES CAMPELLO DE SOUZA, DES/UFPE e JULIANO BANDEIRA LIMA, POLI/UPE sob a presidência do primeiro, consideram o candidato **ROBERTO FERNANDO BATISTA SOTERO FILHO** **APROVADO**.

Recife, 30 de outubro de 2009.

RAFAEL DUEIRE LINS Coordenador do PPGEE	**HÉLIO MAGALHÃES DE OLIVEIRA** Orientador e Membro Titular Interno
JULIANO BANDEIRA LIMA Membro Titular Externo	**RICARDO MENEZES CAMPELLO DE SOUZA** Membro Titular Interno

Dedico a Paulo Sotero (in memoriam), meu avô.

AGRADECIMENTOS

Agradeço, primeiramente, a **DEUS**, por me conceder saúde e força para buscar meus objetivos, e por colocar pessoais tão especiais na minha vida, sem as quais certamente não teria conseguido concluir este trabalho.

Agradeço também:

A todos os funcionários e professores do Departamento de Eletrônica e Sistemas da UFPE, em especial ao meu orientador, Prof. **Hélio Magalhães**, por ter me aceitado como seu orientando e por toda sua dedicação em minha orientação, estando sempre presente nas horas em que precisei.

A todos os meus colegas da pós-graduação, pelo harmonioso e agradável convívio, em especial aos prezados: *André Ricardson*, *Arthur Barreto*, *Brenno Miro*, *Caio Barros*, *Daniel Simões*, *Douglas Contente*, *Eduarda Simões*, *Eurico Moura*, *Gabriel Dantas*, *Giovanna Angelis*, *Márcio Lima*, *Moisés Cordeiro* e *Victor Oliveira*.

Ao *Santa Cruz Futebol Clube*, meu time de coração, por me ensinar a nunca desistir dos meus objetivos, mesmo nos momentos mais difíceis.

À minha irmã **Viviane**, pelo apoio e incentivo, e as meus sobrinhos **João Filho** e **Mariana**, pela alegria que sempre me transmitiram.

Aos meus avós **Paulo** (*in memoriam*) e **Maria José**, **Antonio** e **Maria de Lourdes**, pelo amor e força.

À minha noiva **Marília**, por ser tão importante na minha vida. Devido a seu companheirismo, amizade, paciência, compreensão e apoio, este trabalho pode ser concretizado.

E, em especial, aos meus pais **Roberto** e **Dione**, pelo carinho e dedicação em minha criação, nunca medindo esforços para me proporcionar uma boa educação. Sintam-se igualmente vitoriosos!!

RESUMO

A área de processamento digital de sinais de voz (PDSV) é uma das mais importantes do processamento digital de sinais. Como sub-áreas relevantes do PDSV estão a Codificação da Voz e o Reconhecimento Automático de Locutor (RAL). Esta dissertação propõe uma nova abordagem para um *vocoder* baseado no Mascaramento Pleno em Frequência por Oitavas (MPFO) em adição a uma técnica de preenchimento espectral via distribuição beta de probabilidade. O método do MPFO consiste em simplificar a magnitude do espectro em frequência do sinal, considerando apenas uma amostra por oitava. Tal abordagem, que oferece um compromisso entre taxa de bits (e.g. 2,7 kbits/s), complexidade, inteligibilidade e qualidade dos sinais de voz, permitiu a criação de um novo formato binário de representação digital da voz: o formato *voz*. Apresenta-se, também, um novo método de baixa complexidade computacional para RAL, baseando-se em uma das propriedades-chave da percepção auditiva humana: o mascaramento acústico em frequência. O vetor característico dos quadros do sinal de voz é representado pela fração média das amplitudes dos tons de mascaramento em cada oitava. Ambos os tipos de reconhecimento de locutor (de texto dependente e de texto independente) são estudados. Os resultados confirmam que o algoritmo proposto oferece um compromisso entre a complexidade e a taxa de identificações corretas (típico 85%), sendo atrativo para aplicações em sistemas embarcados.

Palavras-chave: *Vocoder*. Reconhecimento automático de locutor. Mascaramento em frequência.

ABSTRACT

Digital processing of speech signals (DPSS) is one of the most important areas of digital signal processing. Voice coding and automatic speaker recognition (ASR) are relevant DPSS sub-fields. This dissertation introduces a new vocoder scheme, which is based on full frequency masking per octave (FFMO), jointly with a new spectral stuffing technique through the beta probability distribution. The FFMO method consists of simplifying the magnitude of the voice spectrum. It retains just one spectral sample per octave. This approach offers a tradeoff between the bit rate (e.g., 2.7 kbits/s), complexity, intelligibility and voice quality. A new file format, termed *voz*, was proposed. A novel and low-complexity ASR technique, based one of the key-properties of the human hearing perception - the auditory frequency masking - is also presented. The feature vectors of voice frames are represented by the average amplitude of the largest spectral samples within each octave. Both text-dependent and text-independent speaker recognition is investigated. Results support a tradeoff between recognition efficiency (typically 85%) and complexity of this kind of vocoder-based systems, being thereby attractive for embedded systems.

Keywords: *Vocoder*. Speaker automatic recognition. Frequency masking.

LISTA DE ILUSTRAÇÕES

Figura 1 – Imagem da onda longitudinal criada com as sucessões de rarefações e compressões das partículas de ar. 23
Figura 2 – Imagem da anatomia do ouvido humano. 26
Figura 3 – Curvas de igual sonoridade: padrão ISO 226:2003. 29
Figura 4 – Gráfico do limiar de audição. 31
Figura 5 – Gráfico do modelo da banda crítica obtido da Equação (4). 33
Figura 6 – Gráfico da relação entre a frequência (Hz) e a banda crítica na escala *Bark*. . 33
Figura 7 – Esquema do banco de filtros de banda crítica idealizado. 34
Figura 8 – Ilustração dos efeitos de mascaramento de um tom. 36
Figura 9 – Imagem da resposta da membrana basilar para dois tons senoidais. 37
Figura 10 – Ilustração das propriedades do mascaramento temporal do ouvido humano. . 38
Figura 11 – Imagem da anatomia do sistema fonador humano. 39
Figura 12 – (a): Pronúncia da palavra “sino”, (b): Ampliação do segmento não-vocal da palavra “sino”, (c): Ampliação do segmento vocal da palavra “sino”, (d): espectro em frequência para o item (b), (e): espectro em frequência para o item (c). 42
Figura 13 – Ilustração do modelo fonte filtro para o processo de geração de voz. 43
Figura 14 – Ilustração do modelo preditivo linear para o processo de geração da voz. . . 45
Figura 15 – Gráfico dos espectros LPC e FFT para um sinal de voz. 46
Figura 16 – Ilustração da predição de uma amostra $x(m)$ a partir de P amostras passadas. 46
Figura 17 – Fluxograma do algoritmo de Durbin. 51
Figura 18 – Ilustração do efeito, no domínio da frequência, da amostragem no domínio do tempo. (a) Espectro do sinal original. (b) Espectro do sinal amostrado quando $\Omega_s > 2\Omega_N$. (c) Espectro do sinal amostrado quando $\Omega_s < 2\Omega_N$. . . . 56
Figura 19 – Gráfico das características de entrada-saída de um quantizador uniforme. . . 58
Figura 20 – Gráfico da função de compressão de um quantizador não-uniforme típico. Δ_1, Δ_2 e Δ_3 são os diferentes passos de quantização. 60
Figura 21 – Diagrama de blocos simplificado do *vocoder* proposto. 73
Figura 22 – Imagem da interface gráfica do codificador. 74
Figura 23 – Imagem da interface gráfica do decodificador. 76
Figura 24 – Ilustração da forma das janelas citadas. 79
Figura 25 – Gráfico da resposta em frequência das funções janela descritas: (a) Retangular, (b) Hamming, (c) Hanning, (d) Kaiser $\beta = 7.8$. 80
Figura 26 – Diagrama de blocos detalhado do codificador proposto. 81
Figura 27 – Ilustração da configuração de um quadro de 20 ms do formato binário *voz* . 85

Figura 28 – Diagrama de blocos detalhado do estágio de síntese (decodificador). 86
Figura 29 – Ilustração da curva de distribuição Beta para alguns valores de α e β. 87
Figura 30 – Representação do espectro obtido pela FFT de um quadro teste de voz, para três situações distintas: (a) Espectro do sinal original, (b) Espectro simplificado pelo MPFO, (c) Espectro anterior preenchido via distribuição beta. 89
Figura 31 – Ilustração dos tipos de RAL 96
Figura 32 – Ilustração da estrutura básica do sistema de VAL. 97
Figura 33 – Ilustração da estrutura básica do sistema de IAL. 97
Figura 34 – Modelo de um HMM com dois estados (topologia esquerda-direita). 105
Figura 35 – Diagrama de blocos simplificado do sistema de RAL proposto. 110
Figura 36 – Ilustração da interface gráfica do software de RAL proposto. 111
Figura 37 – Imagem do gravador de som do Windows®. 112
Figura 38 – Ilustração da interface gráfica do gerador de padrões de voz. 112
Figura 39 – Representação do espectro de frequência de um quadro de voz, antes e depois do processo de mascaramento auditivo. 117
Figura 40 – Diagrama de blocos do sistema de identificação proposto. 119

LISTA DE TABELAS

Tabela 1 – Exemplos de situações sonoras. 29
Tabela 2 – Banco de filtros idealizado. 34
Tabela 3 – Escala MOS. 65
Tabela 4 – Taxa de bits e pontuação MOS dos codificadores apresentados. 71
Tabela 5 – Algoritmos do sistema de codificação de voz. 74
Tabela 6 – Número de frequências estimadas pela DFT de comprimento 160 em cada oitava do espectro vocal. 83
Tabela 7 – Alocação dos bits para um quadro de voz de 20 ms. 84
Tabela 8 – Resultado do teste ACR para as 4 variações do *vocoder* proposto. 91
Tabela 9 – Taxa de bits e pontuação MOS dos vocoders propostos e dos codificadores apresentados. 92
Tabela 10 – Algoritmos do sistema de RAL proposto. 110
Tabela 11 – Número de frequências estimadas pela DFT de comprimento 160 em cada oitava do espectro vocal. 116
Tabela 12 – Resultado dos testes para o reconhecimento de locutor dependente de texto, frase “O prazo tá terminando”. 120
Tabela 13 – Resultado dos testes para o reconhecimento de locutor dependente de texto, frase “Amanhã ligo de novo”. 121
Tabela 14 – Resultado dos testes para o reconhecimento de locutor independente de texto. 121
Tabela 15 – Comparação com o estado da arte. 122

LISTA DE ABREVIATURAS E SIGLAS

ACR	Classificação por Categoria Absoluta (*Absolute Category Rating*)
AM	Método de Autocorrelação (*Autocorrelation Method*)
BC	Bandas Críticas
bps	bits por segundo
BSD	*Bark Spectral Distance*
CELP	Codificador com Predição Linear Excitada a Código (*Coded-Excited Linear Prediction*)
CM	Método de Covariância (*Covariance Method*)
CS-ACELP	Codificador Algébrico de Estrutura Conjugada com Predição Linear (*Conjugate Structure-Algebraic Code Excited Linear Prediction*)
dB	decibel
DCR	Classificação por Categoria de Degradação (*Degradation Category Rating*)
DFT	Transformada Discreta de Fourier (*Discrete Fourier Transform*)
DMOS	Pontuação de Opinião Média de Degradação (*Degradation Mean Opinion Score*)
DSI	Interpolação Digital de Voz (*Digital Speech Interpolation*)
DTW	*Dynamic Time Warping*
EMQ	Erro Médio Quadrático
ERB	Largura de Banda Equivalente Retangular (*Equivalent Rectangular Band*)
FA	Falsa Aceitação
FIR	Resposta ao Impulso Finita (*Finite Impulse Response*)
FCC	Coeficientes Cepstrais (*Frequency Cepstral Coefficients*)
FFT	Transformada Rápida de Fourier (*Fast Fourier Transform*)
FR	Falsa Rejeição
GMM	Modelos de Misturas Gaussianas (*Gaussian Mixtures Models*)

HMM	Modelos Ocultos de Markov (*Hidden Markov Models*)
IAL	Identificação Automática de Locutor
IFFT	Transformada Rápida Inversa de Fourier (*Inverse Fast Fourier Transform*)
ITU	União Internacional de Telecomunicação (*International Telecommunication Union*)
LAA	Limiar Absoluto de Audição
LBC	Largura de Banda Crítica
LD-CELP	Codificador de Baixo Atraso com Predição Linear Excitada a Código (*Low-Delay Coded-Excited Linear Prediction*)
LIT	Linear Invariante no Tempo
LPC	Codificação Preditiva Linear (*Linear Predictive Coding*)
MFCC	Coeficientes Mel-Cepstrais (*Mel-Frequency Cepstral Coefficients*)
MFLOPS	Milhões de Operações em Ponto Flutuante por Segundo (*Million Floating point Operations Per Second*)
MIPS	Milhões de Instruções por Segundo (*Million Instructions Per Second*)
MOS	Pontuação de Opinião Média (*Mean Opinion Score*)
MPFO	Mascaramento Pleno em Frequência por Oitava
NMR	Relação Ruído Limiar de Mascaramento (*Noise-to[minimum of]-Mask Ratio*)
PCM	Modulação por Largura de Pulsos (*Pulse Code Modulation*)
PDS	Processamento Digital de Sinais
PDSV	Processamento Digital de Sinais de Voz
PESQ	*Perceptual Evaluation of Speech Quality*
PSQM	*Perceptual Speech Quality Measure*
QV	Quantização Vetorial
RAL	Reconhecimento Automático de Locutor
RNA	Rede Neural Artificial

SF	Função de Espalhamento (*Spread Function*)
SNR	Relação Sinal Ruído (*Signal-to-Noise Ratio*)
SNRR	Relação Sinal Ruído de Segmentação (*Signal-to-Noise Segmentation*)
SMR	Relação Sinal Mascarador (*Signal-to-Mask Ratio*)
SVM	*Support Vector Machine*
SPL	Nível de Pressão Sonora (*Sound Pressure Level*)
VAD	Detector de Voz Ativa (*Voice Activity Detection*)
VAL	Verificação Automática de Locutor
VBR	Taxa de Bits Variável (*Variable Bit Rate*)

LISTA DE SÍMBOLOS

Γ	Letra grega Gama
Λ	Lambda
ζ	Letra grega minúscula zeta
$\in$	Pertence

SUMÁRIO

1 INTRODUÇÃO 19
1.1 Objetivos 20
1.2 Estrutura 21

2 O SOM E SUA PERCEPÇÃO PELOS SERES HUMANOS 23
2.1 Definição de Som 23
2.2 Características do Som 23
2.2.1 Frequência 24
2.2.2 Altura 24
2.2.3 Volume 24
2.2.4 Timbre 25
2.3 Percepção Humana do Som 25
2.3.1 A Anatomia do Ouvido Humano 25
2.3.1.1 Ouvido Externo 26
2.3.1.2 Ouvido Médio 27
2.3.1.3 Ouvido Interno 27
2.3.2 Largura de Banda, Sensitividade e Faixa Dinâmica da Audição 28
2.3.2.1 Limiar Absoluto da Audição 30
2.3.3 Bandas Críticas da Audição 32
2.3.4 Mascaramento 35
2.3.4.1 Mascaramento em Frequência 35
2.3.4.2 Mascaramento no Domínio do Tempo 37

3 A VOZ 39
3.1 Mecanismo de Produção da Voz 39
3.2 Classificação dos Sons Produzidos pelo Sistema de Fonador Humano 41
3.3 Modelamento do Sistema de Produção da Voz 43
3.4 Modelamento Preditivo Linear para Sinais de Voz 44
3.4.1 Minimização do Erro 47
3.4.2 Método da Autocorrelação 49
3.4.3 Método da Covariância 52

4 CODIFICAÇÃO DIGITAL DA VOZ 55
4.1 Digitalização da Voz 55
4.1.1 Amostragem do Sinal 55

4.1.2 Quantização . . . 57
4.1.2.1 Quantização Uniforme . . . 57
4.1.2.2 Quantização Não-uniforme . . . 59
4.2 Codificação Paramétrica da Voz . . . 61
4.3 Atributos dos Codificadores de Voz . . . 62
4.3.1 Taxa de Bits . . . 63
4.3.2 Qualidade do Sinal de Saída . . . 63
4.3.2.1 Métodos Objetivos . . . 64
4.3.2.2 Métodos Subjetivos . . . 65
4.3.2.2.1 **ACR (Absolute Category Rating)** . . . 65
4.3.2.2.2 **DCR (Degradation Category Rating)** . . . 65
4.3.3 Complexidade dos Algoritmos e Quantidade de Memória Necessária 66
4.3.4 Atraso . . . 67
4.3.5 Sensibilidade ao Erro . . . 67
4.4 Técnicas de Codificação de Voz . . . 67
4.4.1 G.711 - PCM . . . 68
4.4.2 G.722 - SB-ADPCM . . . 68
4.4.3 G.726 - ADPCM . . . 69
4.4.4 G.728 - LD-CELP . . . 69
4.4.5 G.729 - CS-ACELP . . . 70
4.5 Sumário dos Codificadores . . . 71

5 UM NOVO PADRÃO DE CODIFICAÇÃO DE VOZ . . . 72
5.1 Introdução . . . 72
5.2 Visão Geral do Sistema . . . 72
5.2.1 Implementação do Sistema . . . 73
5.3 Pré-Processamento do Sinal . . . 77
5.3.1 Segmentação da Voz . . . 77
5.3.2 Janelamento . . . 77
5.4 Análise da Voz pela Técnica do MPFO . . . 81
5.4.1 Características Psicoacústicas do Sistema Auditivo Humano . . . 82
5.4.1.1 Insensibilidade à Fase do Som . . . 82
5.4.2 Simplificação do Espectro Via MPFO . . . 82
5.4.3 Quantização e Codificação dos Sinais de Voz . . . 83
5.4.4 Formato Binário *voz* . . . 85
5.5 Síntese da Voz . . . 85
5.5.1 Preenchimento Espectral via Distribuição Beta . . . 86
5.5.1.1 Metodologia Empregada . . . 87
5.6 Simulações e Classificações da Qualidade de Voz . . . 90

6 RECONHECIMENTO AUTOMÁTICO DE LOCUTOR 93
6.1 Introdução 93
6.2 Tipos de RAL 94
6.2.1 VAL 94
6.2.2 IAL 95
6.3 Estrutura Básica dos Sistemas de RAL 96
6.3.1 Extração das Características 98
6.3.2 Parâmetros Extraídos do Sinal de Voz 99
6.3.2.1 Banco de Filtros 99
6.3.2.2 Energia de Tempo Curto 99
6.3.2.3 Taxa de Cruzamento pelo Zero 99
6.3.2.4 Coeficientes Cepstrais 100
6.3.2.5 Coeficientes Mel-Cepstrais 101
6.3.2.6 Coeficientes LPC 101
6.3.3 Modelamento 101
6.3.4 Modelos Ocultos de Markov - HMM 102
6.3.4.1 Introdução 102
6.3.4.2 Descrição do Modelo 102
6.3.4.3 Simplificações da Teoria do HMM 105
6.3.4.4 Treinamento 105
6.3.4.5 Reconhecimento 107

7 SISTEMA PROPOSTO DE RAL 109
7.1 Introdução 109
7.2 Visão Geral do Sistema 109
7.2.1 Implementação do Sistema 110
7.3 Aquisição dos Sinais de Voz 113
7.4 Pré-Processamento dos Sinais de Voz 113
7.4.1 Pré-Ênfase 114
7.4.2 Detecção de Pontos Extremos (*Endpoints)* 114
7.4.3 Segmentação dos Dados em Quadros e Janelamento 115
7.5 Geração do Padrão do Locutor 115
7.5.1 Extração das Características dos Quadros de Voz 115
7.5.2 Obtenção do Padrão dos Locutores 118
7.6 Comparação dos Padrões de Voz 118
7.7 Testes e Resultados Obtidos 120
7.7.1 IAL Dependente de Texto 120
7.7.2 IAL Independente de Texto 121
7.7.3 Comparação com o Estado da Arte 122

8 DISCUSSÕES E CONCLUSÕES 124
8.1 Síntese das Contribuições Pessoais 124
8.2 Sistema de Codificação de Voz Proposto 124
8.3 Sistema de Reconhecimento Automático de Locutor Proposto . . 125
8.4 Sugestões e Trabalhos Futuros 126
8.4.1 Para o *Vocoder* 126
8.4.2 Para a Identificação Automática de Locutor 127

REFERÊNCIAS 128

APÊNDICES 133

APÊNDICE A – ARTIGOS PUBLICADOS 134

ANEXOS 148

ANEXO A – CÓDIGOS FONTE (*VOCODER*) 149

ANEXO B – CÓDIGOS FONTE (*RAL*) 165

ANEXO C – MANUAL PARA GERAÇÃO DO PADRÃO DE VOZ . . 257

ANEXO D – CD 260

1 INTRODUÇÃO

Nos dias atuais, os frequentes avanços na eletrônica e na informática estão causando um explosivo crescimento no uso de máquinas para o processamento de informação. Em meio a toda essa evolução, faz-se necessário uma forma eficiente de transferência de informação entre homem e máquina, em ambos os sentidos. A possibilidade de realizar essa comunicação através da voz torna essa interação mais ágil e produtiva, uma vez que, além de ser a forma mais simples, natural e universal de comunicação do ser humano, permite ao usuário uma maior liberdade para a realização de outras tarefas, oferecendo inúmeras vantagens, como velocidade de transmissão, mobilidade e acesso remoto à informação (RABINER; SCHAFER, 1978).

O interesse pela interação homem-máquina a partir da fala tem aumentado consideravelmente, dando origem a uma demanda muito grande por sistemas capazes de representar a voz de uma maneira eficiente, reconhecer o que está sendo dito ou quem se está falando, ou ainda responder ao que está sendo solicitado (MINKER; BENNACEF, 2004).

Historicamente, pode-se considerar que a barreira da distância na comunicação falada foi quebrada em 1876, quando da invenção do telefone por Alexander Graham Bell (SCHROEDER, 1981). Desde então, a importância da comunicação à distância na sociedade não tem parado de crescer, sendo essencial para difusão de informação entre pessoas e países. A introdução da comunicação digital na década de 70 iniciou uma nova era na comunicação, tendo a representação eficiente de sinais de voz tornado-se uma área de grande importância. Devido à existência de várias questões que ainda não foram solucionadas, muito estudo ainda vem sendo realizado na área de Processamento Digital de Sinais de Voz (PDSV). Muitos deles são feitos visando a redução da taxa de bits, sendo esse um parâmetro muito importante na definição da largura de banda do canal de transmissão. A necessidade desta redução permanece mesmo com o aumento da largura de banda desses canais de transmissão, possibilitando a transmissão de um número maior de sinais no mesmo canal ou permitindo a utilização em canais ruidosos (SPANIAS; PAINTER; ATTI, 2007). Da mesma forma, o advento das tecnologias multimídia e a necessidade de armazenamento de grandes quantidades de informação para utilização posterior, exige redução da taxa de bits, já que esta determina o espaço requerido na unidade de armazenamento (MIRANDA, 2002).

Codificação pode ser entendido como a representação eficiente do sinal com vista à sua transmissão ou armazenamento, mas mantendo uma qualidade aceitável, exigida pelas eventuais aplicações. Os sistemas responsáveis por realizar compressões em sinais de voz são os codificadores de voz ou *vocoders*. Os mais eficientes, ao se aproveitar de algumas propriedades psicoacústicas do ouvido humano e de características redundantes do sinal de voz, conseguem trabalhar com taxa de 2 kbits/s (MIRANDA, 1996).

Outras áreas de PDSV também são bastante investigadas. Uma bastante relevante, cujo estudo foi inaugurado há mais de 30 anos e que ganhou força com a inclusão dos modelos ocultos de Markov (HMM, do inglês *Hidden Markov Models*), é o Reconhecimento Automático de Locutor (RAL) (PARANAGUÁ, 1997). Este sistema consiste de uma técnica fundamental para os mecanismos de segurança, que podem analisar um segmento de voz e reconhecer a pessoa que o produziu. O RAL é dividido em Verificação Automática de Locutor (VAL) e Identificação Automática de Locutor (IAL) (CAMPBELL, 1997). A VAL é o processo de aceitar ou rejeitar a identidade pretensa de um locutor teste. A IAL objetiva identificar o autor de uma dada elocução teste baseado em um conjunto de possíveis locutores previamente treinados. O RAL é uma área da Inteligência Artificial em que o desempenho da máquina pode superar o desempenho de seres humanos: usando curtas locuções testes e um grande número de locutores, a precisão da VAL ou da IAL, frequentemente, excede àquela dos seres humanos. Verifica-se isto, especialmente, para locutores não familiares, cujo "tempo de treinamento" requerido para assimilação de uma nova voz é maior quando comparado ao tempo necessitado pela máquina.

Para efetuar o reconhecimento em si, os principais métodos utilizados são os baseados em HMMs (TISHBY, 1991), Modelos de Mistura Gaussiana (GMMs) (REYNOLDS; ROSE, 1995), *Dynamic Time Warping (DTW)* (CAMPBELL, 1997) e Redes Neurais Artificiais (RNAs) (FARRELL; MAMMONE; ASSALEH, 1994). Nas que utilizam HMMs, a precisão do reconhecimento pode chegar a quase 99%, mas essa taxa é conseguida sob altas complexidade e demanda computacionais (PEACOCKE; GRAF, 1990).

Algumas pesquisas na área de reconhecimento de locutor visam reduzir a complexidade computacional de métodos já existentes e que, invariavelmente, requerem grande carga computacional para o processamento. O trabalho publicado em (DAN et al., 2008), baseado no LS-SVM (Least Square Support Vector Machine), transforma um problema de programação quadrática, do convencional Support Vector Machine (SVM), num problema de programação linear, reduzindo assim a complexidade computacional. Outras publicações procuram aprimorar o desempenho dos métodos de reconhecimento em ambientes ruidosos, como em (WANG et al., 2007) (SHAO; WANG, 2006).

Neste trabalho, os interesses são nas áreas de codificação de voz, sugerindo um novo tipo de *vocoder* e introduzindo um inovador formato binário de armazenamento de sinais de voz, e no RAL, apresentando uma nova técnica para as identificações dos locutores.

1.1 Objetivos

Esta dissertação possui dois objetivos principais. O primeiro é o desenvolvimento de um novo sistema de codificação de voz (*vocoder*) que seja útil para economizar largura de banda em aplicações requerendo inteligibilidade, bem como para o monitoramento de conversas de voz de longa duração, decorrentes de espionagem autorizada. Este sistema utiliza uma técnica

definida por Mascaramento Pleno em Frequência por Oitava (MPFO), em adição a um método de preenchimento espectral via distribuição beta de probabilidade. Tal abordagem permitiu a criação de um novo formato de codificação para sinais de voz: o formato binário *voz*. O segundo objetivo é apresentar uma nova abordagem para o RAL, motivado a partir da síntese do *vocoder* proposto e que herda parte da sua teoria. O sistema oferece um compromisso entre complexidade computacional e taxa de identificações corretas, podendo ser atrativo para aplicações em sistemas embarcados.

1.2 Estrutura

A dissertação está dividida em oito capítulos. O *Capítulo 1, Introdução,* contextualiza a pesquisa e apresenta seus objetivos. O *Capítulo 2, O Som e sua Percepção pelos Seres Humanos*, aborda características relacionadas ao som e as formas pelas quais o sistema auditivo humano percebe a sua presença. O *Capítulo 3, A Voz,* apresenta uma descrição do sinal de voz, desde sua geração em forma de ar pelos pulmões até sua saída, de diversas maneiras, pela boca. Exibe, também, um modelamento matemático para o sistema de produção da voz. Os Capítulos 4 e 5 são pertinentes ao sistema de codificação de voz proposto na dissertação. O *Capítulo 4, Codificação Digital da Voz,* apresenta o processo de digitalização da voz, introduzindo a codificação paramétrica e especificando alguns codificadores existentes. Também são expostas algumas técnicas de codificação da voz. O *Capítulo 5, Um Novo Padrão de Codificação de Voz*, diz respeito ao desenvolvimento do sistema de codificação proposto pela dissertação, introduzindo as técnicas utilizadas para sua implementação e apresentando resultados de testes realizados. Os Capítulo 6 e 7 são referentes ao sistema de RAL proposto no trabalho. O *Capítulo 6, Reconhecimento Automático de Locutor,* descreve o funcionamento dos sistemas de RAL e detalha alguns de seus tipos. Cita, também, algumas possíveis aplicações para eles. O *Capítulo 7, Sistema de RAL Proposto,* detalha o sistema de RAL desenvolvido e exibe resultado de simulações envolvendo procedimentos padrão. O *Capítulo 8, Discussões e Conclusões,* apresenta as conclusões gerais deste trabalho e algumas sugestões para trabalhos futuros. A dissertação, também, é composta de um apêndice e quatro anexos. O *Apêndice A, Artigos Publicados,* lista os dois artigos publicados temas desta dissertação, juntamente com a cópia de cada um deles. O *Anexo A, Códigos Fonte (vocoder),* e o *Anexo B, Códigos Fonte (RAL),* exibe os códigos fonte, em MATLAB®, dos algoritmos utilizados na implementação dos sistemas propostos no trabalho. O *Anexo C, Manual para Geração do Padrão de Voz,* contém procedimentos necessários para obtenção dos padrões de voz dos locutores a partir de um número de elocuções de treinamento definido pelo usuário do sistema. O *Anexo D* inclui um CD contendo os seguintes itens:

- Versão em *pdf* desta dissertação;
- Algoritmos do *vocoder* proposto;

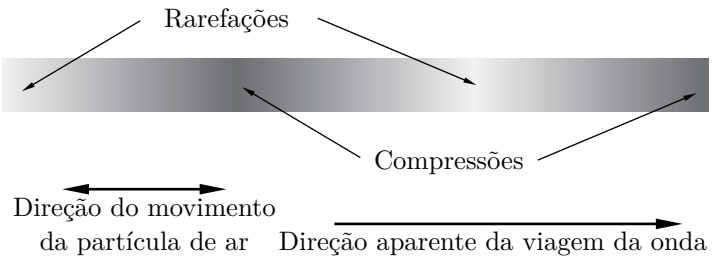
- Algoritmos do RAL proposto;
- Arquivos de voz utilizados nos testes.

2 O SOM E SUA PERCEPÇÃO PELOS SERES HUMANOS

2.1 Definição de Som

O som é definido como um fenômeno acústico. É uma vibração propagante em um meio na frequência de 20 Hz a 20 kHz. Essas vibrações geram mudanças na pressão, resultando numa série de compressões e rarefações longitudinais do meio, criando regiões mais densas e outras menos densas, que, ao atingirem o ouvido dos seres vivos, fazem vibrar os seus tímpanos (RUMSEY; MCCORMICK, 2006). Nervos que ligam o tímpano ao cérebro interpretam essas vibrações e o resultado é a sensação auditiva (BRANCO NETO, 2000). Fontes sonoras simples, como colunas de ar, cordas, membranas, etc podem gerar essas vibrações. A Figura 1 ilustra a onda longitudinal criada pela sucessão de compressões e rarefações das partículas de ar.

Figura 1 – Imagem da onda longitudinal criada com as sucessões de rarefações e compressões das partículas de ar.



Fonte: Adaptado de Rumsey e McCormick (2006).

Como uma onda mecânica, o som necessita de um meio (elástico, viscoso) dotado de forças internas para se propagar — impossibilitando a propagação no vácuo — e sua velocidade de propagação depende desse meio. No ar (fluido de baixa viscosidade) o som se propaga com uma velocidade de aproximadamente 340 m/s (BARBOSA, 2008).

2.2 Características do Som

As principais características do som são:

- Frequência
- Altura ou *Pitch*

- Volume
- Timbre

2.2.1 Frequência

A taxa na qual a fonte sonora produz vibração em um meio de propagação, num dado intervalo de tempo, é referida como a frequência da onda sonora. Ela é medida em hertz (Hz) em homenagem ao físico alemão Heinrich Rudolf Hertz (1857–1894) (BARBOSA, 2008). A frequência está diretamente ligada à altura do som. Quanto maior for a frequência de uma onda, maior será a sua altura e, consequentemente, mais aguda será a sua sonoridade (BRANCO NETO, 2000).

2.2.2 Altura

Altura é a sensação auditiva que cada pessoa possui em relação às frequências de um determinado som, sendo restrita à interpretação individual de cada uma. É o que determina se o som é alto ou baixo. Esta percepção auditiva em relação à frequência da componente fundamental de um som é referida como *pitch*, termo também utilizado para se referir a própria frequência fundamental da onda sonora (SMITH, 2003).

Um som com um alto *pitch* corresponde a um som com uma alta frequência fundamental, consequentemente a um som mais agudo (mais alto). Em contrapartida, um som com um baixo *pitch* corresponde a um som com uma baixa frequência fundamental e, por conseguinte, a um som mais grave (mais baixo). As frequências múltiplas da frequência fundamental são nomeadas harmônicos do sinal.

É comum se confundir altura com intensidade. Por exemplo, após se tocar uma corda no violão, suavemente, ouve-se um som; ao tocá-la, novamente, com uma força maior, não será ouvido um som mais alto, será ouvido um som mais forte, intenso. Para se ouvir um som mais alto, basta tocar uma corda que soe em uma frequência maior, independente da força aplicada a ela.

2.2.3 Volume

O volume é uma medida da intensidade de um som (SMITH, 2003). Está relacionado com a quantidade de compressão e rarefação do ar que resulta da vibração da onda sonora (RUMSEY; MCCORMICK, 2006). É medido em unidade de fons. Um fon corresponde a 1 dB do nível de pressão sonora acima do limiar de audição nominal (o nível de pressão sonora de 20 μPa).

2.2.4 Timbre

O timbre é determinado pelo conteúdo harmônico do sinal (SMITH, 2003) — a relação entre o nível da fundamental, os níveis dos harmônicos e suas evoluções no tempo — que surge da conformação da onda. É uma característica importante por diferenciar a voz das pessoas e o som de diferentes tipos de instrumentos. Esta característica deve-se ao fato de que uma onda sonora, em geral, não é produzida por apenas uma frequência fundamental, mas por uma composição da frequência fundamental associada com os seus harmônicos.

Uma mesma nota musical tocada por dois instrumentos distintos apresenta um mesmo *pitch*. Entretanto, as notas não soam identicamente porque seus conteúdos harmônicos são diferentes (as amplitudes dos seus harmônicos são diferentes). Esses sons produzidos tem, portanto, timbres diferentes. Em geral, uma dada forma de onda particular possui apenas um único timbre, mas um timbre particular pode ser produzido por um número infinito de possíveis formas de onda.

2.3 Percepção Humana do Som

A percepção humana do som é um problema complexo. O sistema auditivo humano não capta, não interpreta, nem "sente" todos os sons da mesma maneira (COOK, 2002). A psicoacústica, um ramo da psicofísica, busca compreender todo este processo, estudando a relação entre as medidas físicas dos sons e as formas com que são interpretadas pelo cérebro humano (BRANCO NETO, 2000).

Vários métodos de processamento de sinais de áudio utilizam-se de modelos da psicoacústica e da sensitividade da audição humana (VASEGHI, 2007) para projetar sistemas com taxas de codificação mais eficientes, todavia, sem sacrificar, significativamente, a qualidade do som (BHARIKTAR; KYRIAKAKIS, 2006). Algumas propriedades da percepção auditiva humana (e.g., seletividade em frequências, insensitividade à fase e mascaramento em tempo e frequência) são exploradas para projetar sistemas mais eficientes. Algumas delas foram relevantes para a implementação desse trabalho, em especial o mascaramento psicoacústico auditivo humano.

A fim de se entender algumas propriedades psicoacústicas sentidas pelos seres humanos, as quais serão detalhadas mais adiante, faz-se necessário conhecer a estrutura fisiológica do ouvido humano e compreender o seu processo de audição.

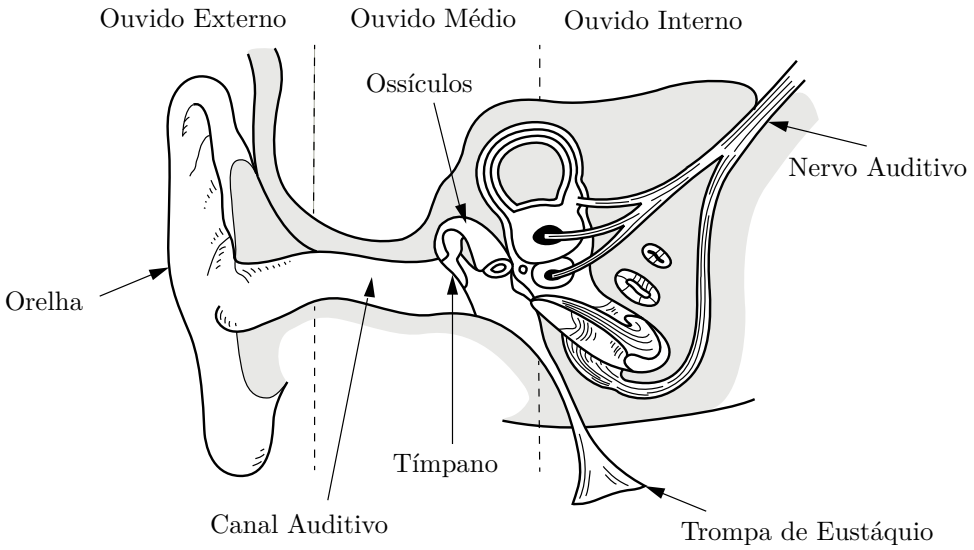
2.3.1 A Anatomia do Ouvido Humano

O ouvido é o órgão usado para detectar as ondas sonoras. Ele funciona basicamente como um transdutor, convertendo as variações de pressão do ar, que chega ao tímpano, em impulsos elétricos transmitidos para o cérebro e interpretados como som (VASEGHI, 2007) .

A Figura 2 ilustra a anatomia do ouvido humano, compreendendo suas três partes básicas:

1. **ouvido externo**: responsável por captar as vibrações do ar e direcioná-las ao tímpano.
2. **ouvido médio**: responsável por converter as vibrações de ar, que se chocam ao tímpano, em vibrações mecânicas dos ossículos, as quais irão colidir com a região mais interna do ouvido.
3. **ouvido interno**: responsável por transformar as vibrações mecânicas do ouvido médio em vibrações hidráulicas dos tubos cheios de fluido da cóclea.

Figura 2 – Imagem da anatomia do ouvido humano.



Fonte: Adaptado de Watkinson (2001).

2.3.1.1 Ouvido Externo

O ouvido externo é constituído pelo pavilhão auditivo ou orelha, pelo canal auditivo — um tubo de cerca de 0,5 cm de diâmetro que se estende por cerca de 2,5 cm para dentro do ouvido — e pela camada exterior ao tímpano. A orelha, a estrutura visível do ouvido, é a responsável por distinguir a direção da fonte sonora, uma vez que ao possuir uma estrutura refletiva, consegue amortecer determinadas componentes do som e amplificar outras, em função da localização da mesma. Por outro lado, a mesma não desempenha qualquer função no reconhecimento do som no plano horizontal (PENHA, 1996).

A presença de um ouvido em cada lado da cabeça permite a audição estereofônica e a habilidade de encontrar a direção de chegada do som pela análise da intensidade relativa e fase (atraso) das ondas sonoras que alcançam cada ouvido (VASEGHI, 2007).

Pelo canal auditivo penetra o som vindo do exterior, o qual é conduzido até à membrana timpânica ou tímpano, que possui uma ressonância natural de aproximadamente 3400 Hz. Esta estimativa é encontrada pela seguinte expressão:

$$f = \frac{c}{4L}, \tag{1}$$

em que c é a velocidade de propagação e L é o comportamento do canal (PENHA, 1996). Neste caso, pode-se adotar c = 340 m/s e L = 2,5 cm.

2.3.1.2 Ouvido Médio

Estrutura do ouvido que serve, basicamente, como um casador de impedância e como um amplificador (VASEGHI, 2007). É constituído de três pequenos ossos interconectados que ligam a membrana timpânica ao ouvido interno por meio da janela oval (uma outra membrana). Os sons que chegam ao tímpano induzem-o a vibrar e essas vibrações são transmitidas através do ouvido médio pelos três ossículos (martelo, bigorna e estribo) para o ouvido interno. O tímpano age como um transdutor acústico-mecânico.

Aos ossículos estão acoplados músculos tensores, que podem atenuar a transmissão de vibrações, especialmente em baixas frequências, em que a atenuação pode atingir 30 dB. Estes músculos proporcionam uma compressão da faixa dinâmica da percepção sonora, além de agirem como proteção reflexa contra sons de alta intensidade. A presença desses ossículos melhora a propagação do som, reduzindo a quantidade de reflexões por meio do princípio do casamento de impedância. A maior parte desse processo de conversão de impedância resulta da diferença, em área, entre o tímpano, cuja área é de cerca de 60 mm^2, e a janela oval, cuja área é de aproximadamente 4 mm^2. Uma vez que a pressão é inversamente proporcional à área, há um aumento da pressão da onda sonora por quase 15 vezes (SMITH, 2003).

Uma outra estrutura importante da região intermediária do ouvido é o tubo de Eustáquio, que conecta o ouvido médio à nasofaringe da garganta. Esse tubo se abre com o ato de engolir ou de tossir para equalizar a pressão entre o ouvido médio e a pressão ambiente estabelecida na garganta (VASEGHI, 2007).

2.3.1.3 Ouvido Interno

O ouvido interno é o principal órgão da audição. É nele que se encontram as estruturas que permitem ao seres humanos identificarem e caracterizarem os sons e suas características fundamentais: frequência, intensidade e timbre. Tem como função transformar as vibrações mecânicas dos ossículos do ouvido médio em um padrão de ondas viajantes, que atingem a membrana basilar e produzem descargas neurais nas células capilares do órgão de Corti (VASEGHI, 2007).

O ouvido interno é constituído pela cóclea, por canais semicirculares e pelo nervo auditivo. A cóclea é um órgão em forma de caramujo, que, se esticado, teria comprimento de 2 a 3 cm. Além de estar preenchida de fluido, a superfície interna da cóclea, conhecida como membrana basilar, é recoberta por cerca de 20.000 células nervosas, que se sensibilizam de forma diferente, dependendo da intensidade e da frequência do som. Tal fato nos faz perceber, de forma diferente, dois sons de igual intensidade, mas de frequências distintas e vice-versa.

À medida que uma onda de compressão se move da interface entre o ouvido médio ao ouvido interno através da cóclea, as células nervosas, em forma de cílios, são empurradas e entram em movimento. Cada uma destas células possui uma sensibilidade natural para uma dada frequência de vibração. Quando a frequência da onda de compressão casa com a frequência natural da célula nervosa, a célula ressoa com uma grande amplitude de vibração. Esta vibração ressonante induz a célula a liberar um impulso elétrico, que passa ao longo do nervo auditivo para dentro do cérebro. A cóclea opera, então, como um analisador de espectro ou como um banco de filtros seletivos distribuídos (VASEGHI, 2007).

O ouvido normal é, assim, capaz de distinguir sons com frequências entre 20 Hz e 20 kHz, embora a resposta para altas frequências diminua acentuadamente com a idade (50% das pessoas não ouvem acima de 15 kHz).

2.3.2 Largura de Banda, Sensitividade e Faixa Dinâmica da Audição

O ouvido apresenta resposta perceptual aproximadamente logarítmica, tanto na distinção de frequências como na de intensidade sonora. É conveniente, então, medir o nível de pressão sonora em escala logarítmica, sendo definida a unidade "dB SPL" (do inglês "*Sound Pressure Level*") como:

$$SPL \triangleq 20 \cdot \log \frac{p}{p_0} \qquad (dB\ SPL), \tag{2}$$

em que p_0 é o limiar inferior de audibilidade para um ouvido normal. A quantidade é associada a uma amplitude de pressão sonora de 20 μPa r.m.s. (2.10^{-5} N/m^2) na frequência de 1 kHz, com p_0 = 20 μPa. O limiar de desconforto, aquele em que o som passa a ser doloroso e, potencialmente, prejudicial à saúde é atingido com pressões sonoras 1.000.000 de vezes maiores. À título de exemplo, a Tabela 1 lista as intensidades sonoras de alguns tipos de sons.

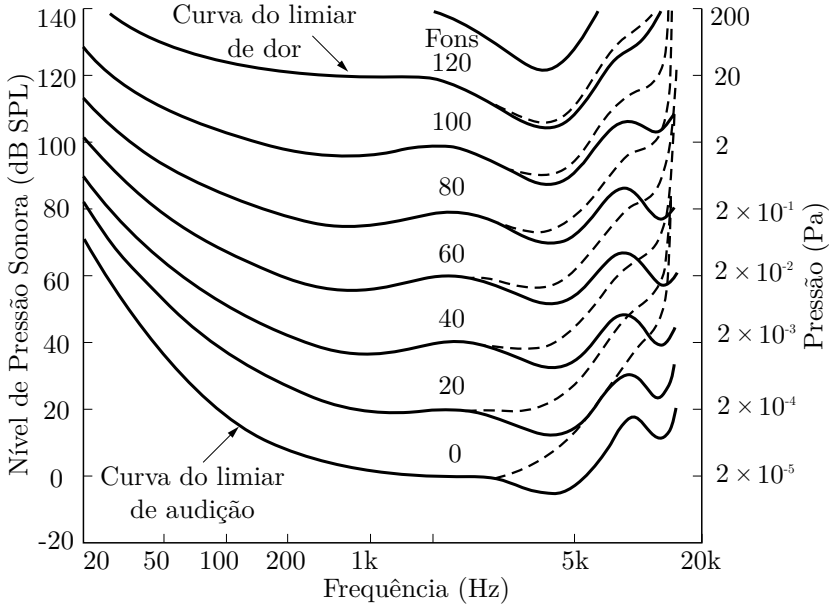
Tabela 1 – Exemplos de situações sonoras.

	dB SPL		**dB SPL**
Avião a jato	155	Restaurante	60
Limiar de dor	140	Residência urbana	40
Limiar de desconforto	$120 = 1W/m^2$	Casa de campo	30
Orquestra fortíssimo	110	Orquestra pianíssimo	30
Rebitadeira	100	Estúdio de gravação	20
Fábrica	78	Folhagens na brisa	10
Tráfego pesado	68	Limiar de audição	$0 = 1pW/m^2$
Escritório ruidoso	65	Ruído térmico do ar	-10 dB

Fonte: Produzido pelo autor.

O ouvido é sensível à uma faixa de frequências audível entre 20 Hz e 20 kHz. Dentro deste intervalo, sua resposta em frequência varia acentuadamente devido às formas e propriedades físicas do canal do ouvido e do ouvido médio. O ouvido é menos sensível a vibrações de pressão sonora abaixo de 100 Hz e acima de 10 kHz, e é mais sensível a vibrações do som nas frequências entre 500 Hz e 5 kHz, com sensibilidade máxima em torno de 3 kHz (VASEGHI, 2007).

Figura 3 – Curvas de igual sonoridade: padrão ISO 226:2003.



Fonte: Adaptado de Watkinson (2001).

Na Figura 3, é possível observar o comportamento da sensibilidade do ouvido humano, que é representado em curvas de nível de igual sonoridade. Ao longo de cada uma dessas curvas, as diferentes frequências tem mesma sonoridade percebida que a frequência de referência de 1 kHz. Essas curvas foram, primeiramente, desenvolvidas por Fletcher Munson em 1933 (VASEGHI, 2007). Novas curvas foram, posteriormente, obtidas por Robinson e Dadson em 1956, sendo consideradas mais precisas do que as antecedentes. Elas tornaram-se a base para o padrão ISO 226 até 2003. Desde então, o novo padrão revisado ISO 226:2003 é adotado.

A curva pontilhada inferior da Figura 3 representa o contorno do campo audível mínimo e retrata, na média (entre os ouvintes), o limite inferior absoluto da audição humana em várias frequências (BHARIKTAR; KYRIAKAKIS, 2006). Este nível foi escolhido, já que é, aproximadamente, igual ao limiar médio de audição para o ser humano com percepção auditiva normal em 1 kHz (frequência em que o ouvido humano é significativamente sensível (HOLMES; HOLMES, 2002)). Qualquer som que se encontra nesse limiar de audição (apenas perceptível) é dito ter uma sonoridade de 0 fons. Todos os pontos ao longo de umas das curvas da Figura 3 terão sonoridades iguais, embora claramente um maior "nível de pressão sonoro" seja necessário nos extremos do espectro (RUMSEY; MCCORMICK, 2006). As ondulações na sensibilidade auditiva ocorridas na faixa de 1 a 10 kHz são causadas pela onda estacionária ressonante no canal auditivo (VASEGHI, 2007). A curva pontilhada superior corresponde ao limiar de dor. Se um som encontra-se neste limiar, terá uma sonoridade de aproximadamente 120 fons (RUMSEY; MCCORMICK, 2006). Note que o limiar de audição é cerca de 70 dB SPL em 10 Hz, 0 dB SPL em 1 kHz — frequência de referência —, e cerca de -10 dB SPL em torno de 3 kHz.

Chama-se de faixa dinâmica da audição a diferença entre o limiar de audição (0 dB SPL) e o limiar da dor (120 dB SPL). Dessa forma, a diferença entre o som mais alto e o som mais fraco que os humanos podem ouvir é cerca de 120 dB SPL, que corresponde a uma variação de um milhão na amplitude. As pessoas podem detectar mudanças na amplitude do sinal quando ele é alterado por cerca de 1 dB SPL (12% na amplitude), existindo portando 120 níveis que podem ser percebidos de um mais fraco murmúrio para um mais barulhento trovão (SMITH, 2003).

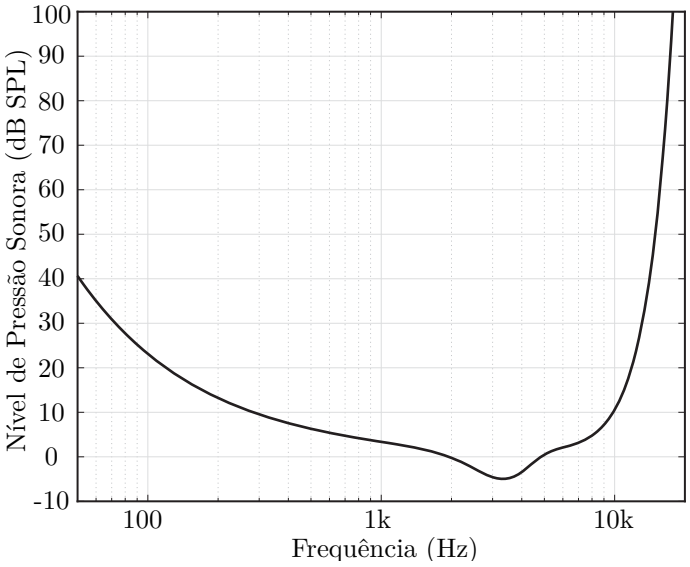
Experimentalmente, determinou-se que as regiões confortáveis para audição de música ou voz abrangem faixas dinâmicas, respectivamente, de 70 dB SPL e 30 dB SPL.

2.3.2.1 Limiar Absoluto da Audição

O limiar absoluto da audição (LAA) caracteriza a quantidade de energia necessária para que um tom puro possa ser detectado por um ouvinte em um ambiente sem ruído.

A Figura 4 ilustra uma curva típica do limiar absoluto, na qual o eixo horizontal indica a frequência em Hz, enquanto que o eixo vertical representa o limiar absoluto em dB SPL, que é associado a uma intensidade de referência de 10^{-12} W/m^2 (uma quantidade padrão para a medida da intensidade do som) (SMITH, 2003).

Figura 4 – Gráfico do limiar de audição.



Fonte: Produzido pelo autor.

A curva do limiar de audição pode ser bem aproximada pela função não linear (VASEGHI, 2007)

$$LAA(f) = 3.64(f/1000)^{-0.8} - 6.5e^{-0.6(f/1000-3.3)^2} + 10^{-3}(f/1000)^4 \qquad (\text{dB SPL}), \quad (3)$$

a qual é representativa de uma pessoa jovem com uma audição apurada, mas que reflete apenas um comportamento médio. A forma atual varia de pessoa para pessoa e é medida experimentalmente soando um tom de uma certa frequência e variando sua intensidade até a pessoa já não perceber o seu efeito. Ao repetir-se as medidas para um grande número de valores de frequência, obtém-se a curva do limiar de audição da Figura 4 (CHU, 2003). Note que a máxima sensitividade da audição (o mínimo valor de LAA) ocorre em torno de 3 e 4 kHz e é cerca de -5 dB SPL. Os níveis de pressão somente detectáveis na frequência de máxima sensitividade da audição não são detectáveis em outras frequências.

Considerando estas propriedades, pode-se tirar vantagem da curva de limiar absoluto no projeto de codificadores de voz, pois:

- Não há necessidade de se considerar quaisquer sinais com uma intensidade inferior ao limiar absoluto, uma vez que eles não tem qualquer impacto sobre a qualidade final do codificador.
- Mais recursos devem ser alocados para a representação dos sinais dentro da faixa de frequência mais sensível, em linhas gerais de 1 a 4 kHz, já que distorções nesta faixa são mais perceptíveis (CHU, 2003).

2.3.3 Bandas Críticas da Audição

Anatomicamente, as bandas críticas (BC) da audição são segmentos da cóclea de cerca de 1.3 mm de comprimento, que agem como filtros passa-banda (VASEGHI, 2007). São representações para um modelo matemático do ouvido humano, que divide o espectro audível em bandas de largura não-uniformes, dentro das quais as características auditivas podem ser consideradas constantes.

O conceito de banda crítica é importante no entendimento da audição, porque ele ajuda a explicar como alguns sinais são "mascarados" na presença de outro (RUMSEY; MCCORMICK, 2006). Este conceito foi baseado em observações experimentais da percepção de sinais de áudio ao longo da membrana basilar da cóclea, lugar onde acontecem as transformações espaço-frequência. Alguns experimentos mostraram que, em qualquer frequência ao longo da cóclea, o ouvido comporta-se como uma série de filtros passa-banda conhecidos como bandas críticas. As larguras de bandas críticas são não-uniformes (dependentes de frequências) (VASEGHI, 2007) e suas respostas em magnitude são assimétricas e não-lineares. Por apresentar essas características, o efeito de dois ou mais tons presentes em uma mesma banda crítica é diferente do efeito deles em diferentes bandas críticas.

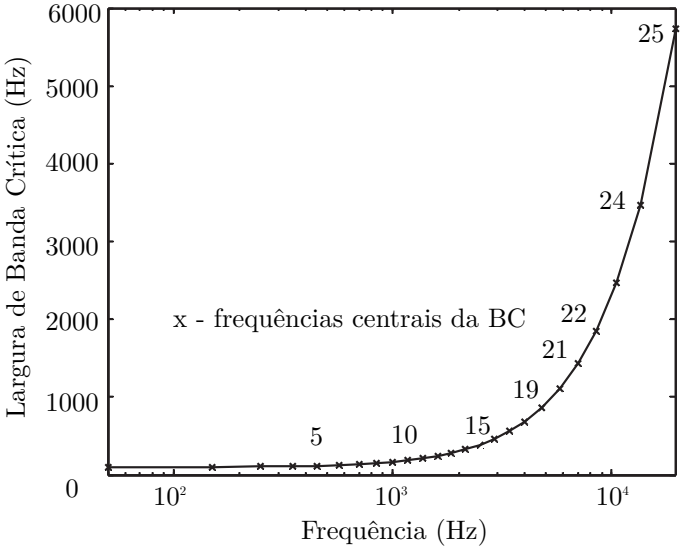
O termo largura de banda crítica (LBC) foi introduzido por Fletcher em 1940, que também usou a expressão BC para referir-se ao conceito de filtro auditivo (HOLMES; HOLMES, 2002). Desde que ele introduziu esse conceito, uma série de experimentos foram realizados para investigar o fenômeno de BC e estimar sua largura. Baseado em alguns experimentos preliminares, Zwicker, em 1961, classificou a LBC como função de uma frequência central. Esses valores são mostrados graficamente na Figura 5, os quais são modelados matematicamente pela seguinte equação:

$$\mathrm{BC}(f) = 25 + 75\left(1 + 1,4\left(\frac{f}{10^3}\right)^2\right)^{0,69} \qquad (\mathrm{Hz}). \tag{4}$$

Embora a função $\mathrm{BC}(f)$ seja contínua, é útil, para a construção de sistemas práticos, tratar o ouvido como um conjunto discreto de filtros conforme a Equação (4). Pela Figura 5, nota-se que a LBC tende a permanecer constante (cerca de 100 Hz) até 500 Hz e cresce para aproximadamente 20% da frequência central acima de 500 Hz.

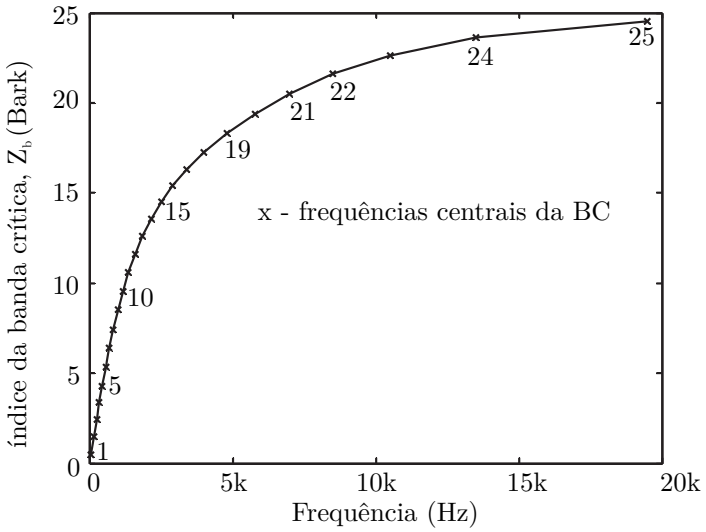
Como dito anteriormente, Zwicker classificou a LBC como função de uma frequência central. Ele propôs a escala *Bark* para essa finalidade, segundo o qual a diferença de 1 *bark* representa a largura de uma BC sobre uma inteira faixa de frequência (HOLMES; HOLMES, 2002).

Figura 5 – Gráfico do modelo da banda crítica obtido da Equação (4).



Fonte: Adaptado de Spanias, Painter e Atti (2007).

Figura 6 – Gráfico da relação entre a frequência (Hz) e a banda crítica na escala *Bark*.



Fonte: Adaptado de Spanias, Painter e Atti (2007).

O mapeamento da escala linear de frequência, em Hertz, para a escala *Bark*, $Z_c(f)$, ilustrado na Figura 6, é dado pela equação abaixo:

$$Z_c(f) = 13,0 \arctan(7,6 \times 10^{-4}) + 3,5 \arctan\left[\left(\frac{f}{7,5 \times 10^4}\right)^2\right] \qquad (Bark). \qquad (5)$$

A Tabela 2 apresenta um banco de filtros idealizado que corresponde aos pontos discretos numerados nas curvas das Figuras 5 e 6.

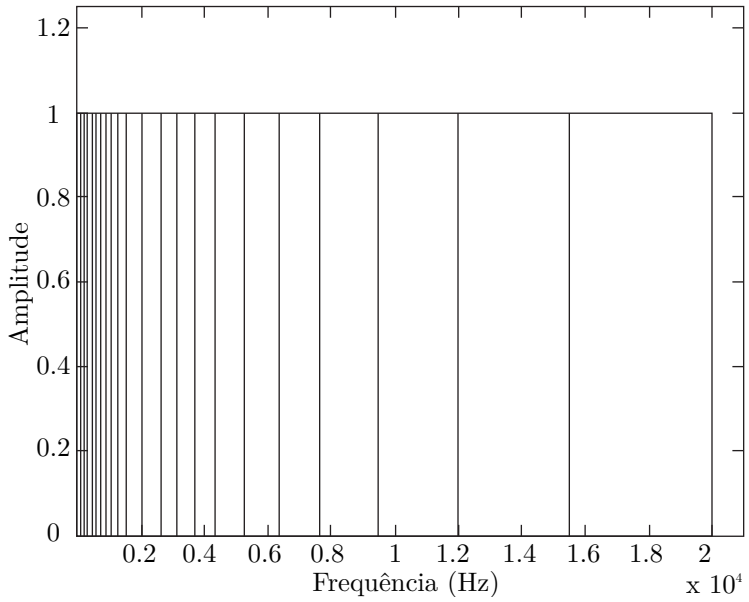
Tabela 2 – Banco de filtros idealizado.

Índice da Banda	**Frequência Central(Hz)**	**Largura de Banda (Hz)**
1	50	– 100
5	450	400 – 510
10	1175	1080 – 1270
15	2500	2320 – 2700
19	4800	4400 – 5300
21	7000	6400 – 7700
22	8500	7700 – 9500
24	13.500	12000 – 15500
25	19.500	15500 –

Fonte: Produzido pelo autor.

Correspondentes às frequências centrais da Tabela 2, os pontos numerados nas Figuras 5 e 6 ilustram que o espaçamento, em Hertz, não-uniforme dos banco de filtros (Figura 7) é, na prática, uniforme na escala *Bark* (SPANIAS; PAINTER; ATTI, 2007). Dessa forma, uma LBC engloba um *bark*.

Figura 7 – Esquema do banco de filtros de banda crítica idealizado.



Fonte: Adaptado de Spanias, Painter e Atti (2007).

Muitas das mais recentes estimativas da largura de banda crítica são baseados em experiências de mascaramento para determinar a forma do filtro auditivo e estimar a largura de banda

equivalente retangular (ERB, do inglês *"Equivalent Rectangular Bandwidth"*). Tais experiências são motivadas pelo fato dos filtros auditivos não terem uma resposta retangular no domínio da frequência e não serem completamente especificados pelas suas larguras de bandas críticas (HOLMES; HOLMES, 2002). A ERB é definida como a largura de banda de um filtro passa-banda retangular ideal que permite passar uma mesma potência que uma BC (WATKINSON, 2001).

2.3.4 Mascaramento

A faixa dinâmica de 120 dB do ouvido humano representa a relação entre os níveis de audibilidade e de desconforto para sons individuais. No entanto, quando dois sons de intensidades e frequências diferentes são combinados simultaneamente, ocorre o fenômeno do mascaramento, que pode ser descrito como um deslocamento relativo do limiar de audibilidade, provocado pela presença de tons de maior intensidade.

O mascaramento refere-se, em linhas gerais, ao processo no qual um som é interpretado como inaudível devido à presença de outro som (CHU, 2003), tornando-se, desta forma, irrelevante para a percepção auditiva humana. É esta característica que permite, aos diversos sistemas de codificação de áudio, conseguirem uma maior compressão de dados ao eliminar sons mascarados. Outras aplicações práticas em Engenharia de Áudio também aproveitam-se deste conceito, como em sistemas de redução de ruído. Projetistas destes sistemas podem, por exemplo, supor que um baixo nível de ruído que exista associado a um sinal de música de alta intensidade será efetivamente mascarado pelo sinal de música, estando os dois presentes na mesma banda de frequência. Este fato possibilita que eles utilizem uma menor resolução em bandas de frequência, pois o ruído será mascarado, de maneira efetiva, pelo sinal propriamente dito (RUMSEY; MCCORMICK, 2006).

A seguir, são descritos os dois principais tipos de mascaramento: o mascaramento em frequência (também chamado de mascaramento simultâneo) e o mascaramento no tempo (ou mascaramento não-simultâneo).

2.3.4.1 Mascaramento em Frequência

O mascaramento em frequência está relacionado ao conceito de bandas críticas da audição (VASEGHI, 2007). É um fenômeno pelo qual um sinal de baixa amplitude (sinal mascarado) se torna inaudível na presença simultânea de um sinal com uma amplitude muito maior (sinal mascarador). De uma forma simplificada, a presença de um ruído forte ou de um tom mascarador cria uma excitação de força suficiente sobre a membrana basilar, para bloquear, de forma eficaz, a detecção de um sinal mais fraco (SPANIAS; PAINTER; ATTI, 2007).

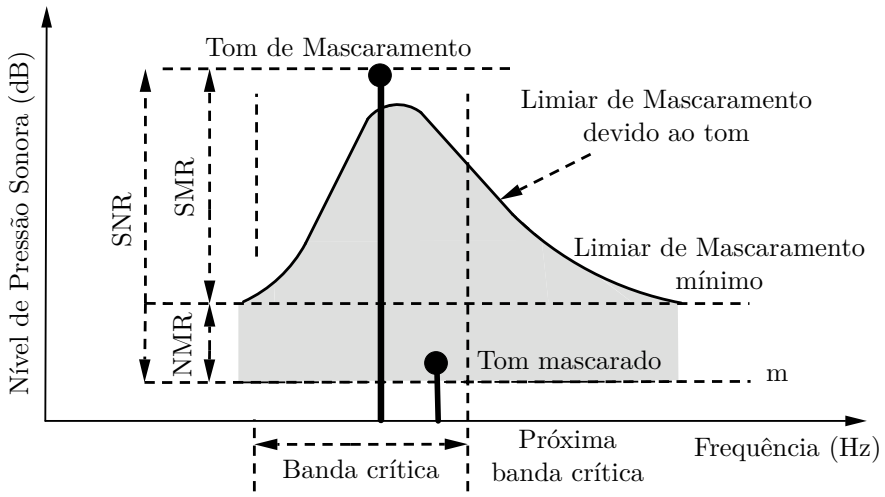
Os efeitos do mascaramento simultâneo não estão exclusivamente concentrados aos limites de uma única banda crítica. O mascaramento entre bandas também ocorre, permitindo que um tom mascarador centrado em uma banda tenha algum efeito previsível no limiar de detecção de outras bandas críticas. Este efeito, também conhecido como espalhamento do mascaramento (do inglês *"spread of masking"*), é muitas vezes modelado em aplicações de codificação por uma função de espalhamento (do inglês *"spread function"*) aproximadamente triangular, que tem inclinações de 25 e -10 dB por bark (SPANIAS; PAINTER; ATTI, 2007). Sua forma, $SF(x)$, pode ser obtida através da seguinte expressão:

$$SF(x) = 15,81 + 7,5(x + 0,474) - 17.5\sqrt{1 + (x + 0,474)^2}, \qquad (dB) \tag{6}$$

em que x está em unidades de *bark*.

No contexto de codificação de áudio, noções de largura de banda crítica e mascaramento simultâneo dão origem a algumas terminologias convenientes, como as ilustradas na Figura 8.

Figura 8 – Ilustração dos efeitos de mascaramento de um tom.



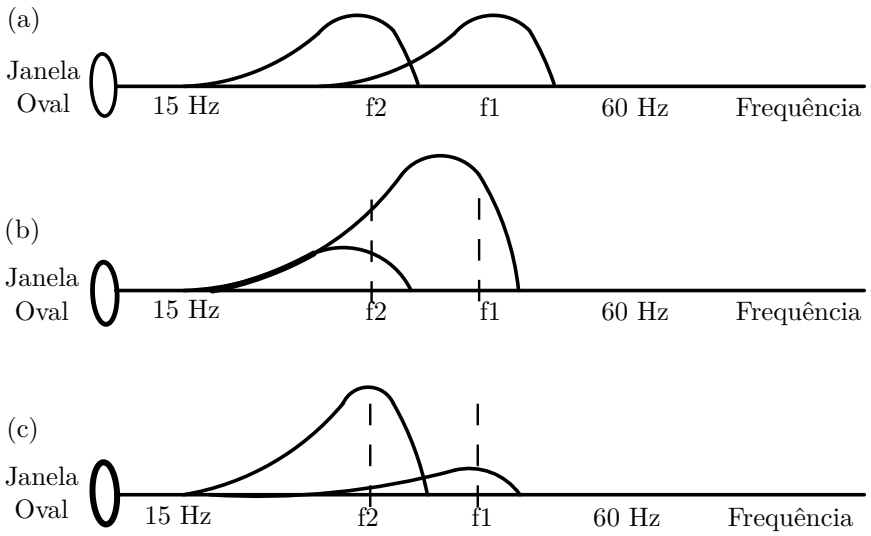
Fonte: Adaptado de Vaseghi (2007).

Nela, considera-se o caso de um único tom de mascaramento ocorrendo no centro de uma banda crítica. Supondo-se que este tom de mascaramento hipotético ocorre em algum nível de mascaramento, é gerada uma excitação ao longo da membrana basilar que é modelada por uma função de espalhamento e um limiar de mascaramento correspondente. Assumindo-se que o sinal mascarador é quantizado através de um quantizador escalar uniforme de m bits, algum ruído pode ser introduzido no nível de pressão sonora m. As relações sinal-mascarador (SMR) e ruído-mascarador (NMR) indicam as distâncias logarítmicas a partir do limiar de mascaramento mínimo para os níveis de mascaramento e ruído, respectivamente (SPANIAS; PAINTER; ATTI, 2007).

A faixa de frequências mascarada por um tom depende principalmente da área da membrana basilar colocada em movimento pelo tom, sendo o padrão de movimento desta membrana mais alargado nas altas do que nas baixas frequências. Se o sinal requerido produz mais movimento na membrana do que o tom de mascaramento, então ele irá ser percebido (RUMSEY; MCCORMICK, 2006).

A assimetria do mascaramento em frequência pode ser explicada pelo padrão de vibração da membrana basilar (Figura 9); devido à sua assimetria, tons de baixa frequência ($f1$) mascaram, mais fortemente, os de alta frequência ($f2$) do que o inverso (caso (a)). No caso (b), o tom $f1$ de maior amplitude mascara, completamente, o tom $f2$. Já no caso (c), um som de frequência $f2$, mesmo de maior amplitude, não consegue encobrir o som $f1$.

Figura 9 – Imagem da resposta da membrana basilar para dois tons senoidais.



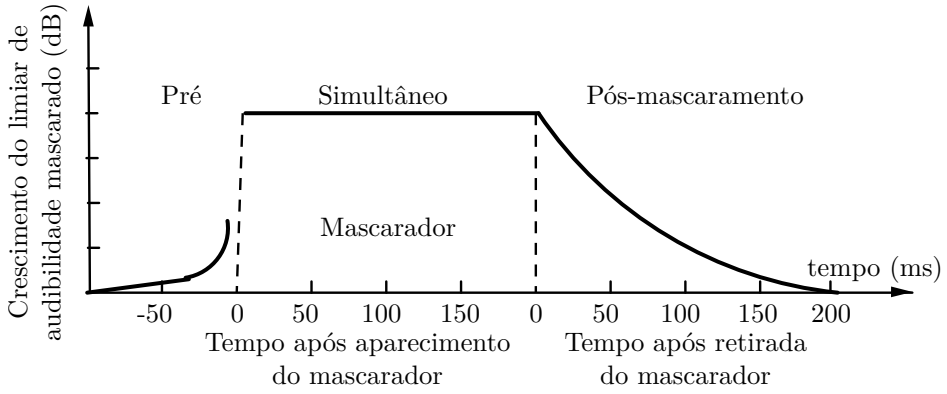
Fonte: Adaptado de Rumsey e McCormick (2006).

2.3.4.2 Mascaramento no Domínio do Tempo

O segundo principal efeito do mascaramento é o mascaramento no domínio do tempo ou mascaramento não-simultâneo. Como mostrado na Figura 10, o efeito do mascaramento de um sinal se estende a janelas temporais imediatamente anteriores (pré-mascaramento) ou posteriores (pós-mascaramento) ao instante de início do sinal mascarador (KAHRS; BRANDENBURG, 2002).

O pré-mascaramento ocorre antes do princípio de ação do mascarador e dura somente poucos milisegundos. O pós-mascaramento pode persistir por mais que 100 ms após a retirada de ação do mascarador, dependendo de sua intensidade e duração (SPANIAS; PAINTER; ATTI, 2007).

Figura 10 – Ilustração das propriedades do mascaramento temporal do ouvido humano.



Fonte: Adaptado de Spanias, Painter e Atti (2007).

Essencialmente, os limiares absolutos de audibilidade são artificialmente aumentados antes, durante e depois da ocorrência do sinal mascarador (SPANIAS; PAINTER; ATTI, 2007). Esse efeito torna possível o uso de sistemas de análise/síntese com um limitado tempo de resolução (e.g., banco de filtros de alta resolução) para a codificação de áudio em alta qualidade (KAHRS; BRANDENBURG, 2002).

Ao observar a Figura 10 é possível notar que, enquanto o pré-mascaramento tende a durar apenas 1–2 ms, o pós-mascaramento tende a se entender de 50 até 300 ms, a depender da intensidade e duração do sinal mascarador (SPANIAS; PAINTER; ATTI, 2007).

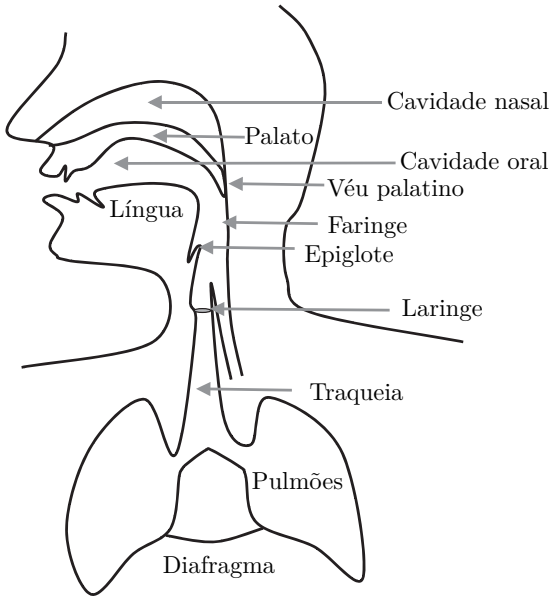
3 A VOZ

No capítulo anterior foi descrito o fenômeno acústico “som”, relatando suas características principais e as peculiaridades que o sistema auditivo humano apresenta ao interpretá-lo. Neste capítulo, o estudo é direcionado ao sinal de voz, foco das aplicações propostas por esta dissertação. Inicialmente é introduzido o seu mecanismo de produção, desde a geração a partir do fluxo de ar nos pulmões até a saída em forma de som pelas narinas e boca. Adicionalmente, são apresentados os tipos de sinais produzidos pelo sistema fonador humano, enfatizando suas diferenças mais relevantes. O capítulo é concluído com um modelamento matemático, de tempo discreto, para o sistema de produção da voz.

3.1 Mecanismo de Produção da Voz

Os sinais de voz são compostos de uma sequência de sons que servem como uma representação simbólica da mensagem produzida pelo locutor para o ouvinte (RABINER; SCHAFER, 1978). Essa sequência de sons é originada de movimentos controlados da estrutura anatômica que compõe o sistema fonador humano, e envolve três processos: fonte de geração, articulação e radiação (CHU, 2003). Uma visão simplificada deste sistema é apresentada na Figura 11, em que são mostrados os principais órgãos responsáveis pela geração da voz: pulmões, traqueia, laringe, faringe, cavidades nasal e oral. Todos eles interconectados formam o complexo órgão vocal humano (FURUI, 2000).

Figura 11 – Imagem da anatomia do sistema fonador humano.



Fonte: Adaptado de Vaseghi (2007).

É esse combinado mecanismo que origina uma variedade de vibrações e composições espectral-temporal e que forma os diferentes tipos de som (VASEGHI, 2007).

Uma importante estrutura do sistema vocal, composta pela faringe (a conexão do esôfago para a boca) e boca (ou cavidade oral), é chamada de trato vocal (RABINER; JUANG, 1993). Essa estrutura, análoga a um sistema acústico de transmissão, pode assumir diversas conformações dependendo dos movimentos da mandíbula, língua, lábios e outras partes internas (FURUI, 2000). Na média masculina, o comprimento do trato vocal é de cerca de 17 cm (RABINER; JUANG, 1993). O trato nasal, outra estrutura importante e que é responsável pela produção dos sons nasais, começa na úvula ou véu palatino e termina nas narinas (RABINER; SCHAFER, 1978).

O processo de produção da voz tem início quando o ar penetra nos pulmões via o mecanismo de respiração. À medida que o ar é expelido dos pulmões por meio da traqueia, as cordas vocais situadas nas laringes, na altura do pomo-de-Adão (saliência em frente a garganta de muitos adultos do sexo masculino), são induzidas a vibrarem em frequências entre 50 e 100 Hz (SMITH, 2003) pelo fluxo de ar (RABINER; JUANG, 1993). O efeito dessas vibrações induz a glote, pequeno orifício compreendido entre as cordas vocais — normalmente aberto durante a respiração — a se fechar e se abrir rapidamente (BARBOSA, 2008), enviando pulsos quase periódicos de ar pressurizados, os quais são modulados em frequência e passam pela faringe, cavidade da boca e, possivelmente, pela cavidade nasal.

As cordas vocais podem ser contraídas ou relaxadas para modificar a taxa de vibração ou ainda serem desativadas, permitindo a passagem de ar, sem obstrução, durante a respiração (PELTON, 1993). A velocidade com que elas se abrem e se fecham é única para cada indivíduo e define a característica e individualidade de uma voz particular (CHU, 2003).

Dependendo da posição dos articuladores (mandíbula, língua, úvula, lábios), diferentes formas da cavidade oral são criadas e diferentes sons são produzidos (RABINER; JUANG, 1993).

A natureza da onda sonora irradiada pelos lábios depende das diferentes configurações da cavidade oral, das características de absorção e reflexão acústica dos diversos materiais que a compõe, como também da fisiologia individual de cada pessoa (BARBOSA, 2008). Devido a essas distintas conformações da cavidade oral, o ar expelido pelos pulmões se choca com vários obstáculos durante seu percurso em direção aos lábios e/ou narinas, perdendo parte de sua energia. Grande parte dela é refletida, também, nas cavidades, combinando-se com outras frentes de onda. Alguma dessas ondas ressoam no trato vocal, que tem dimensões de alguns centímetros, reforçando a energia das ondas ressonantes ou atenuando a energia das ondas não-ressonantes, dando à voz características de timbre, altura (*pitch*) e intensidade. Essas frequências ressonantes aparecem no espectro de voz como máximos locais e são chamados de formantes (BARBOSA, 2008).

Uma forma de onda vocal contém muitos formantes, mas somente os três primeiros são importantes para a análise de sinais (os demais, no geral, são agrupados como um só). Normalmente, o primeiro formante, o de frequência mais baixa, ocorre numa faixa de 200 a 1200 Hz. O segundo cai numa faixa de transição entre 500 Hz e 3 kHz (PELTON, 1993). É possível modificar a amplitude e frequência dos formantes ao alterar a posição relativa entre língua e lábios.

3.2 Classificação dos Sons Produzidos pelo Sistema de Fonador Humano

Em termos gerais, os sinais produzidos pelo sistema fonador humano podem ser classificados como vocais ou não-vocais (BARBOSA, 2008). Sons vocais são gerados quando as cordas vocais vibram de tal maneira que o fluxo de ar dos pulmões é interrompido periodicamente, criando uma sequência de pulsos que excitam o trato vocal. Vogais são exemplos de sons vocais.

Os fonemas /i/ e /u/, algumas vezes, não são vogais. Eles aparecem apoiados em uma vogal, formando com ela uma só emissão de voz (uma sílaba). Nesse caso, são chamados de fonemas semi-vocais ou semi-vogais.

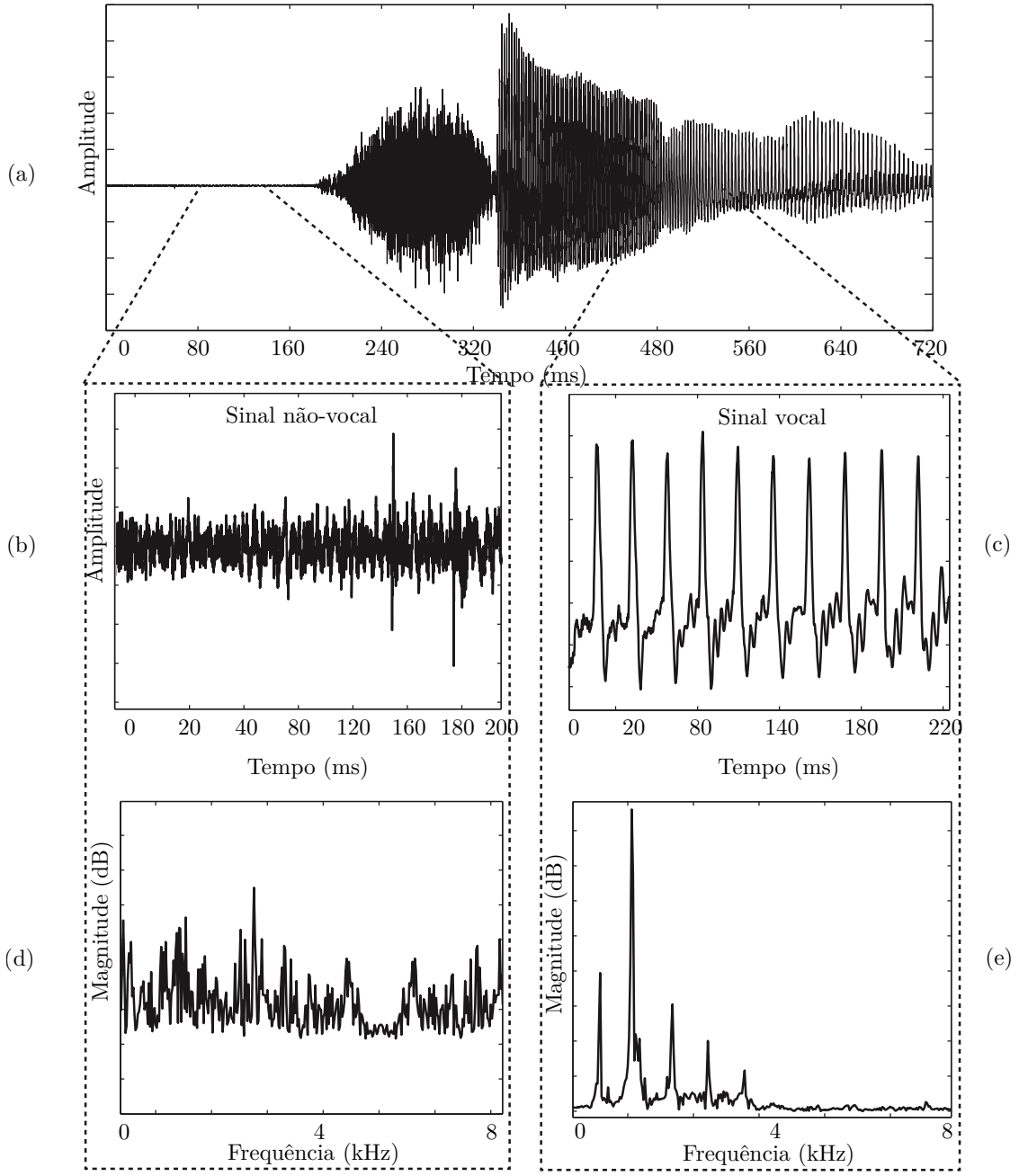
Com as cordas vocais estacionárias, a turbulência criada pelo fluxo de ar passando através de uma constrição do trato vocal produzida pela língua, dentes ou lábios, gera os sons não-vocais. Quando esses sons são resultantes de uma grande turbulência do trato vocal, possuindo grande conteúdo nas altas frequências, são referidos como fricativos. Sons não-vocais incluem /s/, /f/, /sh/, /z/, /v/, entre outras consoantes ou junção delas.

No domínio do tempo, um som vocal é caracterizado por uma forte presença periódica do sinal, tendo uma frequência fundamental referida como frequência de *pitch*. A frequência de *pitch* está situada tipicamente na faixa 50–200 Hz para adultos do sexo masculino e cerca de uma oitava acima para adultos do sexo feminino (HOLMES; HOLMES, 2002). Sons não-vocais, por outro lado, não exibem periodicidade e, essencialmente, apresentam um natureza aleatória (CHU, 2003).

A Figura 12 mostra um exemplo de uma onda de voz pronunciada por um indivíduo do sexo feminino, em que os sinais vocais e não-vocais estão presentes. É possível apreciar deste exemplo a característica não-estacionária dos sinais de voz, em que as estatísticas do sinal mudam constantemente com o tempo. Vê-se que, no trecho vocal do sinal, há uma clara periodicidade no domínio do tempo, na qual o sinal repete-se num padrão quase-periódico; e também, no domínio da frequência, onde uma estrutura harmônica é observada. No espectro, pode-se notar facilmente o comportamento dos formantes (picos da Figura 12 (e)).

Para os sinais não-vocais a característica é essencialmente aleatória, com baixas amplitudes e predominância de altas frequências. É essa composição de sinais vocais e não vocais, com diferentes timbres e amplitudes que possibilita a distinção de diversos tipos de vozes.

Figura 12 – (a): Pronúncia da palavra “sino”, (b): Ampliação do segmento não-vocal da palavra “sino”, (c): Ampliação do segmento vocal da palavra “sino”, (d): espectro em frequência para o item (b), (e): espectro em frequência para o item (c).

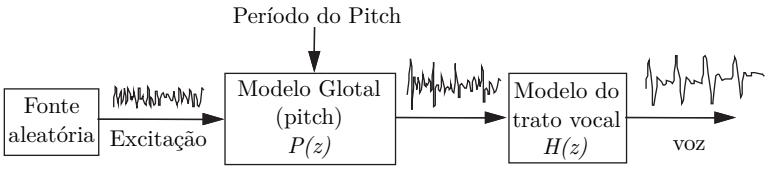


Fonte: Produzido pelo autor.

3.3 Modelamento do Sistema de Produção da Voz

Esta seção estabelece um *link* entre a fisiologia do sistema de produção da voz e as funções dos órgãos estudados na seção 3.1 e um modelo linear de produção da voz. Em termos gerais, um modelo é uma representação simplificada do mundo real (CHU, 2003). O sistema de produção da voz humana pode ser modelado através de um sistema de tempo discreto, conhecido como modelo *fonte-filtro* de produção da voz. O diagrama de blocos simplificado deste modelo é ilustrado na Figura 13. Ele sugere a separação do sistema articulatório em dois subsistemas independentes (GREENBERG et al., 2004): (i) um gerador de excitação (fonte aleatória), que simula os diferentes modos de geração do som no trato vocal e (ii) um filtro linear variante no tempo que simula a modelagem frequencial do trato vocal (RABINER; SCHAFER, 2007). Amostras do sinal de voz são assumidas como sendo a saída do sistema linear variante no tempo.

Figura 13 – Ilustração do modelo fonte filtro para o processo de geração de voz.



Fonte: Adaptado de Vaseghi (2007).

Na representação do modelo de produção da voz, a fonte aleatória cria uma excitação apropriada para o tipo de som a ser produzido, sendo geralmente constituída de duas componentes. A primeira é formada por uma sequência quase-periódica de pulsos discretos (glotais), que é responsável, principalmente, por produzir os sons vocais, incluindo as vogais e os sons semi-vocais. É também parcialmente responsável pela produção das consoantes vocais (fricativas, nasais e oclusivas). A localização dessa fonte quase-periódica é a glote (GREENBERG et al., 2004). A outra componente, caracterizada por um gerador de número aleatório, produz um sinal de ruído de tempo discreto com um espectro plano e é responsável por produzir os sons não-vocais (RABINER; SCHAFER, 2007). A ação, em conjunto, da fonte de excitação (que comuta de vocal para não-vocal) e da resposta em frequência do filtro (aplicada sobre o espectro do sinal) irá produzir o sinal de voz na saída.

Para produzir um sinal caracterizado como a voz, o tipo de excitação e as propriedades ressonantes do sistema linear devem variar no tempo. Na forma de onda ilustrada na Figura 12 é possível observar que as propriedades do sinal de voz mudam lentamente com o tempo. É razoável assumir, para sinais desse tipo, que as propriedades gerais de excitação e do trato vocal permanecem fixas para períodos de 10–20 ms (RABINER; JUANG, 1993). Assim, é comum caracterizar o filtro linear da Figura 13 por uma função de transferência, $H(z)$, com N pólos e

M zeros da forma:

$$H(z) = \frac{\sum_{j=0}^{M} b_j z^{-j}}{1 - \sum_{j=1}^{N} a_j z^{-j}} = \frac{b_0 \prod_{j=1}^{M} (1 - d_j z^{-1})}{\prod_{j=1}^{N} (1 - c_j z^{-1})}, \quad (1)$$

em que a_j e b_j são os coeficientes do filtro (parâmetros do trato vocal da Figura 13), que mudam a uma taxa da ordem de 50 a 100 vezes/s). Alguns dos pólos (c_j) da função transferência situam-se próximos ao círculo unitário e criam ressonâncias para modelar as frequências formantes. Os zeros d_j são empregados no modelo detalhado de produção da voz para modelar os sons nasais e fricativos (RABINER; SCHAFER, 2007). Muitas aplicações do modelo fonte-filtro (e.g., modelo de predição linear) simplificam a análise requerida para estimar os parâmetros do sinal de voz, considerando apenas os pólos no modelo.

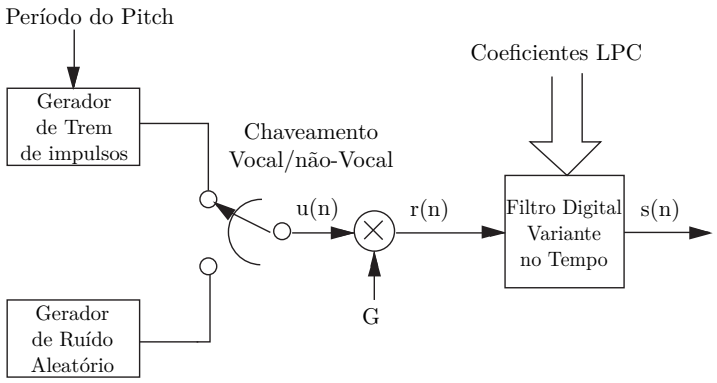
O modelo fonte-filtro, capaz de representar um sinal de voz como a saída de um filtro digital variando suavemente no tempo, em adição a uma excitação que capta a natureza vocal/não-vocal da voz é a base para muitos sistemas que representam o sinal de voz por um conjunto de parâmetros do modelo, em oposição à sua forma de onda amostrada. Ao assumir que as propriedades do sinal (e do modelo) são constantes sob curtos intervalos de tempo, é possível estimar esses parâmetros através da análise de curtos segmentos de amostras do sinal de voz (RABINER; SCHAFER, 2007), permitindo, assim, representá-lo em um formato digital.

Um dos métodos que utiliza o modelo teórico fonte-filtro como meio de análise de voz (codificação, reconhecimento e aprimoramento) é o método por predição linear ou análise LPC (do inglês *"Linear Predictive Coding"*), como é frequentemente intitulado. Ele é utilizado para estimar os parâmetros básicos da voz — frequência fundamental (*pitch*), formantes, espectro e função área do trato vocal — e para representá-la em uma baixa taxa de bits na transmissão ou armazenamento (COSTA FILHO, 2005). Na próxima seção tal modelo é apresentado.

3.4 Modelamento Preditivo Linear para Sinais de Voz

A forma particular do modelo de tempo discreto, adaptada para a análise LPC, é representada na Figura 14. Nela, o sinal de excitação é modelado como um trem de impulsos para sinais vocais e como um gerador de ruído aleatório para os sons não-vocais. A composição dos efeitos da radiação dos lábios, trato vocal e excitação glotal é representada pelo filtro digital variante no tempo (KONDOZ, 2004), referido, neste modelo, como o filtro de síntese de predição linear (LP) (SPANIAS; PAINTER; ATTI, 2007).

Figura 14 – Ilustração do modelo preditivo linear para o processo de geração da voz.



Fonte: Adaptado de Kondoz (2004).

Viu-se que no modelo fonte-filtro a função de transferência do filtro digital é composta de pólos e zeros. Se a intenção é representar sons nasais e fricativos, os zeros devem ser mantidos na análise. Porém, se a ordem do denominador for suficientemente alta, $H(z)$ pode ser aproximado por uma função apenas com pólos (Equação (2)), fornecendo uma boa representação para quase todos os sons do aparelho vocal (COSTA FILHO, 2005).

$$H(z) = \frac{S(z)}{U(z)} = \frac{G}{1 - \sum_{j=1}^{p} a_j z^{-j}} = \frac{G}{A(z)}, \tag{2}$$

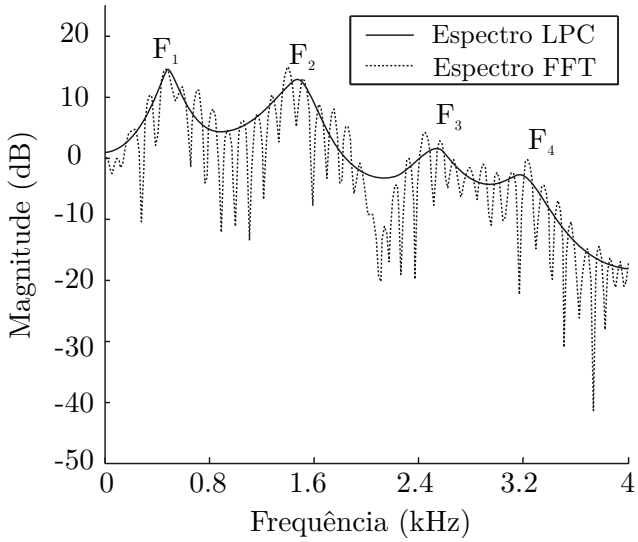
em que

$$A(z) = 1 - \sum_{j=1}^{p} a_j z^{-j} \tag{3}$$

é o filtro de análise LP. $S(z)$ e $U(z)$ representam as transformadas Z do sinal de voz, $s(n)$, e do sinal de excitação, $u(n)$, respectivamente. Os outros parâmetros do modelo são: classificação vocal/não-vocal, período de *pitch* para sinais vocais, parâmetro de ganho G (ajustado para controlar a intensidade da excitação), coeficientes a_j de predição linear (coeficientes LPC) e ordem p do filtro de predição. A maior vantagem desse modelo é que o ganho G e os coeficientes a_j podem ser estimados de maneira direta e computacionalmente eficiente pelo método de predição linear (RABINER; SCHAFER, 2007).

A resposta em frequência associada com o filtro de síntese LP representa, através do espectro LPC, a estrutura dos formantes em um sinal de voz (SPANIAS; PAINTER; ATTI, 2007). A Figura 15 ilustra esta propriedade em um sinal com quatro formantes (F1, F2, F3, F4).

Figura 15 – Gráfico dos espectros LPC e FFT para um sinal de voz.



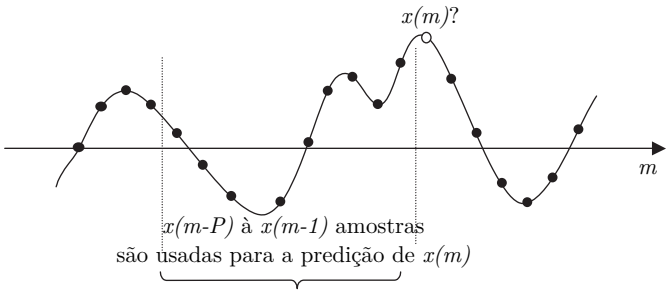
Fonte: Adaptado de Spanias, Painter e Atti (2007).

No modelo preditivo linear da Figura 14, as amostras de voz $s(n)$ são relacionadas à excitação pela seguinte equação:

$$s(n) = \sum_{j=1}^{p} a_j s(n-j) + Gu(n). \tag{4}$$

Ela expressa a ideia básica do modelo LPC: as amostras de saída $s(n)$ podem ser determinadas a partir de uma combinação linear de amostras passadas (RABINER; JUANG, 1993). Uma ilustração dessa ideia é mostrada na Figura 16.

Figura 16 – Ilustração da predição de uma amostra $x(m)$ a partir de P amostras passadas.



Fonte: Adaptado de Vaseghi (2007).

O problema básico da análise de predição linear é determinar o conjunto de coeficientes preditores a_j, de maneira a obter uma boa estimativa das propriedades espectrais do sinal de voz. Devido a natureza não-estacionária do sinal de voz, os coeficientes devem ser calculados em curtos intervalos de tempo. Deve-se, também, minimizar o erro médio quadrático sob curtos

segmentos de voz. Os parâmetros resultante são assumidos como os parâmetros de $H(z)$ no modelo de produção da voz (KONDOZ, 2004).

Um preditor linear com coeficientes de predição linear, α_j, é definido como um sistema cuja saída (RABINER; SCHAFER, 2007) é

$$\tilde{s}(n) = \sum_{j=1}^{p} \alpha_j s(n-j). \tag{5}$$

Se os coeficientes $\alpha'_j s$ representam uma estimativa para os $a'_j s$, o erro de predição $e(n)$, definido como a diferença entre o valor atual $s(n)$ e o valor predito $\tilde{s}(n)$, é dado por

$$e(n) = s(n) - \tilde{s}(n) = s(n) - \sum_{j=1}^{p} \alpha_j s(n-j). \tag{6}$$

Da Equação (6), pode ser visto que o erro de predição é a saída de um sistema linear cuja função do sistema é $A(z) = 1 - \sum_{j=1}^{p} a_j z^{-j}$, a mesma expressão do filtro de análise LP de (3). Também pode ser observado, comparando-se as Equações (4) e (6), que, se o sinal de voz obedece o modelo de (4), e se α_j = a_j, então $e(n) = Gu(n)$. Assim, o filtro do erro de predição, $A(z)$, será um *filtro inverso* para o sistema, $H(z)$, de (2), i.e. (RABINER; SCHAFER, 2007),

$$H(z) = \frac{G}{A(z)}. \tag{7}$$

3.4.1 Minimização do Erro

A ideia básica da análise preditiva é a determinação dos coeficientes a_j do preditor que minimizem o erro médio quadrático sobre um curto segmento de voz. Uma das justificativas para o uso do erro médio quadrático mínimo como estimativa para os parâmetros do modelo é que essa abordagem leva a uma representação extremamente útil e precisa do sinal de voz, que pode ser obtido por uma solução eficaz de um conjunto de equações lineares (RABINER; SCHAFER, 2007). O erro médio quadrático é dado por

$$\varepsilon = E[e^2(n)] = E\left[\left(s(n) - \sum_{j=1}^{p} \alpha_j s(n-j)\right)^2\right], \qquad 0 < p < 1. \tag{8}$$

Os valores de α_j que minimizam ε podem ser obtidos pelas derivadas parciais de ε com respeito aos coeficientes do preditor, que são posteriormente igualadas a zero, isto é:

$$\begin{aligned}
\frac{\partial \varepsilon}{\partial \alpha_i} &= 2E\left\{\left[s(n)-\sum_{j=1}^{p}\alpha_j s(n-j)\right]\left[\frac{\partial}{\partial \alpha_i}\left(\sum_{j=1}^{p}[-\alpha_j s(n-j)]\right)\right]\right\}=0 \\
&= 2E\left\{\left[s(n)-\sum_{j=1}^{p}\alpha_j s(n-j)\right][-s(n-i)]\right\}=0 \\
&= E\left\{-s(n)s(n-i)+\sum_{j=1}^{p}\alpha_j s(n-j)s(n-i)\right\}=0, \qquad 1\leq i\leq p. \qquad (9)
\end{aligned}$$

Finalmente, temos que

$$\sum_{j=1}^{p}\alpha_j E\left\{s(n-j)s(n-i)\right\}=E\left\{s(n)s(n-i)\right\}. \qquad (10)$$

Se nós definirmos

$$\phi_n(i,j)\triangleq E\{s(n-i)s(n-j)\}, \qquad (11)$$

então a Equação (10) pode ser escrita de uma forma mais compacta como

$$\sum_{j=1}^{p}\alpha_j\phi_n(i,j)=\phi_n(i,0), \qquad i=1,\ldots,p. \qquad (12)$$

Na dedução da Equação (12), uma hipótese importante é que o sinal do modelo seja estacionário. Para segmentos de voz, de longa duração, esta condição não se verifica. Entretanto, para curtos segmentos de voz, a suposição é admissível. Portanto, pode-se substituir as esperanças na equação (11) por somatórios finitos sobre amostras de voz de curta duração, i.e.,

$$\begin{aligned}
\phi_n(i,j) &= E\{s(n-i)s(n-j)\} \\
&= \sum_{m} s_n(m-i)s_n(m-j), \qquad i=1,\ldots,p, \qquad j=0,\ldots,p. \qquad (13)
\end{aligned}$$

A Equação (12) descreve um sistema com p equações e p incógnitas, que podem ser resolvidas para os coeficientes do preditor $\{\alpha_j,\ \ 1\leq j\leq p\ \}$ que minimizem ε na Equação (8).

Os métodos mais usados para resolvê-las são (CAVALCANTI, 2009):

I *Método da Autocorrelação* -

- Vantagem: Utiliza um algoritmo eficiente, com complexidade computacional $O(n^2)$;
- Desvantagem: Não representa adequadamente o espectro do sinal de excitação caso o mesmo esteja no estado estacionário.

II *Método da Covariância* -

- Vantagem: representa o sinal adequadamente estando este no estado estacionário;
- Desvantagem: Possui um algoritmo pouco eficiente $O(p^3)$.

3.4.2 Método da Autocorrelação

Para o método da autocorrelação (AM, do inglês *"Autocorrelation Method"*), o segmento do sinal de voz é assumido como sendo nulo fora do intervalo $0 \leq m \leq N-1$, em que N é o comprimento da sequência amostrada. Isto pode ser obtido através de um processo de janelamento, o qual consiste em multiplicar o segmento de voz $s(n)$ por uma janela $w(n)$ de comprimento N obtendo um sinal definido por

$$s_n(m) = \begin{cases} s(n)w(n), & 0 \leq m \leq N-1 \\ 0, & \text{caso contrário.} \end{cases} \tag{14}$$

Uma vez que o segmento de análise é definido, pelo janelamento, para ser nulo fora do intervalo $0 \leq m \leq N-1$, segue-se que a sequência do erro de predição pode ser não nula somente no intervalo $N \leq m \leq N+p$. Portanto, os limites da Equação (13) podem ser expressos como

$$\phi_n(i,j) = \sum_{m=0}^{N-1-|(i-j|)} s_n(m)s_n(m+|i-j|), \qquad i = 1,\ldots,p, \qquad j = 0,\ldots,p. \tag{15}$$

Esta equação pode ser reduzida à função de autocorrelação de curta duração, a qual é dada por

$$\phi_n(i,j) = R_n(|i-j|), \qquad i = 1,\ldots,p, \qquad j = 0,\ldots,p, \tag{16}$$

em que

$$R_n(j) = \sum_{m=0}^{N-1-j} s_n(m)s_n(m+j). \tag{17}$$

Usando o método da autocorrelação, a Equação (12) pode ser expressa como

$$\sum_{j=1}^{p} \alpha_j R_n(|i-j|) = R_n(i), \qquad 1 \leq i \leq p \tag{18}$$

O conjunto de equações dadas por (18) pode ser reescrito em forma matricial como

$$\begin{bmatrix} R_n(0) & R_n(1) & \dots & R_n(p-1) \\ R_n(1) & R_n(0) & \dots & R_n(p-2) \\ R_n(2) & R_n(1) & \dots & R_n(p-3) \\ \vdots & \vdots & \ddots & \vdots \\ R_n(p-1) & R_n(p-2) & \dots & R_n(0) \end{bmatrix} \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_p \end{bmatrix} = \begin{bmatrix} R_n(1) \\ R_n(2) \\ R_n(3) \\ \vdots \\ R_n(p) \end{bmatrix}. \tag{19}$$

Esta matriz $p \times p$, cujos elementos são valores de autocorrelação, tem as seguinte propriedades: é simétrica e tem todos os elementos, em cada diagonal com inclinação negativa, iguais, sendo portanto uma matriz de Toeplitz (RABINER; SCHAFER, 2007). Estas características apresentadas por matrizes de Toeplitz são exploradas por procedimentos recursivos eficientes, objetivando facilitar o processo de inversão da matriz e a obtenção dos coeficientes LPC. O procedimento mais largamente utilizado é o *algoritmo de Durbin* (KONDOZ, 2004), cujo diagrama é ilustrado na Figura 17 e que é um processo recursivo, a saber:

$$E_n^{(0)} = R_n(0); \tag{20}$$

$$k_i = \left[R_n(i) - \sum_{j=1}^{i-1} \alpha_j^{i-1} R_n(i-j) \right] \Big/ E_n^{(i-j)}, \qquad 1 \leq i \leq p; \tag{21}$$

$$\alpha_i^{(i)} = k_i; \tag{22}$$

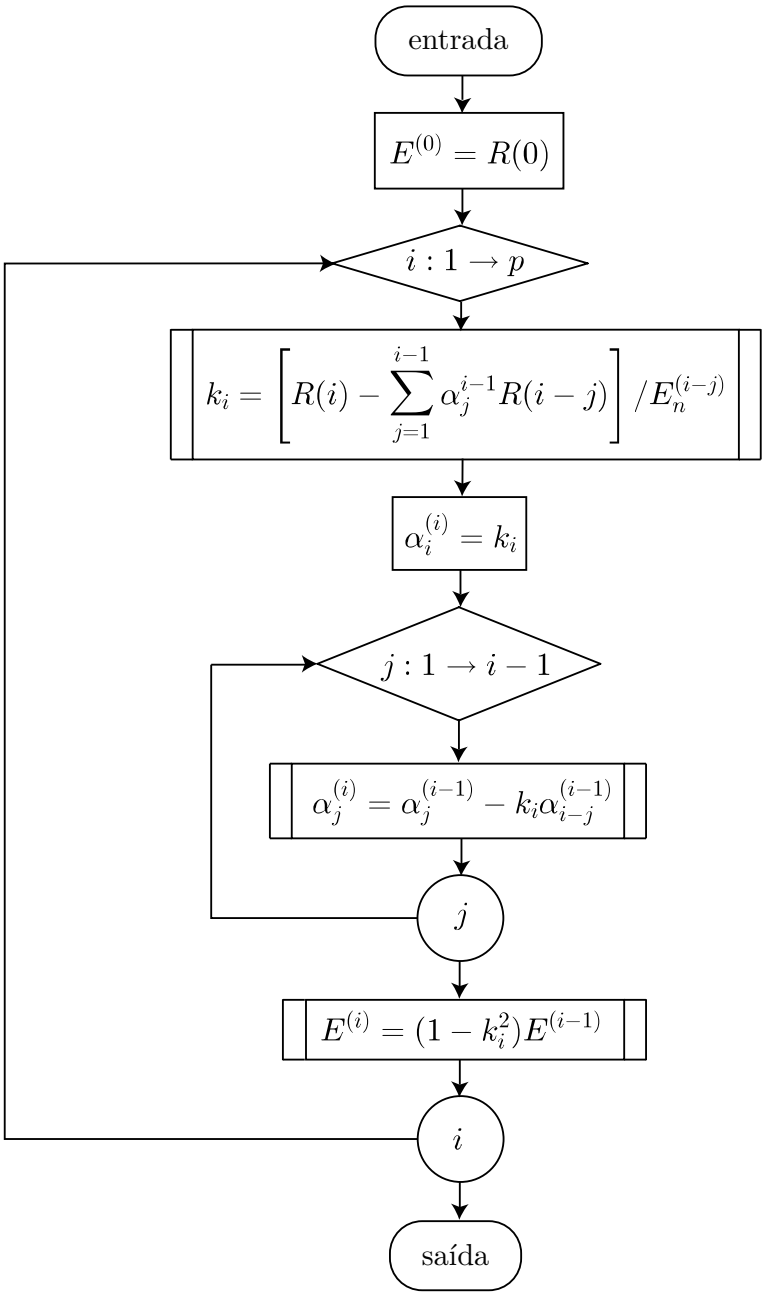
$$\alpha_j^{(i)} = \alpha_j^{(i-1)} - k_i \alpha_{i-j}^{(i-1)}, \qquad 1 \leq j \leq i-1; \tag{23}$$

$$E_n^{(i)} = (1-k_i^2) E_n^{(i-1)}. \tag{24}$$

Após o algoritmo resolver as cinco equações acima recursivamente para $i = 1, 2, \dots, p$, a solução final é dada por

$$a_j = a_j^p, \qquad 1 \leq j \leq p. \tag{25}$$

Figura 17 – Fluxograma do algoritmo de Durbin.



Fonte: Produzido pelo autor.

Considere um exemplo em que $p = 2$,

$$\begin{bmatrix} R_n(0) & R_n(1) \\ R_n(1) & R_n(0) \end{bmatrix} \begin{bmatrix} \alpha_0 \\ \alpha_1 \end{bmatrix} = \begin{bmatrix} R_n(1) \\ R_n(2) \end{bmatrix}. \tag{26}$$

Então, para $i = 1$,

$$
\begin{aligned}
E_n^{(0)} &= R_n(0); && (27)\\
k_1 &= \frac{R_n(1)}{R_n(0)}; && (28)\\
\alpha_1^{(1)} &= \frac{R_n(1)}{R_n(0)}; && (29)\\
E_n^{(1)} &= 1 - \frac{R_n^2(1)}{R_n^2(0)}; && (30)\\
R_n(0) &= \frac{R_n^2(0) - R_n^2(1)}{R_n(0)}. && (31)
\end{aligned}
$$

E para $i = 2$,

$$
\begin{aligned}
k_2 &= [R_n(2) - \alpha_1^{(1)} R_n(1)]/E_n^{(1)} = \frac{R_n(2)R_n(0) - R_n^2(1)}{R_n^2(0) - R_n^2(1)}; && (32)\\
\alpha_2^{(2)} &= k_2; && (33)\\
\alpha_1^{(2)} &= \alpha_1^{(1)} - k_2\alpha_1^{(1)} = \frac{R_n(1)R_n(0) - R_n(1)R_n(2)}{R_n^2(0) - R_n^2(1)}. && (34)
\end{aligned}
$$

E, dessa forma

$$\alpha_1 = \alpha_1^{(2)} \qquad \text{e} \qquad \alpha_2 = \alpha_2^{(2)}. \tag{35}$$

3.4.3 Método da Covariância

O Método da Covariância (CM, do inglês *"Covariance Method"*) utiliza uma abordagem oposta ao AM (KONDOZ, 2004). Nele o intervalo sobre o qual o erro médio quadrático é calculado é fixo, i.e,

$$E = \sum_{m=0}^{N-1} e_n^2(m). \tag{36}$$

Tal procedimento, permite escrever a Equação (13) como

$$\phi_n(i,j) = \sum_{m=0}^{N-1} s_n(m-i)s_n(m-j), \qquad i = 1,\ldots,p, \qquad j = 0,\ldots,p. \tag{37}$$

Alterando-se o índice do somatório tem-se:

$$\phi_n(i,j) = \sum_{m=-i}^{N-i-1} s_n(m)s_n(m+i-j), \qquad i = 1,\ldots,p, \qquad j = 0,\ldots,p. \tag{38}$$

Esta expressão é levemente diferente da descrita em (15) usada no AM, que requer o uso das amostras no intervalo $-p \leq m \leq N-1$. Na verdade a Equação (37) não é exatamente uma função de autocorrelação, mas, sim, uma correlação cruzada entre duas sequências de comprimento finito muito semelhantes, mas não idênticas (KONDOZ, 2004). Fazendo-se uso dela, a Equação (12) pode ser expressa como

$$\sum_{j=1}^{p} \alpha_j \phi_n(i,j) = \phi_n(i,0), \qquad i = 1,\ldots,p, \tag{39}$$

ou, em sua forma matricial,

$$\begin{bmatrix} \phi_n(1,1) & \phi_n(1,2) & \ldots & \phi_n(1,p) \\ \phi_n(2,1) & \phi_n(2,2) & \ldots & \phi_n(2,p) \\ \phi_n(3,1) & \phi_n(3,2) & \ldots & \phi_n(3,p) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_n(p,1) & \phi_n(p,2) & \ldots & \phi_n(p,p) \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \vdots \\ \alpha_p \end{bmatrix} = \begin{bmatrix} \phi_n(1,0) \\ \phi_n(2,0) \\ \phi_n(3,0) \\ \vdots \\ \phi_n(p,0) \end{bmatrix}. \tag{40}$$

A solução de (39) não é tão simples como a equivalente solução do AM. Isto se deve ao fato da matriz de covariância não ser uma matriz de Toeplitz, apesar de $\phi_n(i,j) = \phi_n(j,i)$. Entretanto, soluções eficientes de inversão de matrizes tais como a decomposição de Cholesky podem ser aplicadas, sendo ϕ expresso como (KONDOZ, 2004):

$$\phi = VDV^T, \tag{41}$$

em que V é uma matriz triangular inferior, cujos elementos da diagonal principal são unitários e D é uma matriz diagonal. Os elementos das matrizes V e D são obtidos a partir da Equação (41) por

$$\phi_n(i,j) = \sum_{m=1}^{j} V_{im} d_m V_{jm}, \qquad 1 \leq j \leq i-1, \tag{42}$$

ou de forma equivalente,

$$V_{ij}d_j = \phi_n(i,j) - \sum_{m=1}^{j-1} V_{im}d_m V_{im}, \qquad 1 \leq j \leq i-1, \tag{43}$$

e para os elementos da diagonal de D,

$$\phi_n(i,i) = \sum_{m=1}^{i} V_{im}d_m V_{im}, \tag{44}$$

ou,

$$d_i = \phi_n(i,i) - \sum_{m=1}^{i-1} V_{im}^2 d_m, \qquad i \geq 2 \tag{45}$$

e finalmente,

$$d_1 = \phi_n(1,1). \tag{46}$$

Desta forma, os coeficientes α_k do preditor, que minimizam o erro médio quadrático, podem ser obtidos.

4 CODIFICAÇÃO DIGITAL DA VOZ

Nos sistemas de comunicação digital, na qual se inclui a codificação de sinais de voz, é necessário que, antes de ser processado pelos diferentes estágios da rede, o sinal de entrada esteja no formato digital.

4.1 Digitalização da Voz

O processo de digitalização (conversão analógica-digital) consiste basicamente em dois processos: amostragem e quantização. A amostragem é o processo de extrair valores instantâneos de um sinal analógico em intervalos regulares de tempo (período de amostragem). Já a quantização envolve o processo de converter cada valor da amplitude do sinal, que varia no *continuum*, em um número finito de valores discretos, sendo estes representados por palavras código. O processo de amostragem, quando realizado segundo o teorema de Shannon-Nyquist, possibilita a recuperação do sinal original sem perda de informação (OPPENHEIM; SCHAFER; BUCK, 1999). Em contrapartida, a quantização é um processo irreversivelmente com perdas. Uma vez quantizado, o valor original da amplitude do sinal não mais pode ser recuperado (KONDOZ, 2004). Nas duas próximas seções essas duas técnicas são descritas.

4.1.1 Amostragem do Sinal

O primeiro estágio do processo de digitalização do sinal é a amostragem. Neste estágio, um sinal analógico $x_c(t)$ é convertido em um sequência de amostras $x[n]$ de acordo com a relação (OPPENHEIM; SCHAFER; BUCK, 1999):

$$x[n] \triangleq x_c(nT), \qquad -\infty < n < \infty, \tag{1}$$

em que T é o período de amostragem, sendo $f_s = 1/T$ a frequência de amostragem em amostras/s, que também pode ser expressa em rad/s como $\Omega_s = 2\pi/T$.

O teorema da amostragem diz que, se um sinal $x_c(t)$ tem uma transformada de Fourier dada por

$$X_c(j\Omega) = \int_{-\infty}^{\infty} x_c(t) e^{-j(\Omega T)t} dt, \qquad \Omega = \frac{\omega}{T}, \tag{2}$$

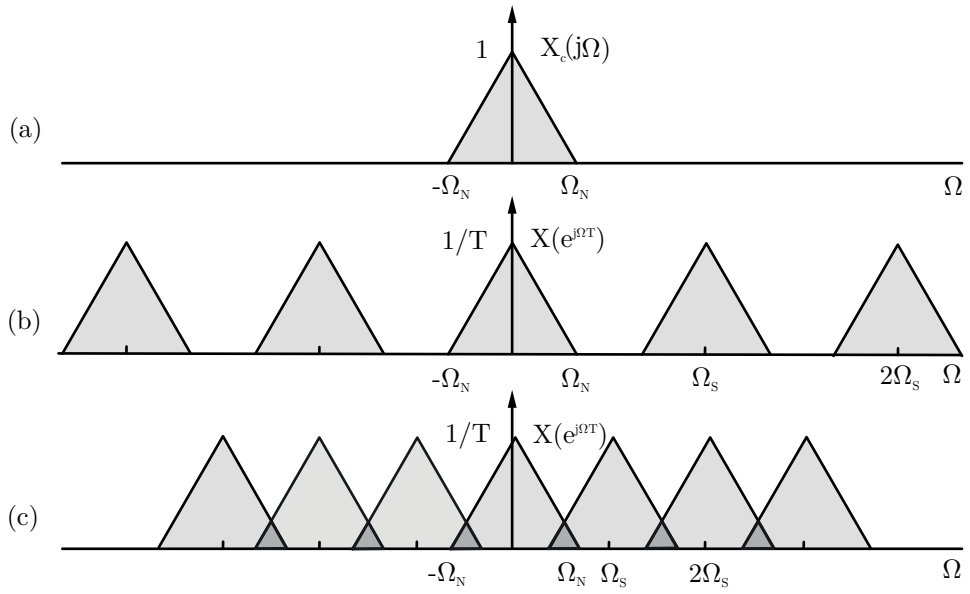
tal que $X_c(j\Omega) = 0$ para $|\Omega| \geq \Omega_N$, então pode ser reconstruído de sua versão amostrada

$x[n] = x_c(nT)$, $n = 0, \pm1, \pm2, \ldots$, se

$$\Omega_s = \frac{2\pi}{T} \geq 2\Omega_N, \tag{3}$$

em que a frequência Ω_N é a chamada *frequência de Nyquist*. O valor que deve ser excedido pela taxa de amostragem, $2\Omega_N$, é chamado de *taxa de Nyquist* (OPPENHEIM; SCHAFER; BUCK, 1999).

Figura 18 – Ilustração do efeito, no domínio da frequência, da amostragem no domínio do tempo. (a) Espectro do sinal original. (b) Espectro do sinal amostrado quando $\Omega_s > 2\Omega_N$. (c) Espectro do sinal amostrado quando $\Omega_s < 2\Omega_N$.



Fonte: Adaptado de Kondoz (2004).

O efeito da amostragem é ilustrado na Figura 18, em que é possível observar a representação da transformada de Fourier banda limitada, $X_c(j\Omega)$, duplicada em todo múltiplo da frequência de amostragem. Isto ocorre devido à transformada de Fourier do sinal amostrado ser avaliada nos múltiplos desta frequência, ou seja

$$X(e^{j\Omega T}) = \frac{1}{T} \sum_{k=-\infty}^{\infty} X_c(j(\Omega - k\Omega_s)). \tag{4}$$

Esta expressão pode ser interpretada observando-se o processo de amostragem no domínio do tempo, em que o sinal de entrada é regularmente (em cada intervalo de amostragem) multiplicado pela função delta ($\delta[n]$). Quando convertido para o domínio da frequência, a multiplicação torna-se uma convolução e o espectro é reproduzido em múltiplos da frequência de amostragem (KONDOZ, 2004).

Em outra análise da Figura 18 é possível notar que não há sobreposição no espectro ao se obedecer a relação imposta pelo teorema de Nyquist e, consequentemente, o sinal $x_c(t)$ pode ser recuperado, de sua versão repetida, com um filtro passa-baixas ideal. Visto de uma perspectiva no domínio do tempo, o filtro age como um interpolador ideal. Em contrapartida, não obedecida a relação, as cópias de $X_c(j\Omega)$ irão se sobrepor e o sinal original não poderá ser recuperado. A distorção causada devido a uma *taxa de Nyquist* insuficientemente alta é irreversível e é conhecida como *aliasing*.

O sinal de voz possui componentes espectrais até uma frequência de aproximadamente 12 kHz, no entanto, uma grande parcela dessas componentes não contribuem de forma essencial para a formação do sinal de voz. A maior parcela da energia encontra-se na faixa de frequências inferiores a 4 kHz. Portanto, em aplicações de telefonia, por exemplo, o sinal de voz é filtrado em 300–3,3 kHz e amostrado a uma taxa de 8.000 amostras por segundo (de acordo com o teorema de Shannon-Nyquist). Deste modo, tem-se não só um sinal inteligível, mas também a possibilidade de reconhecimento do interlocutor (BARBOSA, 2008).

4.1.2 Quantização

A quantização converte um sinal de amplitude contínua em um sinal de amplitude discreta, que difere-se do primeiro por um erro de quantização ou ruído (KONDOZ, 2004). Este deve ser mantido dentro de limites aceitáveis, segundo um determinado critério. Um dos mais utilizados é o da relação sinal-ruído (SNR) de quantização, que deverá ser o maior possível. O processo de quantização pode ser representado pela operação

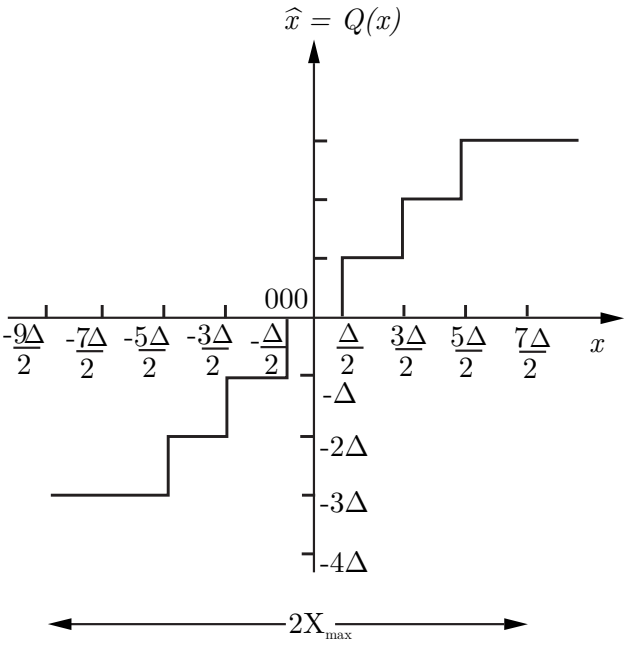
$$\hat{x}[n] = Q(x[n]), \tag{5}$$

em que $x[n]$ representa as amostras do sinal de entrada, $\hat{x}[n]$ as amostras quantizadas e $Q(.)$ a função de quantização ou quantizador. Existem outras técnicas de quantização (e.g., adaptativa e vetorial) (BARBOSA, 2008) (KONDOZ, 2004) mas apenas as quantizações uniforme e não-uniformes serão descritas nesse trabalho.

4.1.2.1 Quantização Uniforme

Um dos métodos mais comuns de quantização é a *quantização uniforme*, na qual a máxima excursão do sinal é dividida em m intervalos de mesmo comprimento, sendo cada um deles representado por uma única palavra-código (BARBOSA, 2008). A Figura 19 exibe a característica de um típico quantizador uniforme com 8 níveis.

Figura 19 – Gráfico das características de entrada-saída de um quantizador uniforme.



Fonte: Adaptado de Rabiner e Schafer (2007).

Pela figura, nota-se que o quantizador é apropriado para sinais cujas amostras são ambas positivas e negativas (bipolar). Para sinais que possuem apenas amostras de entrada positivas (ou negativas), uma reconfiguração apropriada dos níveis deve ser realizada (OPPENHEIM; SCHAFER; BUCK, 1999).

Para simplificar a codificação binária, é comum utilizar-se um número de níveis coincidente com uma potência de 2, de modo a otimizar o número de bits n de codificação por amostra. Nestas condições, a diferença entre níveis de quantização adjacentes, i.e., o passo de quantização, normalmente representado por Δ, é dado por

$$\Delta = \frac{2X_{max}}{m} = \frac{2X_{max}}{2^n}, \tag{6}$$

considerando $[-X_{max}, X_{max}]$ a faixa de variação do sinal de entrada a ser quantizado.

A diferença entre o valor real da amostra $x[n]$ e o seu valor quantizado $\hat{x}(n)$ produz um erro não-linear, conhecido por ruído de quantização e dado pela expressão

$$e[n] = x[n] - \hat{x}[n]. \tag{7}$$

Assumindo que o erro de quantização é um processo aleatório com média zero e uma distribuição de probabilidade uniforme, i.e, $\frac{-\Delta}{2} \leq e[n] \leq \frac{\Delta}{2}$, pode-se expressar a potência do ruído como:

$$\begin{aligned} E[e^2[n]] &= \int_{-\Delta/2}^{\Delta/2} p(e[n])e^2[n]de[n] = \frac{1}{\Delta}\int_{-\Delta/2}^{\Delta/2} e^2[n]de[n] \\ &= \frac{\Delta^2}{12} = \frac{X_{max}^2}{3\times 2^{2n}}, \end{aligned} \tag{8}$$

em que

$$p(e[n]) = \frac{1}{\Delta}, \qquad \text{para} \qquad |e[n]| \leq \frac{\Delta}{2}, \tag{9}$$

é a função densidade de probabilidade do ruído.

Usando-se a Equação (8), pode-se definir a relação sinal ruído de quantização (SQNR) por

$$\begin{aligned} \text{SQNR} &= 10\log_{10}\left(\frac{E[x^2[n]]}{E[e^2[n]]}\right) = 10\log_{10}\left(\frac{P_{sinal}}{X_{max}^2 2^{-2n}/3}\right) \\ &= 10\log_{10} 3 - 10\log_{10}\left(\frac{X_{max}^2}{P_{sinal}}\right) + 10\log_{10} 2^{2n} \\ &= 4.77 - \alpha + 6,02n, \end{aligned} \tag{10}$$

em que P_{sinal} é a potência média do sinal, n é o número de bits do quantizador e α é a relação (em dB) entre a potência de pico do sinal, X_{max}^2, e P_{sinal}. Para uma onda senoidal, tem-se que α = 3 dB. Pode ser visto da Equação (10) que para cada bit adicional, uma melhoria de 6 dB na relação sinal ruído de quantização é observada (VASEGHI, 2007).

4.1.2.2 Quantização Não-uniforme

Uma das desvantagens da utilização da quantização uniforme é a grande dependência da relação sinal-ruído com a potência do sinal de entrada (ver Equação (10)), sendo pior para sinais de baixa amplitude e melhor para sinais com grande amplitude (BARBOSA, 2008). A utilização de intervalos com dimensões diferentes, i.e., *quantização não-uniforme*, pode minimizar este problema. Diferentemente do quantizador uniforme, o tamanho do passo do quantizador não-uniforme cresce para os valores mais altos do sinal de entrada.

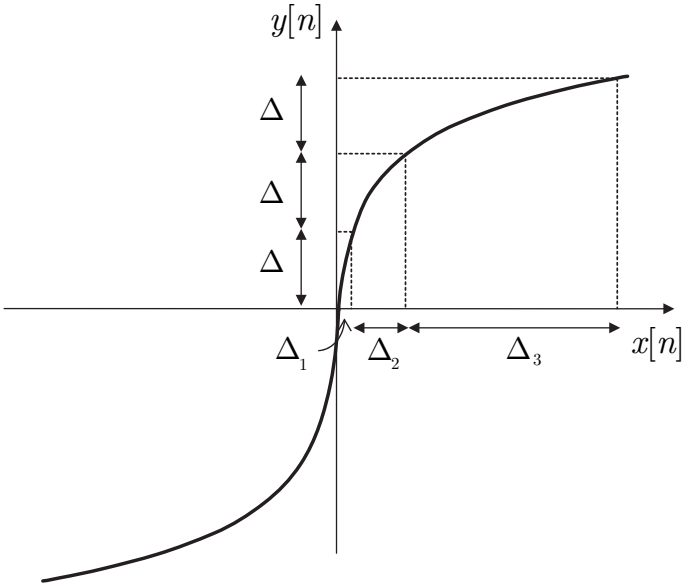
Um quantizador não-uniforme consegue, para o mesmo número de bits, relações sinal-ruído de quantização maiores, sendo útil, por exemplo, para sinais de voz com predominância de pequenas amplitudes. Em geral, a quantização não-uniforme é obtida, primeiramente,

distorcendo-se o sinal original com uma característica de compressão logarítmica (Figura 20) seguido de uma quantização uniforme. A compressão logarítmica é efetuada de acordo com

$$y[n] = h + k\log(x[n]), \tag{11}$$

em que h e k são constantes positivas. Tal expressão é valida apenas para valores positivos de $x[n]$, sendo necessária uma aproximação linear para abranger os valores nulos ou negativos.

Figura 20 – Gráfico da função de compressão de um quantizador não-uniforme típico. Δ_1, Δ_2 e Δ_3 são os diferentes passos de quantização.



Fonte: Adaptado de Spanias, Painter e Atti (2007).

Um processo inverso é requerido no receptor, de modo a não distorcer o sinal quantificado e possibilitar a recuperação em seu formato original. A operação, denominada de expansão, é conseguida através da função exponencial. O ciclo completo é geralmente chamado de *companding*, termo formado a partir da junção das palavras do inglês para compressão e expansão (***com****pressing* e ***expanding***) (BARBOSA, 2008).

Duas leis de compressão são definidas pelo ITU (*International Telecommunication Union*). São elas a lei-μ, que é adotada nos Estados Unidos, e a lei-A, que é utilizada na Europa. A lei-μ é dada pela relação

$$y[n] = X_{max}.\frac{\ln(1 + \frac{|x[n]|}{X_{max}})}{\ln(1+\mu)}.sign(x[n]), \tag{12}$$

em que

$$sign(x[n]) = \begin{cases} +1, & x \geq 0 \\ -1, & x < 0 \end{cases}. \tag{13}$$

Para a codificação de voz, o valor de $\mu = 255$ foi adotado como padrão nos Estados Unidos e Canadá. Este valor reduz em 24 dB a relação sinal-ruído de quantização em comparação à quantização uniforme. Desta forma, um quantizador de 8 bits, usado em conjunto ao um compressor logarítmico com $\mu = 255$, produz a mesma qualidade de voz que um quantizador uniforme de 12 bits. Um valor de $\mu = 0$ corresponde à quantização uniforme. A lei-A de compressão é dada pela expressão

$$y[n] = \begin{cases} \dfrac{A|x[n]|}{1+\ln(A)}.sign(x[n]), & \text{se } 0 \leq |x[n]| < \dfrac{1}{A}, \\ \\ \dfrac{1+A|x[n]|}{1+\ln(A)}.sign(x[n]), & \text{se } \dfrac{1}{A} \leq |x[n]| \leq 1, \end{cases} \tag{14}$$

em que A é o parâmetro de compressão com valores típicos de 86 para codificação PCM (*Pulse Code Modulation*) com 7 bits (nos EUA) e de $87,56$ para a codificação PCM com 8 bits (na Europa), ambos relacionados a sinais de voz (KONDOZ, 2004).

4.2 Codificação Paramétrica da Voz

O propósito da compressão de voz é obter uma concisa representação digital que possibilite a otimização de métodos de transmissão e armazenamento de dados de voz (HUANG et al., 2008). Com um processamento adequado, um sinal de voz pode ser analisado, codificado a uma baixa taxa de dados e, em seguida, ressintetizado. Em muitas aplicações, a codificação digital da voz é necessária para permitir a utilização de algoritmos de criptografia (por questão de segurança) ou de técnicas de correção de erros (para compensar o ruído do canal de transmissão). Utilizando-se de alguns métodos de codificação, pode-se conseguir alta qualidade de voz, mas ao custo de taxas de bits elevadas. Tais codificadores são chamados de codificadores de forma onda. Esses codificadores tentam reproduzir amostra por amostra o sinal original e são projetados para serem independentes do tipo de sinal. Desta forma, podem ser utilizados para codificar um ampla variedade deles (COSTA FILHO, 2005).

A largura de banda disponível para a transmissão de voz digitalizada é estreita, da ordem de 4 kHz (POPE; SOLBERG; BRODERSEN, 1987). Em tais condições, é necessária a utilização de sistemas de codificação que reduzam a taxa de bits, a fim de que a informação possa ser transmitida apropriadamente. Esses sistemas, no entanto, não conseguem reproduzir a forma de onda da voz em seu formato original. Ao invés disso, um conjunto de parâmetros variáveis são extraídos da voz, transmitidos e usados para gerar uma nova forma de onda no receptor. Essa forma de onda recriada poderá não aproximar necessariamente a forma de onda original em aparência, mas deverá ser perceptualmente similar (HOLMES; HOLMES, 2002). Este tipo de codificador, nomeado de *vocoder* (da contração VOice CODER), termo também usado largamente para referir-se a codificadores de análise/síntese em geral, irá utilizar-se de características perceptualmente importantes dos sinais de voz para representá-los de maneira mais eficiente e sem comprometer, significativamente, a sua qualidade (HOLMES; HOLMES, 2002).

O *vocoder* foi descrito, primeiramente, por Homer Dudley do Laboratório de Telefonia da Bell em 1939 e consistia de um sintetizador de voz operado manualmente (PELTON, 1993). A partir daí, iniciou-se uma moderna investigação sobre codificação de sinais de voz. Os *vocoders* baseiam-se no fato de que o trato vocal muda lentamente e seu estado e configuração podem ser representados por um conjunto de parâmetros. Tipicamente, esses parâmetros são extraídos do espectro do sinal de voz e atualizados a cada 10–25 ms. Em geral, dada a sua complexidade no processo de geração da voz sintetizada, as modelagens, simplificações e aproximações utilizadas pelos *vocoders* introduzem perdas e distorções que acabam por tornar a qualidade de voz obtida inferior àquela obtida pelos codificadores de onda (BARBOSA, 2008).

Duas propriedades da comunicação de voz são extensivamente exploradas pelos *vocoders*. A primeira é a capacidade de restrição do sistema auditivo humano, que torna os ouvintes insensíveis a várias imperfeições no processo de reprodução da voz. A segunda diz respeito à fisiologia do processo de geração da voz, que coloca fortes restrições sobre o tipo de sinal que pode ocorrer. Este fato pode ser explorado para modelar alguns aspectos da produção da voz (HOLMES; HOLMES, 2002).

Nos critérios de escolha de um codificador de voz para determinada aplicação, existem alguns atributos que são decisivos, enquanto outros são menos significativos. Assim, um compromisso pode ser levado em consideração. Na seção seguinte, é descrito, de modo sucinto, cada um destes atributos, sendo abordados os principais compromissos envolvidos.

4.3 Atributos dos Codificadores de Voz

Alguns critérios são importantes na escolha de um codificador de voz para uma determinada aplicação. São eles:

1. Taxa de bits;
2. Qualidade do sinal de saída;
3. Complexidade dos algoritmos e quantidade de memória necessária;
4. Atraso;
5. Sensibilidade a erros de canal.

Aplicações diferentes requerem que os codificadores sejam otimizados para diferentes características ou algum balanço entre elas. Para um grande número de aplicações, o principal objetivo é assegurar a similaridade entre o sinal original e o reconstruído e, em alguns casos, como nos sistemas em que a segurança é o principal objetivo, que os arquivos de voz reconstruídos sejam inteligíveis (COSTA FILHO, 2005).

4.3.1 Taxa de Bits

A primeira motivação da codificação da voz é a redução da taxa de bits, com vista a transmitir ou armazenar dados de maneira mais eficiente. Pode-se imaginar que a redução da informação a ser transmitida está, necessariamente, atrelada a uma redução da qualidade do sinal original. Entretanto, codificadores de voz de alta qualidade podem conseguir reduzir a taxa de bits por um fator de 13 ou mais com nenhuma perda perceptível na qualidade ou inteligibilidade (SPANIAS; PAINTER; ATTI, 2007).

Em sistemas que utilizam a largura de banda telefônica, a taxa padrão utilizada é de 8.000 amostras/s. Para que um sinal amostrado a essa taxa seja dito de alta qualidade, sendo considerado uma referência, ele deve apresentar uma resolução de 16 bits por amostra, com quantização uniforme (PCM), resultando numa taxa de bits de 128 kbits/s. É um sinal com essa configuração que será processado de modo a gerar um conjunto de bits com uma menor taxa, sendo posteriormente transmitido ou armazenado. No receptor, esse conjunto de bits (servindo como entrada) será reconstruído numa aproximação original (em PCM), que posteriormente o converterá num sinal acústico (BARBOSA, 2008).

Habitualmente, os codificadores produzem taxa de bits fixas, que são ideais para utilização em canais de transmissão não partilhados, os que não levam em conta critérios para definir, em cada instante, qual a taxa de bits atribuída. Entretanto, hoje em dia já se é capaz de utilizar taxas de bits variáveis (VBR, *Variable Bit Rate*), uma vez que as redes de comunicação utilizam protocolos flexíveis, (CAVALCANTI, 2009).

4.3.2 Qualidade do Sinal de Saída

Uma das maiores dificuldades no projeto e teste dos codificadores é a medição da qualidade do sinal reconstruído. A complexidade está na falta de uma medida objetiva para a

qualidade da voz que represente a percepção entre o sinal original e o sinal reconstruído em forma de uma função erro (COSTA FILHO, 2005).

Os métodos de análise da qualidade dos codificadores podem ser divididos em dois tipos: objetivos e subjetivos (SPANIAS; PAINTER; ATTI, 2007). Nos métodos objetivos, comparam-se características matemáticas do sinal transmitido com as do sinal recebido. Medidas subjetivas envolvem seres humanos, que são solicitados a atribuir classificações para alguns diálogos em tempo real ou gravados (CAVALCANTI, 2009). Ao utilizar-se de seres humanos, as medidas subjetivas são frequentemente mais precisas e úteis para avaliação de um sistema de telefonia.

4.3.2.1 Métodos Objetivos

O desempenho dos codificadores de forma de onda, tais como o PCM, pode simplesmente ser medido pela relação sinal-ruído, SNR. Sendo $x[n]$ o sinal original de voz e $y[n]$ a versão sintética dele, tem-se:

$$\text{SNR} = 10 \cdot \log_{10} \left(\frac{\sum\limits_{n} x^2[n]}{\sum\limits_{n} (x[n] - y[n])^2} \right), \tag{15}$$

com o índice n variando no intervalo de medição.

Um método mais refinado de medida objetiva, no que diz respeito à SNR convencional, é a relação sinal ruído-de-segmentação (SSNR). Este método, definido pela Equação (16), foi criado para lidar com a natureza dinâmica de sinais não estacionários como a voz.

$$\text{SNRR} = \frac{1}{N} \sum_{m=1}^{N} \text{SNR}_m. \tag{16}$$

Da Equação (16), vê-se que a SNRR é uma média dos valores da SNR de quadros isolados. A vantagem de sua utilização, em vez da SNR convencional, é que, calculando-se a média sobre os valores de SNR no domínio logarítmico, ocorre uma melhor ponderação para os segmentos não-vocais da voz com baixa energia. Por essa razão, a SSNR correlaciona melhor a qualidade perceptiva do que a SNR (COSTA FILHO, 2005).

As medidas de SNR e SSNR são significativas apenas para os codificadores de forma de onda. Ambas são extremamente sensíveis a distorções de forma de onda e a distorções de fase, que nem sempre são perceptualmente relevantes. Por outro lado, muitos codificadores de baixa taxa não preservam a forma de onda original e, portanto, a SNR e a SSNR são insignificantes

para a avaliação destes codificadores. As técnicas de medidas subjetivas são feitas para superar essas limitações de abordagem simples da SNR (CHU, 2003).

4.3.2.2 Métodos Subjetivos

Em testes subjetivos, são executadas amostras de voz a um determinado grupo de ouvintes, que são convidados a classificar a qualidade dos sinais ouvidos. Essas classificações são reunidas e é calculada a média para produzir o resultado final. Os testes são feitos, normalmente, para uma grande variedade de condições (naturalidade, inteligibilidade, facilidade de reconhecimento do locutor, etc) de modo a obter uma avaliação do desempenho geral de um determinado codificador (CHU, 2003). Alguns testes (os mais populares) são descritos a seguir.

4.3.2.2.1 **ACR (Absolute Category Rating)**

Nesse teste, um dos mais destacados, ouvintes são convidados a classificar uma frase ou trecho de voz produzido pelo codificador em teste. Os ouvintes atribuem um grau de 1 (ruim) a 5 (excelente). A partir dos graus obtidos, é calculado o valor médio (MOS, do inglês *"Mean Opinion Score"*) e gerada uma classificação final, em termos perceptuais, a partir da Tabela 3. Normalmente, são escolhidos indivíduos leigos, não treinados para o teste, mas pode-se utilizar um número menor de indivíduos treinados capazes de distinguir a fonte e o tipo de degradação. Ambos os conjuntos possuem sua relevância, mas o de indivíduos leigos é mais adequado, pois se aproxima mais da média da população (CAVALCANTI, 2009).

Tabela 3 – Escala MOS.

Avaliação	Qualidade da Voz	Nível de Distorção
5	Excelente	Imperceptível
4	Boa	Apenas perceptível, mas não irritante
3	Satisfatória	Perceptível e levemente irritante
2	Razoável	Irritante, mas não desagradável
1	Ruim	Muito irritante e desagradável

Fonte: Produzido pelo autor.

4.3.2.2.2 **DCR (Degradation Category Rating)**

Nesse teste, é solicitado aos ouvintes que seja feita uma comparação entre o sinal original, tomado como referência, e o sinal sintetizado. De acordo com a quantidade de degradação percebida, eles irão classificar de 1 (degradação muito audível) a 5 (degradação pouco audível). A média dos votos, o resultado final, é conhecida como DMOS (do inglês *"Degradation Mean Opinion Score"*).

A confiabilidade dos resultados dos testes subjetivos depende do número de ouvintes. Na maioria dos testes o número mínimo é de 16, com os ouvintes sendo escolhidos entre a população em geral, de modo a refletir melhor as condições em que o sistema eventualmente será implantado. Pela logística necessária para sua realização, os testes subjetivos tornam-se caros e bastante demorados. Portanto, os esforços de pesquisa atuais estão sendo direcionados para o desenvolvimento de procedimentos e avaliação de testes automáticos e de medidas objetivas que sejam capazes de predizer a qualidade subjetiva da voz. Os algoritmos mais conhecidos para a avaliação objetiva da qualidade da voz baseada no modelo de percepção psicoacústica dos sons são: BSD (do inglês *"Bark Spectral Distance"*) (WANG; SEKEY; GERSHO, 1992), PSQM (do inglês *"Perceptual Speech Quality Measure"*) - recomendação P.861 da ITU-T (ITU-T, 1998), e PESQ (do inglês *"Perceptual Evaluation of Speech Quality"*) - recomendação P.862 da ITU-T (RIX et al., 2001).

Por ser um método já bem estabelecido e bastante utilizado em aplicações envolvendo classificação da qualidade de voz com alto grau de consistência, o sistema de codificação proposto faz uso do método subjetivo ACR para medição da qualidade dos sinais de voz gerados.

4.3.3 Complexidade dos Algoritmos e Quantidade de Memória Necessária

Quanto mais complexidade apresentar um algoritmo de codificação e maior for a quantidade de memória requerida para seu processamento, mais seus sistemas serão dispendiosos e volumosos, requerendo um maior consumo de energia. A complexidade é normalmente aferida pela quantidade de instruções requeridas para processar os algoritmos de codificação. Usualmente é mensurada em MIPS (milhões de instruções por segundo) ou em MFLOPS (milhões de operações em ponto flutuante por segundo), enquanto que a memória necessária é mensurada em número de bytes.

Uma técnica usada por codificadores para economizar o consumo de energia e também para melhorar a eficiência do canal é a interpolação digital da voz (DSI, do inglês *"Digital Speech Interpolation"*) (KONDOZ, 2004). A DSI explora o fato de que somente cerca de metade da conversação é realmente ativa. Em períodos de inatividade, o canal pode ser utilizado para outras finalidades, incluindo a limitação da atividade do transmissor, permitindo uma maior economia de energia. Um subsistema importante da DSI é o detector de atividade de voz (VAD, do inglês *"Voice Activity Detection"*), que deve funcionar de forma eficiente e confiável para garantir que a voz original não seja confundida com o silêncio e vice-versa (KONDOZ, 2004). Utiliza-se o VAD, também, antes do estágio de pré-processamento do sinal em sistemas de reconhecimento de voz e locutor. No sistema de RAL proposto nessa dissertação, o VAD é empregado para economizar tempo de processamento.

4.3.4 Atraso

No contexto de codificação de voz, atraso refere-se ao tempo necessário para o processamento da codificação do sinal, ou seja, o tempo que separa o instante em que uma amostra é apresentada ao emissor, e que a correspondente amostra é gerada pelo receptor. Este tempo é medido em milisegundos, estando o receptor em conexão direta ao emissor. Não se considera, na medição, a contribuição dos equipamentos de emissão e recepção e o tempo de propagação do sinal, mas sim o tempo de transmissão de cada bit. Geralmente, quanto maior a taxa de bits menor o atraso, visto que taxas de bits maiores representam algoritmos de codificação com menor compactação (SOUZA et al., 2007).

Embora não seja importante em aplicações de armazenamento, como no sistema de codificação de voz proposto nesse trabalho, em aplicações de conversação, como na rede telefônica, o atraso pode se tornar inconveniente, tedioso, vindo a afetar a naturalidade da conversação. Limites máximos permissíveis para esse atraso podem chegar a 400 ms (BARBOSA, 2008). Em redes de comunicação que não incluem canceladores de eco, essa restrição pode se tornar mais severa, visto que os próprios interlocutores podem chegar a notar suas próprias elocuções.

4.3.5 Sensibilidade ao Erro

Na transmissão em canais de comunicação, o sinal codificado fica sujeito a erros que são introduzidos pelos canais. Esses erros normalmente são de dois tipos: (i) erros aleatórios, causados pelo ruído estacionário, e (ii) erros em surto, introduzidos por interferências eletromagnéticas nas imediações do canal (BARBOSA, 2008). Para que ambos os tipos de erros afetem, minimamente, a qualidade dos sinais de saída, os codificadores devem possuir técnicas que empreguem códigos capazes de detectar e corrigir esses erros. Entretanto, a introdução desses códigos demanda uma maior taxa de bits (SPANIAS; PAINTER; ATTI, 2007).

4.4 Técnicas de Codificação de Voz

De maneira geral, a diferença básica entre codificação em formato de onda e codificação paramétrica está ligada à filosofia de como o sinal é codificado. Codificadores de forma de onda transmitem o sinal em seu formato original ou como uma variante do mesmo. Codificadores paramétricos, por outro lado, transmitem apenas parâmetros resultantes de tratamentos matemáticos realizados sobre propriedades extraídas do sinal. A seguir, algumas técnicas de codificação são descritas.

4.4.1 G.711 - PCM

O padrão G.711, desenvolvido pelo ITU-T em 1972, define um codificador de forma de onda que utiliza modulação por código de pulso (PCM, do inglês *"Pulse Code Modulation"*) com quantização não-uniforme (ITU-T, Geneva, 1993), cujas características de compressão e expansão do quantizador são aproximações lineares por partes para as leis-μ e A, com μ = 255 e A = 87,56. Essas características não-lineares, ou mapeamentos de entrada-saída, são explicitamente expressas na forma de uma tabela, onde as entradas são amostras PCM uniformes de 13 ou 14-bits (CHU, 2003).

Em concordância com o teorema de Shannon-Nyquist e, levando em conta que o espectro de voz utilizado em telefonia possui uma banda de 4 kHz, foi adotada, pelo padrão, uma taxa de amostragem de 8.000 amostras/s com 256 níveis de quantização, resultando uma taxa de transmissão de 64 kbits/s (CHU, 2003). Por ser o primeiro sistema digital de telefonia, o padrão G.711 foi implantado em vária redes comutadas de telefonia pública (PSTNs, do inglês *"Public Switched Telephone Network"*) (KONDOZ, 2004).

4.4.2 G.722 - SB-ADPCM

Em 1976, o ITU-T definiu o padrão G.722, essencialmente um codificador sub-banda cujo sinal de entrada é dividido em regiões de baixas e altas frequências, posteriormente codificadas separadamente utilizando-se a técnica de modulação por código de pulso diferencial adaptativo (ADPCM, do inglês *"Adaptive Differential Pulse Code Modulation"*) (CHU, 2003). Foi proposto para aplicações de áudio de alta qualidade, codificando a voz no espectro de 50 Hz a 7 kHz (BARBOSA, 2008), operando a uma taxa de transmissão de 64 kbits/s, com codificação de 14 bits e frequência de amostragem de 16 kHz (CAVALCANTI, 2009).

No seu funcionamento, o sinal de entrada amostrado é dividido em duas sub-bandas de iguais larguras, usando-se bancos de filtro de quadratura (QMF, do inglês *"Quadrature Mirror Filter"*). Os filtros de sub-bandas $h_{low}(n)$ e $h_{high}(n)$ satisfazem (SPANIAS; PAINTER; ATTI, 2007):

$$h_{high}(n) = (-1)^n h_{low}(n) \qquad \text{e} \qquad |H_{low}(z)|^2 + |H_{high}(z)|^2 = 1. \tag{17}$$

A sub-banda de baixa frequência é tipicamente quantizada em 48 kbits/s, enquanto a sub-banda de alta frequência é codificada em 16 kbits/s (SPANIAS; PAINTER; ATTI, 2007). O G.722 inclui um esquema adaptativo de alocação de bits e possui três modos de operação, possíveis graças à variação dos bits usados para representar o sinal de banda mais baixa (SPANIAS; PAINTER; ATTI, 2007): 64, 56 e 48 kbits/s, sendo necessário, para os dois últimos, um canal

secundário auxiliar, que opera a taxas de 8 e 16 kbits/s, totalizando 64 kbits/s (CAVALCANTI, 2009). O padrão G.722 é comumente usado como referência para comparação com outros codificadores (CHU, 2003).

4.4.3 G.726 - ADPCM

A recomendação G.726 do ITU-T, lançada em 1990, fornece um padrão que visa ser uma evolução do padrão G.711, convertendo um canal PCM, de lei-μ ou lei-A, que opera a 64 kbits/s, para um canal a 40, 32, 24 ou 16 kbits/s. A conversão é feita usando-se a técnica ADPCM (ITU-T, 1996). O G.726 incorpora três características em relação ao PCM do G.711: predição linear, quantização adaptativa e codificação diferencial. No algoritmo do G.726 também é incorporado um quantizador adaptativo e um preditor adaptativo com 2 pólos e 6 zeros. Os coeficientes do preditor são adaptados baseados no sinal de entrada (GOLDBERG; RIEK, 2000), levando-se em conta as características estatísticas, variantes no tempo, da voz (BARBOSA, 2008).

No codificador, o sinal de entrada PCM, de lei-μ ou lei-A, é convertido em um sinal PCM uniforme, subtraído de um sinal estimado, quantizado em uma resolução menor de bits e encaminhado para transmissão. Um quantizador inverso produz o sinal diferença novamente. Nele, o sinal estimado é adicionado ao sinal diferença para produzir a versão reconstruída do sinal de entrada. O sinal reconstruído e o sinal diferença são operados sobre um preditor adaptativo, produzindo o sinal de entrada estimado e completando o laço de realimentação.

No processo de decodificação, o sinal amostrado é reconstruído a partir da soma do sinal diferença com o sinal estimado pelo preditor adaptativo. Assim, após reduzir-se a taxa de transmissão com o uso do ADPCM, o sinal é novamente transformado em PCM na taxa de 64 kbits/s (padrão G.711).

O G.726 possui uma qualidade praticamente idêntica ao G.711, porém consumindo menos banda. Sua utilização perdeu espaço durante os anos 90, devido à sua incapacidade de trabalhar com sinais de modem e de fax com taxas maiores que 12 kbits/s. No entanto, por exigir pouco processamento, ainda desperta o interesse de quem implementa sistemas de telefonia e transmissão de voz (CHU, 2003).

4.4.4 G.728 - LD-CELP

O G.728, recomendação lançada em 1992 pelo ITU-T, estabelece um padrão de codificação de voz a 16 kbits/s, que utiliza um codificador de baixo atraso com predição linear excitada a código LD-CELP (do inglês *"Low-Delay Code-Excited Linear Prediction"*) (ITU-T, 1992). Neste padrão, a essência das técnicas de codificação CELP, análise por síntese e busca em um dicionário é mantida, no entanto uma adaptação de preditores é requerida para se conseguir um atraso de 0,625 ms. O G.728 é largamente utilizado para aplicações que requerem um algoritmo de baixo atraso (ITU-T, 1992).

Na codificação, após a conversão de PCM de lei-μ ou lei-A para PCM uniforme, o sinal de entrada é segmentado em blocos de cinco amostras consecutivas. Para cada bloco, o codificador analisa cada um dos 1024 vetores quantizados contidos no dicionário, sendo previamente tratados pelos blocos de ganho e filtro de síntese, com o intuito de encontrar aquele que minimize o erro médio quadrático. Realizado esses procedimentos, o sistema transmite o índice da palavra código escolhida, contendo 10 bits de informação. Essa palavra é, então, processada pela unidade de escalonamento e pelo filtro de síntese para estabelecer os novos parâmetros a serem utilizados na codificação do próximo bloco. Os coeficientes dos blocos de filtro de síntese e ganho são ajustados periodicamente de forma retroalimentada, baseados no sinal quantizado e no ganho do bloco anterior (CAVALCANTI, 2009). O padrão G.726, talvez, seja o codificador de baixo-atraso de maior sucesso disponível (CHU, 2003).

4.4.5 G.729 - CS-ACELP

Padronizado pelo ITU-T em 1996, o padrão G.729 é um codificador paramétrico ou *vocoder* (BARBOSA, 2008). Ele utiliza uma forma de codificação algébrica de estrutura conjugada com predição linear CS-ACELP (do inglês *"Conjugate Structure-Algebraic Code Excited Linear Prediction"*) e codifica os sinais de áudio em quadros de 10 ms cada, com um atraso de 15 ms, exigindo cerca de 20 MIPS para codificação e 3 MIPS para decodificação, com uma qualidade de 8 kbits/s (SALAMI et al., 1997).

Baseado no codificador CELP, o modelo opera com quadros de 10 ms, que correspondem a 80 amostras das 8.000 do PCM. Cada um dos quadros do sinal de voz é analisado para retirada dos parâmetros do modelo CELP. Estes parâmetros são codificados e transmitidos ao meio, totalizando 80 bits por quadro amostrado. O sistema utiliza o método da autocorrelação para calcular os coeficientes dos filtros, empregando janelas assimétricas deslizantes de 240 amostras de comprimento, deslocadas a cada 80 amostras (10 ms) (CHU, 2003). As janela assimétricas são definidas por

$$w[n] = \begin{cases} 0,54 - 0,46\cos\left(\frac{2\pi}{399}\right), & n = 0,\ldots,199, \\ \cos\left(\frac{2\pi(n-200)}{159}\right), & n = 200,\ldots,239. \end{cases} \tag{18}$$

O padrão G.729 foi concebido para transmitir sinais de voz com qualidade em ambientes onde baixas taxas de codificação são de extrema importância, como em aplicações de comunicação sem fio e circuitos transoceânicos. Apesar desse padrão ser extremamente eficiente em termos de taxa de codificação, seus requisitos computacionais são extremamente elevados. Dessa maneira, em maio de 1996 foi lançado o Anexo A da recomendação, mantendo a operabilidade com o G.729 original e reduzindo a sua complexidade computacional. Em outubro

desse mesmo ano, foi homologado o Anexo B do padrão. Tal anexo descreve o gerador de ruído de conforto e o detector de voz VAD, utilizados na implementação da compressão de silêncio nos padrões G.729 e G.729A.

4.5 Sumário dos Codificadores

Nesta seção, busca-se fazer um resumo dos codificadores apresentados na seção anterior. Como referência, a Tabela 4 resume as principais características de cada um desses codificadores. Deve ser observado que a coluna MOS denota uma pontuação subjetiva atribuída aos codificadores de voz, que pode variar de 1 a 5. Uma pontuação superior a 3 indica que o codificador é de boa qualidade.

Tabela 4 – Taxa de bits e pontuação MOS dos codificadores apresentados.

Codificador	Taxa de bits	MOS
G.711	64	4,4
G.722	64	4,5
G.726	40, 32, 24 e 16	4,3
G.728	16	4,2
G.729	8	3,9

Fonte: Produzido pelo autor.

5 UM NOVO PADRÃO DE CODIFICAÇÃO DE VOZ

5.1 Introdução

Este capítulo é reservado ao sistema de codificação (*vocoder*) desenvolvido neste trabalho, cuja proposta é reunir simplicidade de implementação, baixa complexidade computacional, taxa de bits inferior aos tradicionais *vocoders* existentes e qualidade aceitável dos sinais de voz produzidos. Este novo padrão de codificação de voz é baseado no Mascaramento Pleno em Frequência por Oitava (MPFO), técnica publicada em (SOTERO FILHO; DE OLIVEIRA, 2009) e (SOTERO FILHO; DE OLIVEIRA; CAMPELLO DE SOUZA, 2014), e que é fundamentada no mascaramento auditivo em frequência, fenômeno psicoacústico já discutido no Capítulo 2. O MPFO atua no espectro de magnitude em frequência de um sinal de voz, descartando amostras potencialmente mascaradas por outras vizinhas de maior amplitude. A técnica é mais detalhada durante a Seção 5.4

Neste capítulo, também, apresenta-se um método de preenchimento espectral via distribuição beta de probabilidade, desenvolvido com o intuito de aperfeiçoar o estágio de síntese do *vocoder* proposto. Este procedimento adiciona informação ao espectro simplificado pelo MPFO, numa tentativa de aproximá-lo ao seu formato original e, dessa forma, obter melhorias na inteligibilidade e no aspecto metalizado dos sinais de voz sintetizados.

Adicionalmente, introduz-se um inédito formato binário para armazenamento de sinais de voz, o qual é representado pela extensão *voz*. Deve-se a esta nova representação a eficiente compactação dos sinais fornecidos pelo sistema.

Ainda, são apresentados resultados de simulações para o *vocoder* proposto e exibidas classificações, a partir de teste subjetivo ACR, da qualidade de alguns sinais de voz por ele gerados.

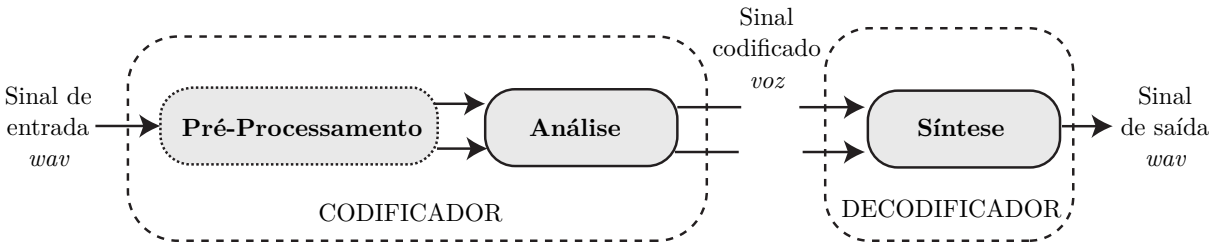
5.2 Visão Geral do Sistema

Em geral, um sistema de codificação de voz é constituído por 3 estágios principais: pré-processamento, análise e síntese do sinal de voz. Estes estágios compõem os seus 2 subsistemas principais: codificador e decodificador.

No estágio de pré-processamento, o sinal de entrada é submetido a etapas de pré-tratamento, a fim de permitir uma representação adequada para o estágio de análise. Estas operações incluem a divisão do sinal, no domínio do tempo, em segmentos menores (segmenta-

ção) e a suavização destes através de um janelamento. O estágio de análise descreve as operações sofridas pelo sinal, a fim de alcançar um resultado final: uma representação no domínio da frequência do sinal ao longo do tempo. O último estágio do *vocoder* é a síntese (ou, mais precisamente, a ressíntese) do sinal de voz no domínio do tempo. A síntese descreve o processo envolvido na criação de um novo sinal, no domínio do tempo, a partir dos segmentos, no domínio da frequência, obtidos durante a fase de análise.

Figura 21 – Diagrama de blocos simplificado do *vocoder* proposto.



Fonte: Produzido pelo autor.

No sistema de codificação proposto, tanto o codificador quanto o decodificador contém técnicas inovadoras desenvolvidas no Departamento de Eletrônica e Sistemas da UFPE. A Figura 21 apresenta um diagrama de blocos simplificado desse sistema. A descrição detalhada é dada nas próximas seções.

5.2.1 Implementação do Sistema

O sistema de codificação de voz proposto neste trabalho foi desenvolvido através de programas elaborados no MATLAB®. A escolha por essa linguagem de implementação foi influenciada pelos seguintes fatores:

i. bastante difundida no meio acadêmico;

ii. ferramenta matemática completa disponível em qualquer plataforma de computação;

iii. sua sintaxe baseada em vetores esconde a maioria das complexidades não-importantes.

Os algoritmos implementados para o sistema tiveram a intenção de mesclar interatividade com o usuário, através da interface gráfica do MATLAB®, e facilidade para simulações e testes de suas rotinas. A Tabela 5 exibe uma listagem dos algoritmos utilizados para a implementação do *vocoder* e suas respectivas funções. Os códigos fonte deles estão disponibilizados no Anexo A.

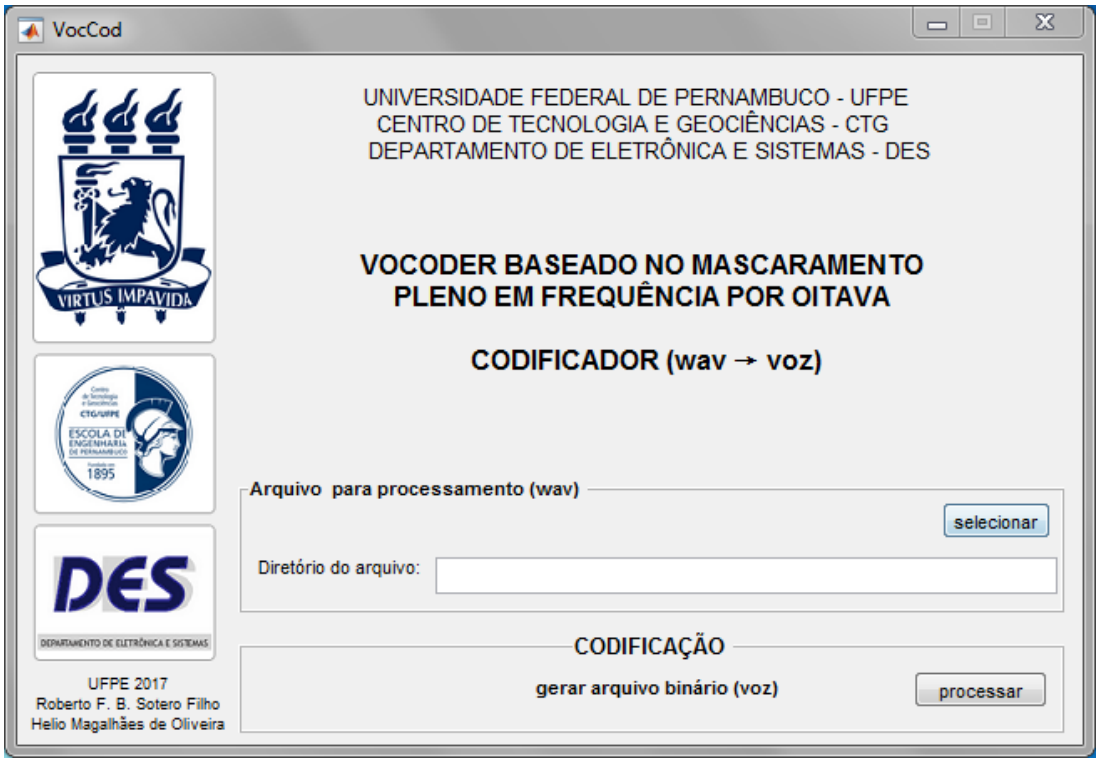
Tabela 5 – Algoritmos do sistema de codificação de voz.

Algoritmo	*Função*
VocCod	Codificador (*wav* → *voz*)
VocDec	Decodificador (*voz* → *wav*)
SpecStuffing	Preenchimento Espectral via Distribuição Beta
SpecStuffWindow	Preenchimento Espectral com Janelamento Extra

Fonte: Produzido pelo autor.

Como visto da tabela, o sistema é composto de dois algoritmos principais, um para o codificador (análise), **VocCod**, e outro para o decodificador (síntese), **VocDec**; e dois algoritmos secundários, **SpecStuffing** e **SpecStuffWindow**, que são chamadas pelo decodificador. O software de análise foi projetado para trabalhar com arquivos de voz do tipo *wav*, extensão padrão para arquivos de áudio do Windows®. Na sua interface gráfica, Figura 22, o usuário pode selecionar quaisquer arquivos *wav* que deseja codificar, ficando a cargo das rotinas internas do programa a conversão das taxas de amostragem e bits de resolução desses arquivos para 8 kHz e 16 bits respectivamente, atributos necessários para o seu adequado funcionamento. Nele também estão contidas as rotinas internas de pré-processamento (segmentação de 20 ms e janelamento de Hamming sem superposição).

Figura 22 – Imagem da interface gráfica do codificador.



Fonte: Produzido pelo autor.

Na rotina principal do **VocCod**, os arquivos *wav* são processados pelo método do MPFO, produzindo vetores representativos do sinal, que são quantizados e codificados para o novo padrão binário de codificação *voz*, detalhado durante a subseção 5.4.4. Este procedimento é realizado através da interface gráfica do programa ao se pressionar o botão "processar". O usuário pode, também, salvar o arquivo *voz* com o nome que desejar. As etapas realizadas pelo **VocCod** são dadas abaixo:

i. leitura dos arquivos de entrada *wav*;

ii. conversão das taxas de amostragem e bits de resolução para 8khz e 16bits, respectivamente;

iii. aplicação da FFT para quadros janelados de 20 ms com a janela de *Hamming* sem superposição;

iv. processamento pela técnica do MPFO;

v. quantização e codificação das amostras de mascaramento e suas posições de ocorrência no espectro;

vi. reunião das informações extraídas do sinal pelo estágio (iv), codificação e armazenamento no formato binário *voz*.

Na síntese do sinal, o software de decodificação **VocDec**, que aceita como entrada apenas arquivos com extensão binária *voz* produzidos pelo software de codificação **VocCod**, extrai as informações codificadas e, através de suas rotinas específicas, remonta o espectro simplificado para cada um dos quadros de voz segmentados na análise. O estágio, também, faz uso de uma técnica de preenchimento espectral baseada na distribuição beta de probabilidade, que é introduzida no decodificador através de dois algoritmos: **SpecStuffing**, para o preenchimento espectral e **SpecStuffWindow**, para o preenchimento espectral com um janelamento de Hamming extra.

A interface gráfica do software de decodificação **VocDec** é exibida na Figura 23. Nela, o usuário pode inserir os arquivos *voz* gerados pelo codificador e escolher entre 4 tipos de recomposição do sinal: (i) utilizando apenas o MPFO, (ii) MPFO acrescido da técnica de preenchimento espectral, (iii) utilizando uma composição dos sinais em (i) e (ii), e (iv) utilizando sinais de (ii) com um janelamento de Hamming extra. Estes procedimentos oferecem ao usuário a possibilidade de avaliar a qualidade dos 4 sinais gerados e optar por aquele que mais o agradou.

Após escolher o tipo de recomposição, o usuário deve clicar o botão "processar" para que o software inicie a decodificação do arquivo de entrada. Os quadros são concatenados e reconvertidos para o formato *wav*, estando aptos a serem reproduzidos. Ao finalizá-la, é aberta uma janela para que o usuário salve o arquivo *wav* com o nome que desejar.

Figura 23 – Imagem da interface gráfica do decodificador.

Fonte: Produzido pelo autor.

As etapas realizadas pelo algoritmo de síntese são dadas abaixo:

i. leitura dos arquivo de entrada *voz*;

ii. remontagem dos espectros do sinal sintetizado a partir das informações binárias de entrada;

iii. introdução do preenchimento espectral para cada quadro de espectro da voz (dependendo da opção escolhida para o processamento);

iv. aplicação da IFFT para cada um desses quadros;

v. concatenação dos quadros de voz;

vi. reconversão do sinal para o formato *wav*, para fins de reprodução.

As próximas seções trazem a descrição detalhada do sistema proposto.

5.3 Pré-Processamento do Sinal

O primeiro estágio do *vocoder* proposto consiste em um pré-processamento do sinal lido de um arquivo *wav*. Este procedimento é necessário em processamento de voz, visto que as características desses sinais têm algumas peculiaridades que precisam ser previamente tratadas.

Como o sistema desenvolvido é exclusivo para sinais de voz, que possuem a maior parte de sua energia concentrada numa faixa limitada de frequências, faz-se necessário utilizar uma taxa de amostragem que leve em consideração essa característica. Além disso, deve-se respeitar a condição imposta pelo teorema da amostragem (DE OLIVEIRA, 2007) para que não haja perda de informação: a frequência de amostragem deve ser maior que o dobro da maior frequência contida no sinal a ser amostrado. A partir desses requisitos, e considerando a máxima frequência do sinal de voz igual a 4 kHz, adota-se 8 kHz como taxa de amostragem para o sistema.

5.3.1 Segmentação da Voz

A segmentação consiste em particionar o sinal de voz em blocos de duração definida, cujo tamanho é escolhido dentro dos limites de estacionariedade do sinal (RABINER; SCHAFER, 1978). Um sinal é dito estacionário quando suas características estatísticas não variam com o tempo (LATHI, 1989). Uma vez que a voz é um sinal de natureza aleatória e sabendo-se que o trato vocal muda muito lentamente na voz contínua, muitas partes da onda acústica podem ser assumidas como estacionárias num intervalo de curtíssima duração (entre 10 e 40 ms) (OPPENHEIM; SCHAFER; BUCK, 1999). No trabalho, são utilizados segmentos de 20 ms, valor usualmente utilizado para processamento de sinais de voz.

A segmentação é, portanto, a operação de multiplicar intervalos sucessivos do sinal de voz no domínio do tempo, por um pulso retangular contendo um número definido de amostras e que pode ser referido como uma janela retangular. Uma vez que a multiplicação no domínio do tempo equivale à convolução no domínio da frequência, a janela retangular produz uma distorção do sinal, no domínio da frequência, que deve estar dentro dos limites toleráveis da aplicação. Assim, para suavizar a sobreposição e evitar efeitos de distorção intoleráveis, são utilizados janelas especiais para realizar a segmentação, tais como as que serão descritas a seguir.

5.3.2 Janelamento

A utilização do janelamento é uma forma de se conseguir aumentar as informações espectrais de um sinal amostrado (SILVA, 2006). Esse "aumento" de informação é decorrente da minimização das margens de transição em forma de ondas truncadas e de uma melhor separação do sinal de pequena amplitude de um sinal de grande amplitude, com frequências muito próximas umas das outras. Algumas janelas comumente utilizadas são descritas em seguida. Todas elas estão, também, disponíveis em funções internas do MATLAB®.

- Janela Retangular: Essa janela apenas particiona o sinal em blocos consecutivos de mesmo tamanho. Sua formulação é dada por

$$w[n] = \begin{cases} 1, & 0 \leq n \leq M, \\ 0, & \text{caso contrário,} \end{cases} \tag{1}$$

em que $M + 1$ é o tamanho da janela.

- Janela de *Hamming*: Proporciona a manutenção das características espectrais do centro do quadro e a eliminação das transições abruptas das extremidades (SILVA, 2006). Esta janela é representada por

$$w[n] = \begin{cases} 0,54 - 0,46 \cos\left(\dfrac{2\pi n}{M}\right), & 0 \leq n \leq M, \\ 0, & \text{caso contrário.} \end{cases} \tag{2}$$

- Janela de *Hanning*: Apresenta características parecidas com a janela de *Hamming*, entretanto ela gera um reforço maior nas amostras centrais e uma suavização maior nas amostras das extremidades. Esta janela é formulada por

$$w[n] = \begin{cases} 0,5 - 0,5 \cos\left(\dfrac{2\pi n}{M}\right), & 0 \leq n \leq M, \\ 0, & \text{caso contrário.} \end{cases} \tag{3}$$

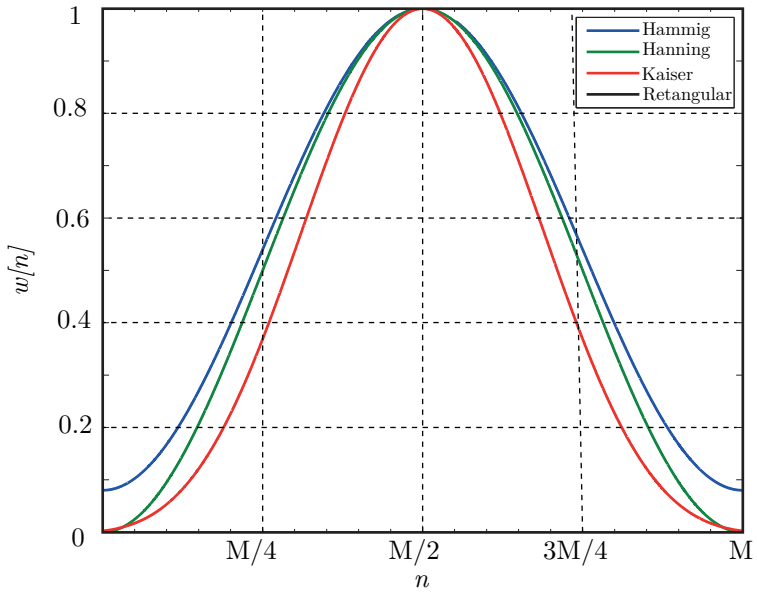
- Janela de *Kaiser* - É uma janela mais flexível que as demais. Através do ajuste do parâmetro β, sua forma pode ser modificada. Dessa forma, dependendo da aplicação, pode-se modificar a forma da janela para controlar a perda espectral. Sua equação é expressa por

$$w[n] = \begin{cases} \dfrac{I_0\left(\beta\sqrt{1 - \left(\dfrac{n-\alpha}{\alpha}\right)^2}\right)}{I_0(\beta)}, & 0 \leq n \leq M, \\ 0, & \text{caso contrário,} \end{cases} \tag{4}$$

em que $\alpha = M/2$, e $I_0(\cdot)$ representa a função de Bessel modificada de ordem zero de primeiro tipo (OPPENHEIM; SCHAFER; BUCK, 1999).

As janelas definidas anteriormente são comumente usadas para análise espectral e também para projeto de filtros de resposta ao impulso finita (FIR) (OPPENHEIM; SCHAFER; BUCK, 1999). As formas no domínio do tempo e no domínio da frequência dessas janelas são mostradas, respectivamente, nas Figuras 24 e 25.

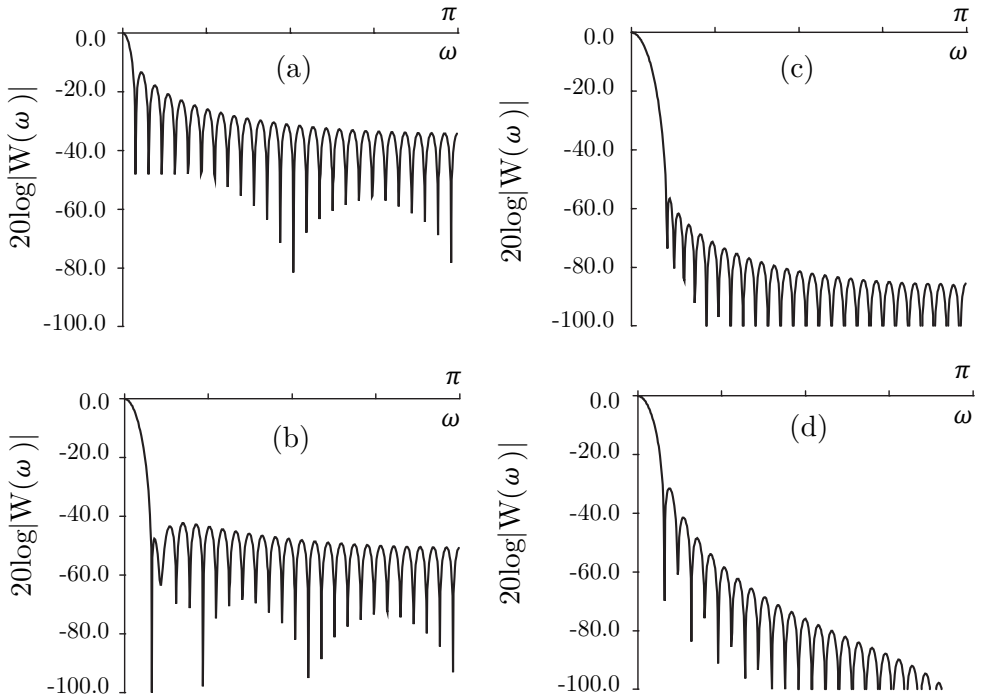
Figura 24 – Ilustração da forma das janelas citadas.



Fonte: Produzido pelo autor.

Dois efeitos ocorrem como resultado da aplicação das janelas no sinal: resolução em frequência reduzida e escoamento espectral (*leakage*). A resolução é influenciada pela largura do lobo principal, enquanto que o grau de escoamento depende do quociente entre a amplitude máxima do lobo principal e a amplitude máxima do primeiro lobo lateral. Uma boa resolução temporal requer uma janela de curta duração. Em contrapartida, uma boa resolução em frequência requer uma janela de longa duração, com um lobo principal mais estreito. Uma vez que a atenuação da janela é essencialmente independente da duração da janela, aumentando-se o valor de M, diminui-se a largura de seu lobo principal (KONDOZ, 2004).

Figura 25 – Gráfico da resposta em frequência das funções janela descritas: (a) Retangular, (b) Hamming, (c) Hanning, (d) Kaiser $\beta = 7.8$.



Fonte: Adaptado de Kondoz (2004).

Como se pode observar da Figura 25, o lobo principal da janela retangular é quase que metade do lobo principal das janelas de *Hamming* e *Hanning*. Entretanto, os lobos laterais dessas últimas são bem menores que os da retangular. O primeiro lobo lateral da janela de *Hanning* é aproximadamente 20 dB maior do que o da de *Hamming*, mas os lobos seguintes diminuem mais rapidamente do que os da janela de *Hamming* (SILVA, 2006).

O alto escoamento espectral produzido pelos largos lobos laterais faz o janelamento através da janela retangular parecer mais ruidoso para a voz, dificultando a discriminação de componentes de baixa amplitude. Essa indesejável característica entre harmônicos adjacentes tende a minimizar os benefícios de uma resposta no domínio do tempo plana (maior resolução em frequência) para a janela retangular. Já nas janelas de *Hamming* e *Hanning*, os lobos secundários menores permitem uma melhor detecção desses componentes (KONDOZ, 2004). A janela de *Kaiser*, por possuir um parâmetro de ajuste, permite quantificar o compromisso entre a largura do lobo principal e a atenuação do lobo lateral.

Uma forma de se aumentar a correlação entre janelas sucessivas é através da superposição de janelas, evitando variações bruscas entre as características extraídas de janelas adjacentes (RABINER; SCHAFER, 1978). Entretanto, esta superposição requer um maior complexidade de implementação, não sendo possível sua implementação para esse trabalho.

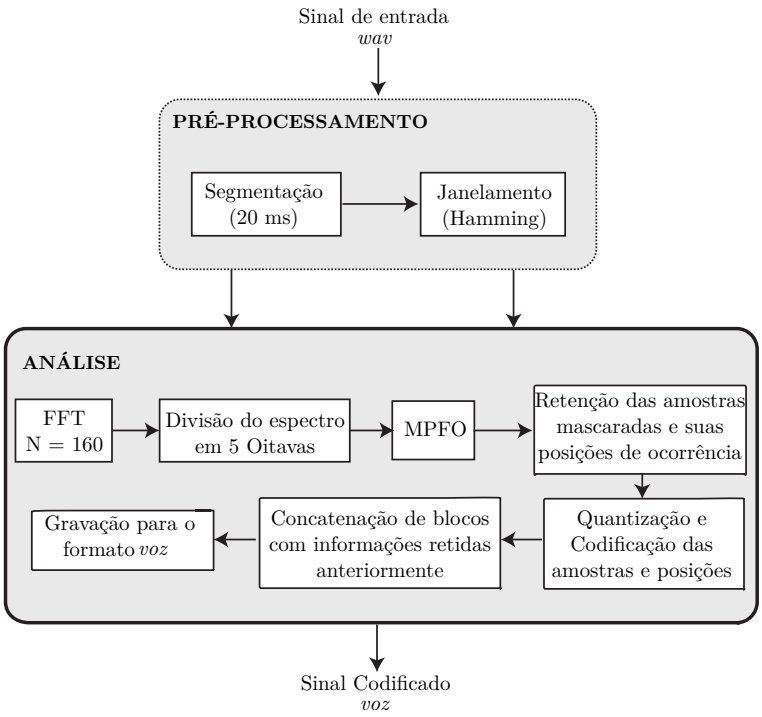
Diante de todos esses fatos, a escolha da janela a ser utilizada é, então, uma questão de avaliação da relação custo-benefício entre um lobo principal estreito e lobos laterais pequenos. No trabalho proposto, o janelamento de *Hamming* (com janelas de 20 ms), sem superposição, é o escolhido por apresentar características espectrais interessantes e suavidade nas bordas (CHU, 2003) (OPPENHEIM; SCHAFER; BUCK, 1999).

Terminado o pré-processamento do sinal, passa-se ao estágio de análise dos sinais de voz através da técnica do MPFO. O procedimento é abordado a seguir.

5.4 Análise da Voz pela Técnica do MPFO

O estágio de análise é o responsável por introduzir a técnica do MPFO sobre o sinal de voz, a fim de se obter um espectro simplificado do sinal e reduzir a taxa de bits necessária para o codificação. A *priori*, o sistema identifica casos de mascaramento em cada uma das oitavas. O estágio descarta sinais que "não seriam audíveis" devido a este fenômeno auditivo e despreza, por completo, a fase do sinal. A escolha pelo particionamento do espectro em oitavas tem haver com as bandas críticas da audição, que também possuem larguras de banda não-uniformes. O intuito foi investigar o efeito das amostras de mascaramento sobre as mascaradas dentro de uma oitava. A Figura 26 exibe o diagrama detalhado do codificador do *vocoder* proposto.

Figura 26 – Diagrama de blocos detalhado do codificador proposto.



Fonte: Produzido pelo autor.

5.4.1 Características Psicoacústicas do Sistema Auditivo Humano

Como na maioria dos sistemas de codificação de voz eficientes, os *vocoders* aproveitam-se de certas propriedades auditivas do sistema auditivo humano para reduzir as taxas de bits. A técnica aqui proposta, além de tomar como base para sua implementação o mascaramento em frequência, aproveitou-se também da insensibilidade do ouvido humano em relação à fase do som.

5.4.1.1 Insensibilidade à Fase do Som

O ouvido humano apresenta mais uma característica peculiar em relação à sua percepção auditiva: praticamente não consegue distinguir diferenças em relação à fase de um sinal sonoro. O processo pode ser explicado examinando-se como um som se propaga por um ambiente. Qualquer som que se propaga e que chega aos nossos ouvidos atravessa diversos obstáculos e percorre caminhos distintos. Parte do som chega defasada das demais, mas tal diferença é minimamente sentida pelos ouvidos (OPPENHEIM; SCHAFER; BUCK, 1999). A informação na voz humana é principalmente concentrada nas amplitudes das frequências. Desta forma, diversos codificadores de voz desconsideram a fase do sinal no estágio de análise, considerando esta percepção como desprezível. Baseado nesse fato, o *vocoder* proposto descartará as características de fase do espectro, considerando apenas a amplitude das amostras.

5.4.2 Simplificação do Espectro Via MPFO

De posse dos sinais pré-processados, pode-se iniciar a etapa da análise propriamente dita do sinal. O procedimento é descrito a seguir.

Para cada segmento do sinal de voz janelado, aplica-se uma FFT de comprimento 160 (número de amostras contidas em um quadro de 20 ms de voz), obtendo-se sua representação no domínio frequencial. A partir daí, divide-se a região do espectro da magnitude do sinal em zonas de influência (oitavas). Devido a frequência mínima dos sinais de voz ser acima dos 300 Hz, a faixa de frequências entre 32 e 256 Hz não é considerada na análise, já que não possui informação prática. A primeira oitava utilizada corresponderá à faixa de frequências de 256 Hz–512 Hz, a segunda cobrindo a banda de 512 Hz–1024 Hz, e assim por diante. A quarta (última oitava) irá corresponder à faixa de 2048 Hz–4000 Hz (notando-se que em 4 kHz o espectro simétrico, produzido pela FFT, começa a se repetir). A Tabela 6 exibe essa divisão.

Como se está fazendo uso de uma taxa de amostragem de 8 kHz, cada amostra da magnitude do espectro corresponderá a uma amostra espectral múltipla de 50 Hz, sendo que a primeira amostra irá representar a componente DC de cada quadro de voz [6]. Como essa amostra tem pouca informação útil, já que representa apenas um offset do sinal, será prontamente desconsiderada da análise. Já que as raias espectrais caminham a passos de 50 Hz, a primeira oitava (de 256 Hz a 512 Hz), será representada pela amostras espectrais de 300, 350, 400, 450 e

Tabela 6 – Número de frequências estimadas pela DFT de comprimento 160 em cada oitava do espectro vocal.

Oitavas (Hz)	**# amostras espectrais/oitava**
256 - 512	5
512 - 1024	10
1024 - 2048	20
2048 - 4096	39
total	74

Fonte: Produzido pelo autor.

500 Hz, a segunda oitava (512 Hz a 1024 Hz) pelas amostras de 550 Hz, 600, 650, 700,..., 1000 Hz, seguindo-se o mesmo raciocínio para as demais. Terminado esse procedimento inicial de divisão do espectro em oitavas e de descarte de amostras espectrais sem relevância, passa-se agora a buscar em cada oitava, em todas as quatro sub-bandas do sinal de voz, o ponto da FFT de maior magnitude, i.e., aquele que irá (potencialmente) mascarar os demais. Essa amostra espectral passará a ser o único representante dentro de cada oitava (por opção, para reduzir a complexidade). As demais serão desconsideradas, assumindo valor espectral nulo. O algoritmo guarda em que posição no espectro de frequências o tom de mascaramento ocorreu, procedimento útil para a correta remontagem do espectro pelo sintetizador. O total de 79 frequências oriundas da estimativa da DFT é reduzido para 4 sobreviventes em adição às suas 4 respectivas posições de ocorrência (retendo pouco mais do que 10% das componentes espectrais). Para cada quadro de voz, as amostras sobreviventes e as posições de ocorrência serão quantizadas e codificadas separadamente, sendo salvas no formato binário *voz*, cuja composição será explicada mais adiante. Os procedimentos de quantização e codificação serão vistos em seguida.

5.4.3 Quantização e Codificação dos Sinais de Voz

Na quantização dos quadros de voz utilizou-se o método mais comum de quantização, a quantização uniforme. Para simplificar a codificação binária, fez-se uso de um número de níveis coincidente com uma potência de 2, de modo a otimizar o número n de bits de codificação por amostra. A máxima excursão do sinal (no caso o tom de maior magnitude do espectro completo do sinal de voz) foi assim dividida em 256 intervalos de mesmo comprimento, sendo cada um deles representado por uma única palavra-código de 8 bits. Como não há amostras negativas a serem quantizadas, já que a magnitude do espectro não apresenta valores negativos, o quantizador utilizado não pode ser bipolar, sendo necessário uma reconfiguração apropriada dos níveis. Uma rotina do software de análise **VocCod** foi especificamente projetada para esse fim. A quantização das posições não foi necessária, visto que elas são números inteiros.

Com a finalidade de se reduzir o número de bits necessário à codificação dos quadros

de voz, o algoritmo de alocação dos bits levou em consideração o comprimento da oitava. À medida que se diminui o número da oitava, reduz-se pela metade a faixa de frequências que ela cobre e, dessa forma, um menor número de bits é necessário para a adequada codificação das posições em que as amostras espectrais de mascaramento ocorreram. Essas posições, em oitavas sucessivas (caminhando no espectro em relação às altas frequências), necessitam de mais um bit pra a sua correta representação. Por exemplo, um tom de mascaramento que acorra na primeira oitava (256–512 Hz), tem 5 possíveis posições de ocorrência (posição 7 à posição 11), todas elas podendo ser completamente representadas por uma palavra-código de 4 bits. Já na oitava seguinte (512–1024 Hz) a posição máxima em que o tom de mascaramento pode ocorrer ($21^{\underline{a}}$ – que corresponde a frequência de 1 kHz) pode ser codificada por um palavra-código de 5 bits. Nas duas oitavas subsequentes, as posições máximas são as posições 41 (descrita por uma palavra de 6 bits) e 80 (representada por uma palavra de 7 bits), respectivamente. Para os valores máximos das amplitudes das amostras espectrais de mascaramento são reservados 8 bits (1 byte) para a sua representação, já que foram escolhidos 256 níveis de quantização. O número de bits alocados para cada um desses parâmetros é mostrado na Tabela 7. Como já discutido antes, a fase do sinal não é considerada.

Tabela 7 – Alocação dos bits para um quadro de voz de 20 ms.

Oitava	**Parâmetro**	**bits**
1	amostra espectral de mascaramento	8
	posição da amostra de mascaramento	4
2	amostra espectral de mascaramento	8
	posição da amostra de mascaramento	5
3	amostra espectral de mascaramento	8
	posição da amostra de mascaramento	6
4	amostra espectral de mascaramento	8
	posição da amostra de mascaramento	7
total		**54**

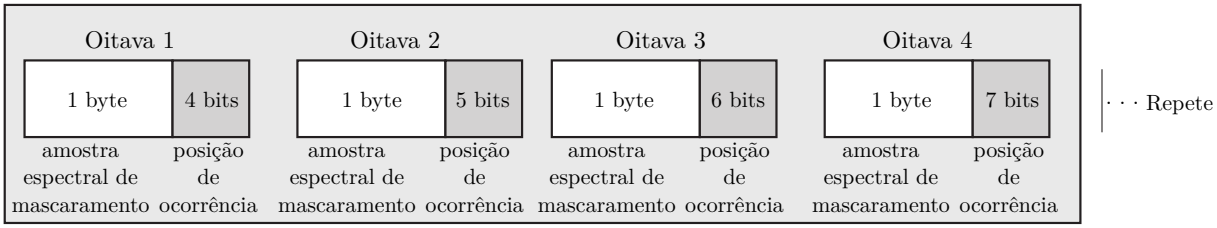
Fonte: Produzido pelo autor.

Vê-se que para cada quadro de voz de 20 ms, são necessário apenas 54 bits (22 para as posições e 32 para os valores dos tons de mascaramento), levando a uma taxa de $\frac{54}{20 \times 10^{-3}}$ bits/s ou 2,7 kbits/s.

5.4.4 Formato Binário *voz*

A alocação dos bits para cada quadro de voz, sumarizada na Tabela 7, sugere que se pense numa representação binária de armazenamento. Foi então proposto o formato *voz*, que é constituído pela concatenação de todos os quadros codificados. O conceito de um quadro deste formato é apresentado na Figura 27.

Figura 27 – Ilustração da configuração de um quadro de 20 ms do formato binário *voz*



Fonte: Produzido pelo autor.

Vê-se que os 54 bits são distribuídos em 4 sub-blocos (um para cada oitava utilizada) compostos pelo valor da amostra de mascaramento associada à sua respectiva posição no espectro. A concatenação de vários desses quadros forma o formato binário *voz*. Os arquivos de voz de entrada, antes no formato *wav*, são todos convertidos para esse novo padrão através do software de codificação **VocCod**.

O software de decodificação **VocDec**, que possui uma rotina de recomposição do espectro sintetizado projetada para reconhecer início e término de um novo quadro de 20 ms, consegue recuperar o sinal de voz sintetizado, transformando-o novamente para o formato *wav*. A seção seguinte aborda este processo.

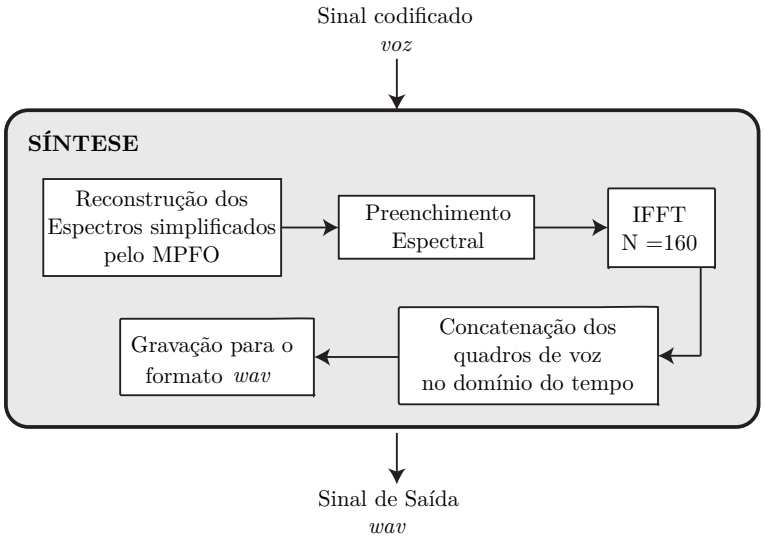
5.5 Síntese da Voz

O objetivo do estágio de síntese é reconstruir os sinais de voz codificados pelo estágio de análise. Para que isso seja possível, é necessário que os dados alimentados ao sintetizador estejam codificados no formato binário *voz*. O algoritmo, então, processa esses dados, identifica o início e o final de cada quadro de voz e, a partir daí, reconstrói o espectro sintetizado através dos valores de suas amostras espectrais e de suas respectivas posições de ocorrência no espectro.

No estágio de análise emprega-se o MPFO para simplificar o espectro de cada quadro de voz. Tal procedimento, apesar de conseguir manter uma voz inteligível na recuperação do sinal, resulta numa configuração de amostras muito isoladas e espaçadas no espectro. Visando aprimorar essa representação, melhorando a inteligibilidade e a característica metalizada da voz gerada, e suavizando a abrupta transição entre amostras em oitavas adjacentes, atribuiu-se ao sintetizador a técnica do preenchimento espectral via distribuição beta. Cada oitava possui uma

distribuição beta particular, que são atualizadas a cada novo quadro de voz. O pico (ou a moda) - descrita mais adiante - de cada uma dessas distribuições é definido como sendo igual ao valor da amostra sobrevivente da simplificação pelo MPFO. A Figura 28 exibe o diagrama de blocos detalhado do estágio de síntese (decodificador) do *vocoder* proposto.

Figura 28 – Diagrama de blocos detalhado do estágio de síntese (decodificador).



Fonte: Produzido pelo autor.

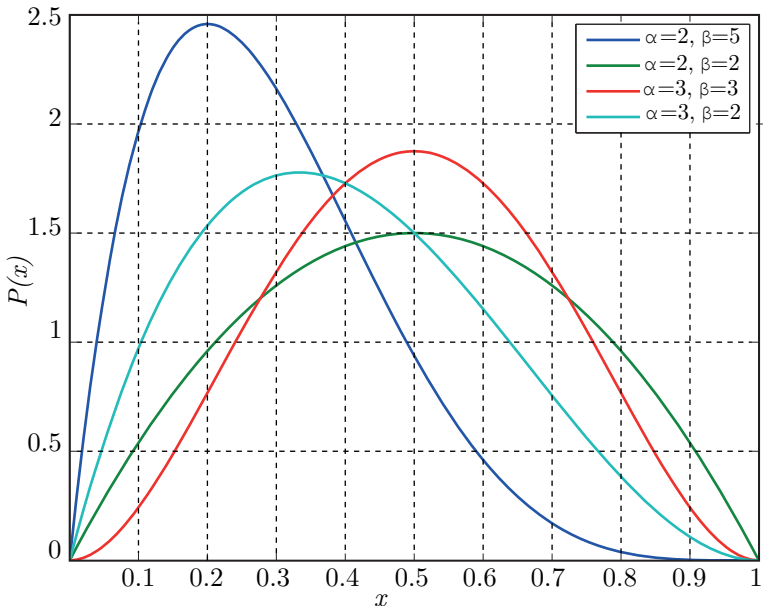
5.5.1 Preenchimento Espectral via Distribuição Beta

A distribuição beta é uma distribuição de probabilidade contínua definida no intervalo $0 \leq x \leq 1$, sendo caracterizada por um par de parâmetros α e β , de acordo com a equação (ANDRADE, 2007):

$$P(x) = \frac{1}{B(\alpha,\beta)} x^{\alpha-1}(1-x)^{\beta-1}, \qquad 1 < \alpha, \qquad \beta < \infty, \tag{5}$$

em que $B(\alpha,\beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}$ é um fator de normalização, $\Gamma(\cdot)$ é a função gama e $B(\cdot,\cdot)$ é a função Beta. A Figura 29 ilustra esta distribuição para alguns valores de α e β.

Figura 29 – Ilustração da curva de distribuição Beta para alguns valores de α e β.



Fonte: Produzido pelo autor.

O ponto da curva onde ocorre a máxima distribuição de probabilidade é chamado de moda, e o seu valor é obtido pela seguinte expressão:

$$\text{moda} = \frac{\alpha - 1}{\alpha + \beta - 2}. \tag{6}$$

5.5.1.1 Metodologia Empregada

Para correta implementação do método do preenchimento espectral, faz-se necessário manipular a expressão original da distribuição beta de forma que o seu intervalo de representação, que é definido apenas para o intervalo [0,1], passe a conter o limite de transição entre uma oitava e sua vizinha. É necessário, também, adotar o valor da moda da distribuição como sendo o valor da amostra sobrevivente por oitava.

Partindo-se da expressão original da distribuição beta dada pela Equação (5), faz-se uma expansão da curva de tal maneira que o limite superior seja correspondente a diferença entre as frequências de corte normalizadas superior (f_M) e inferior (f_m) de cada oitava, isto é, $f_M - f_m$. Essas frequências de corte precisam ser normalizadas, uma vez que o limite de frequências das oitavas (256–512 Hz, 512–1024 Hz, etc) não são múltiplos de 50 Hz, que é o valor de passo do espectro para a taxa de amostragem utilizada (8 kHz). Posteriormente, desloca-se a curva de um valor f_m de forma que os limites inferior e superior sejam f_m e f_M, respectivamente. Ao realizar-se esses procedimentos, é necessário também ajustar o valor da moda, que passa a ser

representada por

$$\text{novamoda} = \frac{\alpha - 1}{\alpha + \beta - 2}(f_M - f_m) + f_m. \tag{7}$$

O novo valor da moda deve corresponder ao valor da amostra predominante normalizada resultante das oitavas (f_c), ou seja:

$$\text{novamoda} = \frac{\alpha - 1}{\alpha + \beta - 2}(f_M - f_m) + f_m = f_c. \tag{8}$$

A partir dessa expressão e após algumas manipulações, obtém-se uma relação entre α e β expressa pela Equação (9) e que será útil na representação da nova expressão da distribuição.

$$\beta - 1 = (\alpha - 1).Q, \tag{9}$$

em que $Q = \dfrac{f_M - f_m}{f_c - f_m}$.

A expressão final, àquela utilizada no algoritmo de preenchimento espectral em cada quadro, é dada por

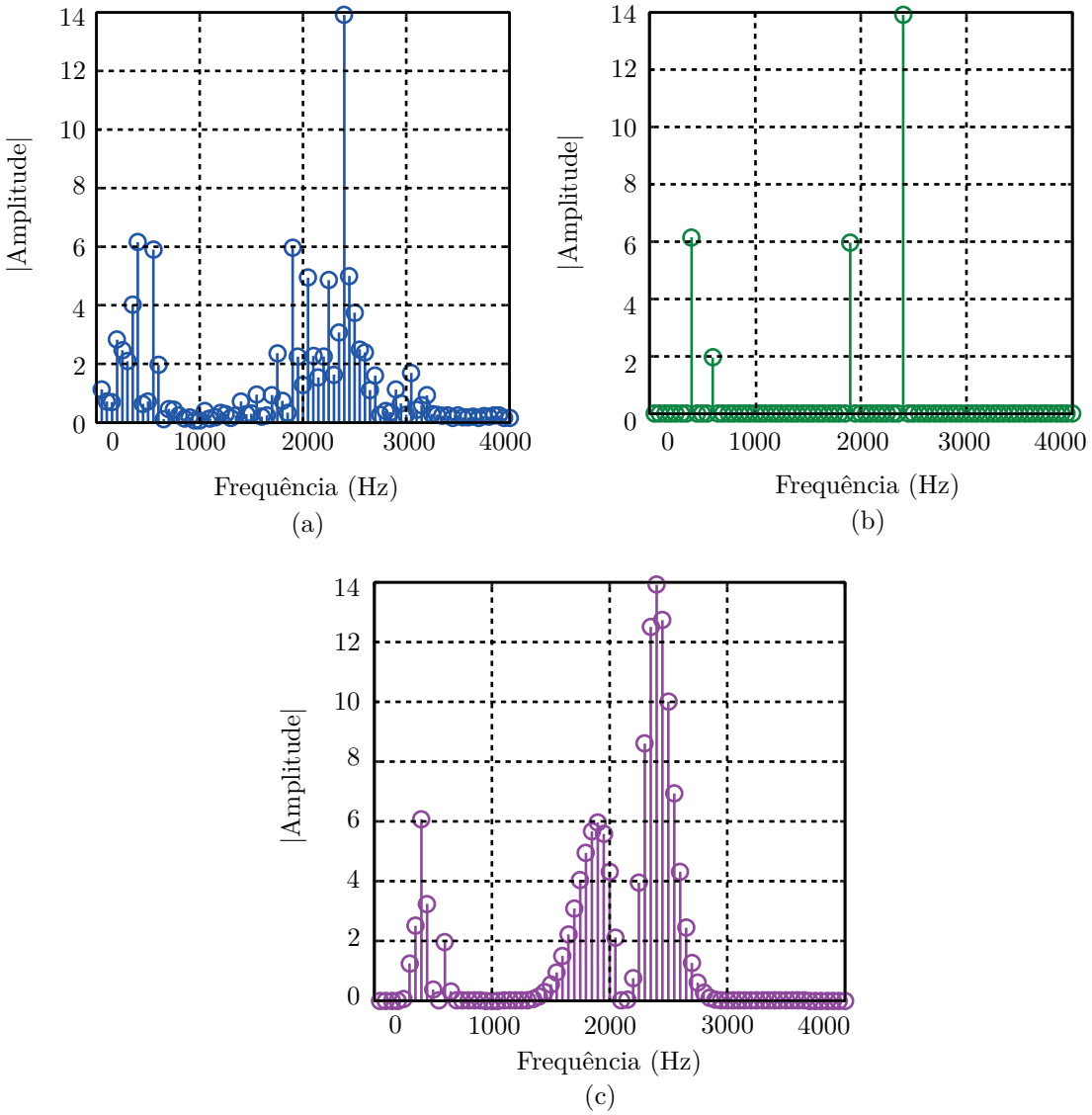
$$P(x) = \left(\frac{1}{f_M - f_m}\right)^{\alpha+\beta-2} (x - f_m)^{\alpha-1}(f_M - x)^{\beta-1}. \tag{10}$$

O valor de α, na expressão, é um parâmetro de expansão ou compressão da curva. Quanto maior o seu valor mais estreita ela se torna. Os valores de α usados foram variados de oitava para oitava, adequando-os para uma melhor qualidade da voz sintética. A partir de testes, os valores de α que mais se adequaram foram: $\alpha = 4$ para a primeira e segunda oitava, $\alpha = 7$ para a terceira e quarta oitava.

Dada a simetria dos espectros gerados pela FFT, faz-se necessário preencher as metades espelhadas dos espectros para a correta recomposição dos sinais; ou um espelhamento para reduzir a complexidade do algoritmo. Se qualquer um desses procedimentos não forem realizados, sinais temporais de natureza complexa são gerados incorretamente.

A Figura 30 ilustra a magnitude do espectro de um quadro de sinal de voz teste, representada antes (a) e depois (b) do processo de simplificação pelo MPFO; por último, representada com o preenchimento espectral via distribuição beta (c).

Figura 30 – Representação do espectro obtido pela FFT de um quadro teste de voz, para três situações distintas: (a) Espectro do sinal original, (b) Espectro simplificado pelo MPFO, (c) Espectro anterior preenchido via distribuição beta.



Fonte: Produzido pelo autor.

Após realizar-se o preenchimento espectral em cada quadro de voz (Figura 30 (c)), passa-se a uma das últimas fases do processo de recomposição do sinal de voz: a transformação do domínio frequencial para o domínio temporal. Tal transformação é conseguida através da transformada rápida inversa de Fourier (IFFT) de mesmo comprimento dos quadros (20 ms). Os quadros transformados para o domínio do tempo são concatenados um a um (sem superposição) para formar um único bloco. Em seguida, são convertidos para um arquivo com extensão *wav*, com 16 bits por amostra e frequência de amostragem de 8 kHz, para que possa ser reproduzido. Dessa forma, estará completo o processo de recuperação do sinal.

5.6 Simulações e Classificações da Qualidade de Voz

Para as simulações, foram utilizados diversos arquivos de voz, muitas deles sendo aproveitados do banco de vozes do sistema de RAL proposto nessa dissertação e que será descrito no capítulo 7. Foi interessante usar algumas dessas frases (“amanhã ligo de novo” e o “prazo tá terminando”) para diversos locutores diferentes (com ambos os sexos, variadas idades e etc), por elas apresentarem tanto fonemas nasalados como vocalizados, dando assim uma visão geral da qualidade dos sinais gerados. Os resultados das simulações foram focados na inteligibilidade e qualidade de voz *versus* taxa de bits.

De início, as simulações foram realizadas sem considerar a técnica do preenchimento espectral via distribuição beta na análise. A intenção foi comparar o efeito deste método na qualidade do sinal de voz sintetizado. Por descartar grande parte das amostras espectrais, os sinais gerados apresentaram uma redução na qualidade em comparação aos sinais originais, tendo como característica principal um som metálico típico dos *vocoders*. Entretanto, mesmo contendo essa característica, a maioria dos sinais de voz recuperados foram inteligíveis, inclusive em simulações com sinais de voz feminina que, em geral, apresentam mais componentes de alta frequência (susceptivelmente mais afetadas pelos ruídos).

Em relação as simulações efetuadas com a utilização do preenchimento espectral, observou-se uma melhora no caráter metálico da voz, elevando sua qualidade e inteligibilidade, mas, em contrapartida, notou-se a presença de ruídos mais fortes nos sinais gerados em comparação aos gerados sem a sua utilização. Acredita-se que este fato esteja associado ao não emprego do janelamento com superposição no processo de recuperação do sinal, que por questão de complexidade não foi implementado. Talvez, a suavização proposta no domínio frequencial tenha causado, no domínio temporal, descontinuidades mais acentuadas entre quadros adjacentes, sendo essa, possivelmente, uma das causas do ruído mais acentuado.

A inviabilidade da utilização, nesta primeira versão do sistema, da superposição de janelas, tentando reduzir o ruído causado pelo preenchimento espectral, sugeriu que fossem tentados outros procedimentos de mais fácil implementação para eliminá-lo ou reduzi-lo. Um deles foi gerar um novo sinal, a partir da soma ponderada dos sinais produzidos sem e com a utilização da técnica de preenchimento espectral, tentando balancear os efeitos positivos e negativos de cada abordagem. Os sinais gerados sem preenchimento apresentaram menos ruído, mas, em compensação, soaram mais metálicos do que os sinais gerados com o preenchimento. Essa composição ganhou em naturalidade, mas em compensação introduziu ainda ruído. Nos testes de classificação de voz, realizados com algumas pessoas, foi observado que umas optaram pelos arquivos gerados com essa composição e outras optaram pelos sem a composição, levando-se em consideração o preenchimento espectral. Com a finalidade de suavizar as transições entre quadros adjacentes, foi proposto um janelamento adicional de *Hamming* aos quadros já novamente transformados para o domínio do tempo. Esse procedimento reduziu consideravelmente o ruído, melhorando-no significativamente e tornando os sinais bem mais

agradáveis de se ouvir.

Os testes foram realizados com um grupo de indivíduos leigos, ou seja, não treinado para o teste. Esse conjunto é mais adequado, pois se aproxima mais da média da população. A qualidade de voz foi estimada a partir do teste subjetivo ACR. Durante esse teste, os ouvintes foram convidados a classificar a qualidade de voz dos arquivos de saída levando em consideração uma escala absoluta de 1 a 5 (muito ruim – excelente). A partir da média das respostas dadas por eles, gerou-se um valor médio MOS para o *vocoder*. A principal dificuldade durante esses testes foi que as pessoas leigas não estavam familiarizadas com *vocoders* de baixa taxa de bits e ficaram confusas entre um som disarmônico, abafado, com zumbido, com qualidade nasal e com ruído adicionado após a codificação. Mesmo assim, os testes foram conduzidos. Foram testados quatro tipos de sinais gerados:

1. Sinais produzidos sem o uso da técnica de preenchimento espectral;
2. Sinais produzidos com o uso da técnica de preenchimento espectral;
3. Sinais formados pela composição dos dois últimos;
4. Sinais do item 2, usando um janelamento de *Hamming* extra.

A Tabela 8 exibe os resultados obtidos nesses testes.

Tabela 8 – Resultado do teste ACR para as 4 variações do *vocoder* proposto.

Codificador	MOS
Proposto (1)	3,0
Proposto (2)	2,5
Proposto (3)	2,8
Proposto (4)	3,0

Fonte: Produzido pelo autor.

A Tabela 9 exibe uma comparação das taxa de bits e pontuação MOS entre as variações do *vocoder* proposto e os codificadores tradicionais citados no trabalho. Comparando-se com as pontuações MOS obtidas pelos codificadores tradicionais, os resultados obtidos a partir das variações do *vocoder* proposto foram válidos, dada a reduzida taxa de bits (2.7 kbits/s) e a baixa complexidade empregada em sua implementação. Logicamente que essa comparação é desleal com esses codificadores, já que os valores MOS obtidos para eles foram muito mais criteriosos e realizados com um vasto número de ouvintes, ou até mesmo utilizando métodos objetivos como o PESQ.

Tabela 9 – Taxa de bits e pontuação MOS dos vocoders propostos e dos codificadores apresentados.

Codificador	Taxa (kbits/s)	MOS
Proposto (1)	2,7	3,0
Proposto (2)	2,7	2,5
Proposto (3)	2,7	2,8
Proposto (4)	2,7	3,0
G.711	64	4,4
G.722	64	4,5
G.726	40, 32, 24 e 16	4,3
G.728	16	4,2
G.729	8	3,9

Fonte: Produzido pelo autor.

Pode-se observar, também, pela Tabela 9 que o ruído ainda é um fator que desagrada muito numa avaliação desse tipo, refletindo num menor valor MOS para os sinais mais ruidosos, aqueles produzidos através da técnica do preenchimento espectral.

6 RECONHECIMENTO AUTOMÁTICO DE LOCUTOR

6.1 Introdução

Os seres humanos são capazes de distinguir pessoas meramente ouvindo-as falar. Diferenças, ainda que sutis, de timbre, sotaque ou entonação, possibilitam a distinção de uma pessoa de outra apenas pela sua voz. Geralmente, curtos trechos de fala (2 a 3 segundos) são suficientes para o reconhecimento de uma voz familiar.

A área de PDSV que torna possível o reconhecimento de pessoas pela voz por meio de máquinas é chamada de Reconhecimento Automático de Locutor (RAL), termo genérico que se refere à tarefa de discriminar pessoas baseando-se apenas nas características vocais (PARANAGUÁ, 1997). A identidade de uma pessoa através de sistemas de RAL tem o intuito de incrementar a confiabilidade em diversas aplicações de segurança (PETRY, 2002), dentre as quais:

- **Aplicações bancárias:** Por meio da voz, pode-se desejar que uma senha seja associada a um cliente para verificação de sua identidade (e.g., consulta de saldos por telefone). Tal procedimento poderia também ser estendido aos caixas 24 horas. A confiabilidade de tais sistemas deve ser alta;
- **Controle de acesso a áreas restritas:** A tecnologia de RAL pode ser útil para restringir o número de pessoas a locais específicos como instalações militares, laboratórios, presídios etc., possibilitando o acesso de forma segura, prática e confiável;
- **Controle de acesso em *softwares*:** Informações confidencias podem estar guardadas em alguns locais específicos em computadores. Pode ser requerido que essas informações sejam apenas acessadas por algumas pessoas. Usando-se o RAL, pode ser feita uma autenticação da identidade das pessoas, permitindo ou não o acesso delas;
- **Aeroportos:** Com a finalidade de se evitar fraudes em aeroportos, cartões de embarque podiam conter amostras de voz dos passageiros.
- **Ponto eletrônico:** Um artifício usado em empresas para registrar a presença de um funcionário, bem como para controlar a sua entrada e saída, é o chamado ponto eletrônico. Normalmente, esse controle é feito por cartões magnéticos, que não contem nenhuma informação mais apurada do usuário, permitindo que outras pessoas tenham a possibilidade de fraudar esse controle. A introdução de sistemas de RAL pode aumentar a confiabilidade e a segurança do processo;

- **Assinatura Eletrônica:** Comercializações à distância poderiam ser autenticadas através de um assinatura eletrônica por voz. Por intermédio da confirmação da identidade vocal do usuário, realizar-se-ia uma transação mais segura, inclusive por meio cartões de crédito;
- **Hotéis:** Em hotéis, o acesso aos quartos é feito por uma chave (às vezes magnética) que é recebida no ato do cadastro. Essa chave deve ser guardada ou deixada na recepção toda vez que o hóspede tem que deixar as dependências do hotel. Tem-se, portanto, o risco da perda dessa chave e a possibilidade de que pessoas não autorizadas tenham acesso ao quarto. A substituição por senhas vocais pode ser um meio mais eficiente desse controle.

Os sistemas que trabalham com RAL calculam, por algum critério específico, a similaridade entre as características da voz do locutor que se deseja reconhecer, com as características de voz de um conjunto de locutores previamente armazenadas pelo sistema de reconhecimento. As técnicas utilizadas variam de acordo com o tipo de problema que se deseja solucionar. Os tipos de RAL, cada um para uma finalidade específica, são descritos em seguida.

6.2 Tipos de RAL

Os sistemas de RAL dividem-se em:

i. Verificação Automática de Locutor (VAL);

ii. Identificação Automática de Locutor (IAL).

6.2.1 VAL

Nos sistemas de VAL, faz-se uso da máquina para verificar a identidade da voz de uma pessoa que a reivindica (CAMPBELL, 1997). É fornecido ao sistema uma amostra de voz e uma identificação correspondente ao suposto falante, cabendo ao sistema avaliar se tal amostra é (ou não) suficientemente similar aos padrões de referência desse locutor específico.

Independentemente do número de locutores cadastrados, é feita apenas uma comparação (aceitar ou não aceitar) específica ao locutor pretenso. A inclusão de mais locutores na população teste não indica que o tempo de processamento necessário para as comparações se alterará significativamente em comparação a uma possível não inclusão de locutores. Por isso, a probabilidade de ocorrência de erros de verificação mantém-se a mesma para cada locutor, mesmo depois da inclusão de um ou mais locutores (PETRY, 2002).

Na VAL, pode haver erros de dois tipos: (i) a falsa aceitação (FA) de um locutor impostor ou mímico, ou (ii) a falsa rejeição (FR) de um locutor verídico (ATAL, 1976) (ROSEMBERG, 1984). Na literatura, há outras denominações para a VAL, incluindo-se: verificação de voz, autenticação de locutor e autenticação de voz (CAMPBELL, 1997).

6.2.2 IAL

Ao contrário da VAL, na IAL não há a reivindicação de autenticidade. O sistema é que deverá decidir, dentre um determinado número de locutores, qual o usuário autêntico ou se o mesmo é desconhecido dentre os possíveis locutores cadastrados (CAMPBELL, 1997). A partir de uma amostra de voz submetida ao sistema, o mesmo dever ser capaz de compará-la aos padrões de referência dos locutores registrados, identificando entre eles o mais semelhante. Esta identificação pode ser implementada com rejeição ou sem rejeição. No primeiro caso, é estabelecido um limiar para cada usuário. Para o locutor ser considerado autêntico, a similaridade entre as características de sua elocução teste e as características extraídas de seu padrão deverá superar esse limiar. Em caso negativo, o locutor é considerado um impostor.

Na IAL sem rejeição o sistema sempre escolherá um dos locutores cadastrados a partir da maior similaridades entre as características extraídas e a elocução teste. O grau de dificuldade associado a IAL com rejeição é maior, visto que há a possibilidade da elocução teste não pertencer a nenhum dos locutores conhecidos.

Como são feitas comparações com todos os locutores cadastrados, a inclusão de mais locutores à população teste acarreta numa maior carga de processamento em relação aos sistemas de VAL. Pelo mesmo motivo, a probabilidade de ocorrência de erros cresce com o incremento do número de locutores a serem testados (PETRY, 2002).

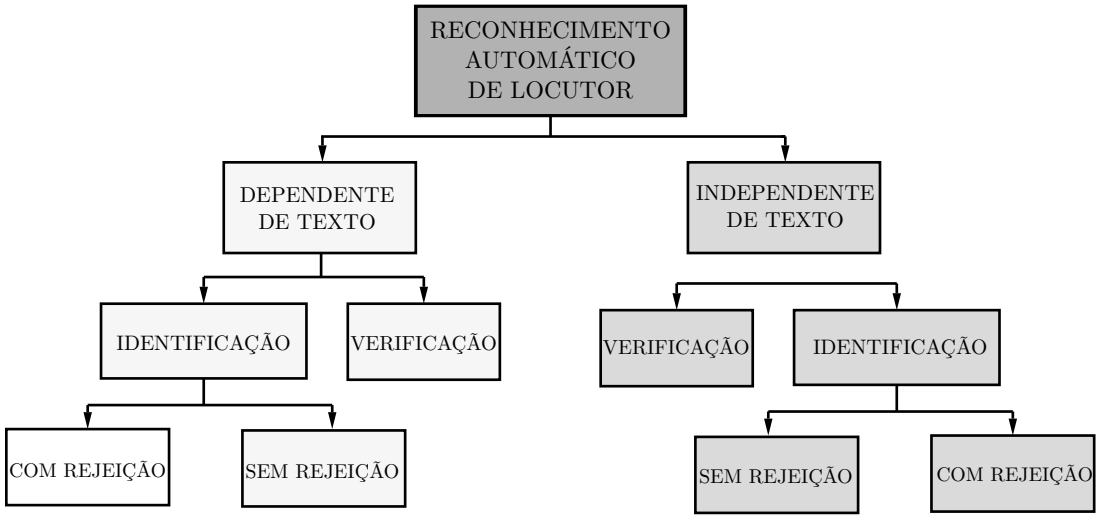
Os sistemas de RAL utilizam-se de frases ou textos para a comparação entre as elocuções testes e os padrões extraídos das elocuções de treinamento. Tais frases ou textos podem ser padronizados ou não. Pode-se submeter, aos algoritmos de RAL, dois tipos de testes distintos:

Teste Dependente de Texto: Nesse tipo de abordagem, são reservadas frases padrão que apresentem grande quantidade de fonemas nasalados e vocalizados. Essas frases são pronunciadas por todos os locutores e são utilizadas para treinamento do sistema e geração dos padrões de referência;

Teste Independente de Texto: Nesses sistemas, as comparações entre as elocuções teste e os padrões gerados são feitas a partir de qualquer amostra de voz pronunciada, pelo locutor em teste, no momento da gravação. O usuário estará livre para pronunciar qualquer frase ou texto que eventualmente queira. Toda essa flexibilidade dos sistemas de RAL independente de texto, acarreta numa maior dificuldade de distinção correta pelos algoritmos de reconhecimento.

A Figura 31 ilustra os tipos de RAL.

Figura 31 – Ilustração dos tipos de RAL



Fonte: Adaptado de Campbell (1997).

Por ser de mais fácil implementação, este trabalho é focado, exclusivamente, na IAL sem rejeição.

6.3 Estrutura Básica dos Sistemas de RAL

Um sistema de RAL é basicamente constituído pelas etapas de **treinamento** e **reconhecimento**. A etapa de treinamento deve ser realizada antes do reconhecimento propriamente dito. Os passos do treinamento são:

i. obtenção dos sinais de voz;

ii. extração de suas características mais importantes;

iii. geração de padrão a partir das amostras do sinal;

iv. obtenção de limiares associados aos padrões gerados (exclusivo para o caso da VAL).

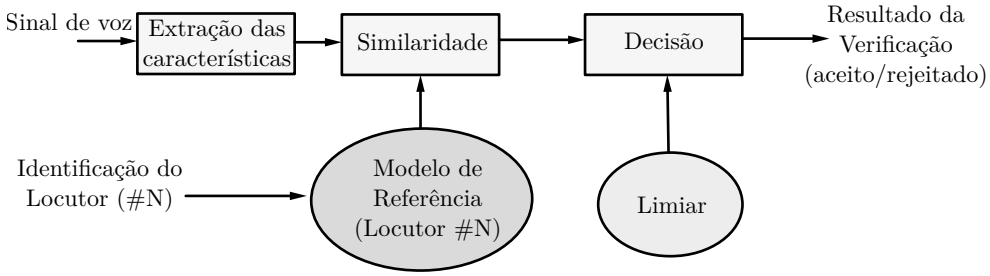
Já os passos do reconhecimento são os seguintes:

i. extração das características mais importantes das elocuções testes;

ii. comparação com os padrões obtidos pela etapa do treinamento, a partir das amostras do sinal.

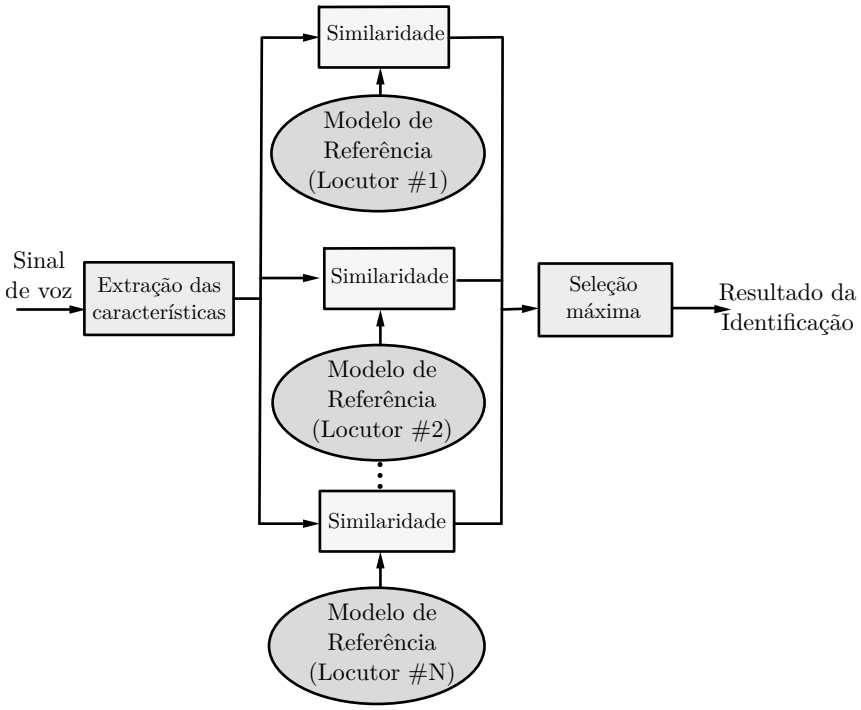
Nos sistemas de VAL, a medida de distorção, obtida na etapa do reconhecimento, será realizada apenas com o padrão do locutor pretenso. Nesse caso, o limiar indicará se a identidade foi aceita ou não. Nos sistemas de IAL, o padrão que obtiver uma menor distorção em relação as características da elocução teste será o selecionado e o locutor detentor desse padrão, o escolhido. As Figuras 32 e 33 ilustram as estruturas básicas dos sistemas de VAL e IAL, respectivamente.

Figura 32 – Ilustração da estrutura básica do sistema de VAL.



Fonte: Adaptado de Campbell (1997).

Figura 33 – Ilustração da estrutura básica do sistema de IAL.



Fonte: Adaptado de Campbell (1997).

6.3.1 Extração das Características

A extração das características consiste, basicamente, em uma compressão de dados com o objetivo de diminuir a quantidade de amostras do sinal original. Normalmente essas amostram apresentam grande quantidade de informação redundante ou que não se relacionam diretamente com o locutor. Isto permite a economia de tempo de processamento por parte dos algoritmos de reconhecimento, tornando mais simples a distinção entre amostras de locutores diferentes (PETRY, 2002). Com uma base de dados mais reduzida, porém apresentando parâmetros mais consistentes, viabiliza-se uma classificação mais confiável e eficiente. Dessa forma, os parâmetros precisam conter apenas informações relevantes da voz. Para um apropriado reconhecimento, algumas características desejáveis desses parâmetros devem ser satisfeitas, entre as quais (WOLF, 1972):

1. Eficiência na representação de informação do locutor que a produziu;
2. Facilidades de se mensurar, reduzindo probabilidade de erros e complexidade;
3. Estáveis no tempo, mantendo sempre uma coerência de valores, independente das situações em que forem adquiridas;
4. Ocorrência natural e frequente, sendo representativos em qualquer amostra de voz;
5. Pouca variação com o ambiente, devendo ser pouco afetados pelas condições em que as amostras foram adquiridas;
6. Não susceptível à mímica, sendo imunes a esse tipo de procedimento.

Hoje em dia, muitas dessas exigências já foram satisfeitas pelas técnicas de reconhecimento, apesar de outros fatores ainda limitarem seu desempenho. A variabilidade das elocuções é o fator de mais preponderância na limitação desse reconhecimento. Ela apresenta-se da seguintes forma (RABINER, 1994):

1. Variabilidade dos sons de um mesmo locutor ou de diferentes locutores;
2. Variabilidade do canal de gravação (e.g., microfone utilizado para as gravações);
3. Variabilidade devido à introdução de ruído pelo ambiente;
4. Variabilidade na produção da fala (e.g, hesitação, ruído de respiração e estalos labiais).

Esses fatores, normalmente, não podem ser eliminados, ficando a cargo da tecnologia de reconhecimento realizar artifícios para reduzir os seus efeitos, dentre os quais (RABINER, 1994):

1. Detecção da voz propriamente dita, suprimindo o ruído de fundo ou o silêncio presentes em instantes de tempo anteriores e posteriores à voz ativa;
2. Reconhecer a sentença falada baseada em reconhecimento de padrões determinísticos ou por métodos fonético-acústicos (PARANAGUÁ, 1997).

6.3.2 Parâmetros Extraídos do Sinal de Voz

A seguir são apresentados os principais parâmetros utilizados por métodos de RAL para representar sinais de voz por meio de algumas de suas características mais relevantes.

6.3.2.1 Banco de Filtros

Em geral, essa técnica consiste em aplicar filtros passa-bandas deslocados em frequência, extraindo como parâmetro a energia na saída de cada um deles (RABINER; JUANG, 1993) (PETRY, 2002).

6.3.2.2 Energia de Tempo Curto

Uma das formas mais simples de ser representar um sinal de voz é pela sua energia de tempo curto. Para um sinal $x[n]$, sua energia de tempo curto é dada por

$$E_n = \frac{1}{N}\sum_{m=0}^{N-1}\left(w[m]x\left[n-\frac{N}{2}+m\right]\right)^2, \tag{1}$$

em que $w[m]$ é uma janela de N pontos, aplicada ao sinal pré-processado, $x[n]$, e n é o índice de amostragem (tempo discreto) do centro da janela.

6.3.2.3 Taxa de Cruzamento pelo Zero

A taxa de cruzamento pelo zero (ZCR), pode ser definida como a média ponderada do número de vezes que o sinal altera sua amplitude de negativa para positiva ou vice-versa (KONDOZ, 2004), ou seja:

$$\text{ZCR} = \frac{1}{N}\sum_{n=0}^{N-1} 0.5|sign(y[n]) - sign(y[n-1])|, \tag{2}$$

em que N é o número de amostras do quadro de voz avaliado, $y[n]$ é o sinal de voz pré-processado e $sign(y[n])$ é a função sinal definida na seção 4.1.2 do Capítulo 4.

6.3.2.4 Coeficientes Cepstrais

Como visto no Capítulo 4, o processo de geração da fala pode ser modelado por um sistema linear variante no tempo e que possui propriedades que variam lentamente. Como a voz apresenta características quase estacionárias para curtos segmentos, pode ser modelada como tendo sido gerada por um sistema linear invariante no tempo (LIT) excitado por um trem de impulsos quase-periódicos ou por uma fonte de ruído aleatório (PARANAGUÁ, 1997). Pode-se definir então, que o sinal de voz, $s[n]$, é o resultado da operação de convolução ($\otimes$) do sinal de excitação $u[n]$ com a resposta ao impulso $h[n]$ do sistema LIT, ou seja:

$$s[n] = u[n] \otimes h[n]. \tag{3}$$

Passando-se a Equação (3) do domínio do tempo para o domínio da frequência, tem-se:

$$S(j\omega) = U(j\omega).H(j\omega), \tag{4}$$

em que a operação de convolução da Equação (3) foi transformada em uma operação de multiplicação pela transformada de Fourier. Aplicando-se a função logarítmica, a Equação (4) transforma-se em uma soma (ou sobreposição) de sinais expressa como

$$\log(S(j\omega)) = \log(U(j\omega)) + \log(H(j\omega)). \tag{5}$$

Do sistema, portanto, obtém-se saídas lineares, ou seja, componentes representativas do sinal tornam-se linearmente combinadas. Aplicando-se a transformada inversa de Fourier ao sistema, obtém-se o *cepstrum* ou os coeficientes cepstrais (FCC, do inglês *"Frequency Cepstral Coefficients"*) do sinal . O processo é ilustrado a seguir.

$$\begin{aligned} \mathcal{F}^{-1}\{\log(S(j\omega))\} &= \mathcal{F}^{-1}\{\log(U(j\omega)) + \log(H(j\omega))\} \\ C_s(n) &= C_u(n) + C_h(n). \end{aligned} \tag{6}$$

Sabe-se que a parcela do sinal de excitação varia mais rapidamente do que a resposta impulsiva do trato vocal, então os dois sinais poderiam ser separados no domínio cepstral.

6.3.2.5 Coeficientes Mel-Cepstrais

Os coeficientes Mel-cepstrais (MFCC, do inglês *"Mel-Frequency Cepstral Coefficients"*) surgiram devido a estudos na área da psico-acústica, que mostraram que a percepção humana das frequências de tons puros ou de sinais de voz não segue uma escala linear (RABINER; JUANG, 1993). A ideia é fazer um mapeamento da escala linear de frequência, medida em Hz, para uma escala denominada escala *mel*, que corresponde à real percepção do sistema auditivo humano. O mapeamento é dado pela seguinte expressão:

$$mel(f) = 2595.\log_{10}\left(1 + \frac{f}{700}\right). \tag{7}$$

Os coeficientes mel-cepstrais são obtidos de maneira similar aos coeficientes cepstrais. A diferença principal está na aplicação de um banco de filtros digitais triangulares (passa-banda), espaçados segundo a escala *mel*, previamente à introdução da função logarítmica (PETRY, 2002).

6.3.2.6 Coeficientes LPC

Os coeficientes da análise LPC são os mesmo obtidos pelo modelo preditivo linear para sinais de voz, apresentados no Capítulo 3.

No sistema de RAL proposto por este trabalho são usadas, como coeficientes representativos dos sinais de voz (vetor característico), as média das amplitudes dos tons de mascaramento predominantes em cada oitava. O procedimento para obtenção desses coeficientes será mostrado no capítulo seguinte.

6.3.3 Modelamento

Após a extração dos parâmetros do sinal de voz, os mesmos devem ser comparados com os padrões gerados e previamente armazenados, com a finalidade de quantificar as similaridades entres as elocuções teste e os locutores registrados no sistema de reconhecimento. Um melhor aproveitamento da informação contida nesses parâmetros é atingido quando uma comparação adequada é realizada. As técnicas mais conhecidas para a comparação dos padrões são as estatísticas e as determinísticas. Nas técnicas estatísticas, as comparações são realizadas através de medidas de verossimilhança ou probabilidade condicional da observação do modelo. Nesta categoria, destacam-se as técnicas baseadas na função densidade de probabilidade e no HMM (TISHBY, 1991). Nas técnicas determinísticas, o padrão é assumido ser uma réplica perfeita e o processo de alinhamento faz-se necessário para calcular a distância. Os principais métodos determinísticos são os baseados em *Dynamic Time Warping (DTW)* (CAMPBELL, 1997), Quantização Vetorial (QV) (SONG et al., 1985), Redes Neurais Artificiais (RNAs) (FARRELL; MAMMONE; ASSALEH, 1994) e Classificadores Polinomiais (ASSALEH; CAMPBELL, 1999).

No sistema de RAL proposto por esse trabalho o padrão gerado de cada locutor será comparado com as características extraídas das elocuções testes, substituindo os classificadores tradicionais (e.g., HMM, DTW, QV), pela técnica simples de *template matching* via distância euclidiana entre os vetores. Nos sistemas de *template matching* os vetores de características das elocuções de treinamento e teste são comparados diretamente, com a suposição de qualquer um deles é um réplica imperfeita do outro. O sistema identifica o locutor pretenso através do cálculo da menor distância para o padrão gerado no treinamento. Apesar de sua simplicidade de comparação, esses sistemas não tem performances tão eficientes quanto as que utilizam o HMM, por exemplo.

6.3.4 Modelos Ocultos de Markov - HMM

6.3.4.1 Introdução

Por manipularem muito bem os aspectos estatísticos e sequências do sinal de voz, os HMM's são largamente utilizados no reconhecimento automático da voz e do locutor (OLIVEIRA, 2001). Esta popularidade deve-se à existência de um algoritmo eficiente e robusto para treinamento e reconhecimento.

No treinamento do modelo, os parâmetros extraídos do sinal, em janelas de intervalo de tempo curto, chamados também de sequência das observações, são modelados por uma sequência de estados (modelo de Markov de primeira ordem) de acordo com as características variante no tempo do sinal de voz. No reconhecimento, a sequência das observações da elocução teste, caso possua alguma medida de verossimilhança acima de um limiar previamente estipulado, é aceita como verdadeira (PARANAGUÁ, 1997).

6.3.4.2 Descrição do Modelo

Um modelo de Markov é um conjunto finito de elementos formando uma máquina de estados, cujas transições entre eles não são governadas por regras determinísticas, mas por probabilidades de transição entre eles (ANDRADE, 1999). Apenas as transições para o mesmo estado e transições esquerda direita (para o caso de VAL dependente do texto), entre estados são permitidas, dada a característica sequencial da voz (OLIVEIRA, 2001). Na abordagem existem dois processos associados, um envolvendo as transições entre os estados (modelando a sequência temporal da voz) e outro envolvendo as observações de saída de cada estado (modelando as características acústicas do sinal de voz). A designação de "oculto" para o modelo dá-se pelo fato de a sequência de estados não ser observada, mas afetar a sequência de estados observados.

Um modelo de HMM pode, então, ser definido como um par de processos estocásticos (X, Y), onde o X representa um modelo de Markov de primeira ordem (não sendo diretamente observável) e Y representa um sequência de variáveis aleatórias no espaço dos parâmetros acústicos (observações) (LIPORACE, 1982).

Um HMM pode ser definido por um conjunto de parâmetros (RABINER; JUANG, 1993):

- Um conjunto N de estados S_j, incluindo um estado inicial S_i e um estado final S_j. A designação $q_t(i)$ indica estar no estado S_i no tempo t;
- O número de símbolos observáveis em um alfabeto, M. Para símbolos discretos, M pode ser inteiro e finito, para símbolos contínuos, M pode ser infinito;
- Uma matriz de transições $A = [a_{ij}]$, onde a_{ij} representa a probabilidade de se efetuar uma transição do estado i para o estado j, ou seja:

$$a_{ij} = P(q_t = j/q_{t-1} = i), \qquad 1 \leq i, j \leq N, \tag{8}$$

q_t indicando o estado atual. A matriz A deve satisfazer

$$a_{ij} \geq 0, \qquad 1 \leq i, j \leq N, \tag{9}$$

e

$$\sum_{j=1}^{N} a_{ij} = 1, \qquad 1 \leq i \leq N. \tag{10}$$

- Uma matriz de probabilidades de saída $B = [b_j(k)]$, onde $b_j(k)$ define a probabilidade de emissão do símbolo k , ao se chegar ao estado j. Para o HMM discreto, tem-se:

$$b_j(k) = P(x = v_k/q_t = j), \qquad 1 \leq j \leq N, \quad 1 \leq k \leq M. \tag{11}$$

v_k representa o k-ésimo símbolo observado no alfabeto. Novamente a seguintes condições estocásticas devem ser satisfeitas:

$$b_j(k) \geq 0, \qquad 1 \leq j \leq N, \quad 1 \leq k \leq M, \tag{12}$$

e

$$\sum_{j=1}^{M} b_j(k) = 1, \qquad 1 \leq j \leq N. \tag{13}$$

Caso a distribuição seja contínua, o conjunto de observações de cada estado é separado em M grupos, possuindo eles um vetor média e uma matriz de covariância associados (PARANAGUÁ, 1997). Através de uma soma das M distribuições gaussianas N, ponderadas por coeficientes c_{jm} (uma mistura de gaussianas), é calculada a densidade de probabilidade em cada estado (RABINER; JUANG, 1993), dada pela expressão

$$b_j(x) = \sum_{m=1}^{M} c_{jm} N(x, \mu_{jm}, U_{jm}), \tag{14}$$

em que, c_{jm} são os coeficientes de ponderação das gaussianas M, μ_{jm} são os vetores média e U_{jm} são as matrizes de covariância. Os coeficientes c_{jm} e a função densidade de probabilidade da mistura devem satisfazer as definições

$$c_{jm} \geq 0, \qquad 1 \leq j \leq N, \quad 1 \leq m \leq M, \tag{15}$$

e

$$\sum_{m=1}^{M} c_{jm} = 1, \qquad 1 \leq j \leq N. \tag{16}$$

- Uma distribuição do estado inicial, $\Pi = \{\Pi_i\}$, em que

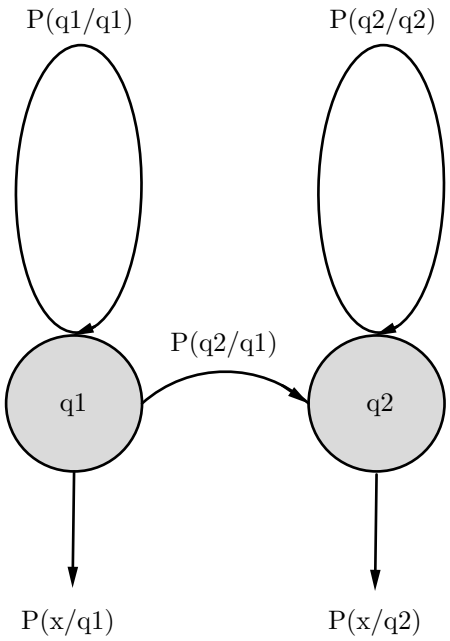
$$\Pi_j = P(q_1 = j), \qquad 1 \leq j \leq N \tag{17}$$

Em face a toda essas definições, o modelo de Markov pode ser representado pela forma compacta

$$\lambda = (A, B, \Pi). \tag{18}$$

Um simples esquema de um HMM é mostrado na Figura 34. De acordo com uma topologia pré-definida, associa-se cada frase, em uma elocução, a um modelo particular de Markov construído a partir de Q estados. Para reconhecimento de voz contínua, os HMM são construídos pela concatenação de unidades elementares da fala, como o fonema (OLIVEIRA, 2001).

Figura 34 – Modelo de um HMM com dois estados (topologia esquerda-direita).



Fonte: Adaptado de Rabiner e Juang (1993).

6.3.4.3 Simplificações da Teoria do HMM

Algumas suposições são realizadas na teoria do HMM, com o objetivo de facilitar o tratamento matemático e computacional. São elas (RABINER; JUANG, 1993):

- **Suposição de Markov:** Assume-se que o modelo é um modelo *markoviano* de primeira ordem, ou seja, o próximo estado depende somente do estado atual.
- **Suposição de Estacionaridade:** Supõe-se que as probabilidades de transição de um estado para o outro não variam com o tempo.
- **Suposição das Observações Independentes:** Assume-se que as observações adjacentes não apresentam nenhum tipo de correlação. Nesse caso, é desconsiderado o efeito de coarticulação.

6.3.4.4 Treinamento

Como visto anteriormente, o modelo de Markov pode ser representado pela forma compacta $\lambda = (A, B, \Pi)$. A função do estágio de treinamento é obter, através do ajuste do modelo λ, um novo modelo $\bar{\lambda}$, a partir de uma sequência de treinamento $\boldsymbol{X} = [x_1, x_2, \ldots, x_N]$, tal que seja máxima a probabilidade $P(\mathbf{X}/\lambda)$. Um procedimento iterativo é necessário para se encontrar o melhor do modelo possível, já que $P(\mathbf{X}/\lambda)$ é uma função não-linear, apresentando muitos máximos locais (OLIVEIRA, 2001). Normalmente os algoritmos que realizam esse procedimento iterativo, para o HMM, são os seguintes (RABINER; JUANG, 1993): algoritmo

Baum-Welch, algoritmo de *Viterbi* e algoritmo *Segmental K-means* para o treinamento, sendo o segundo também usado para o reconhecimento.

Baseando-se no conceito estatístico da esperança do número de transições entre estados e da esperança do número de ocorrências das observações nos estados, o algoritmo de *Baum-Welch* encontra a máxima verossimilhança dos parâmetros do modelo, fazendo uso de duas variáveis auxiliares α e β, e compondo uma terceira variável, $\gamma_t(i)$ (chamada de variável de probabilidade *a posteriori*). Essa variável corresponde à probabilidade de estar no estado i, no instante t, sendo associada à sequência de observações $\boldsymbol{X}$ (RABINER; JUANG, 1993), ou seja:

$$\gamma_t(i) = P(q_t = i/\mathbf{X}, \lambda). \tag{19}$$

A partir das variáveis e dos parâmetros do modelo inicial λ a ser ajustado, os parâmetros do novo modelo $\bar{\lambda}$, discreto, são dados por

$$\bar{\pi}_i = n^o \text{ esperado de vezes do estado } q_i \text{ no instante } t, \tag{20}$$

$$\bar{a}_{ij} = \frac{n^o \text{ esperado de transições do estado } i \text{ para o estado } j}{n^o \text{ esperado de transições do estado } i}, \tag{21}$$

$$\bar{b}_j(x_k) = \frac{n^o \text{ de vezes que } x_k \text{ é observado em } q_j}{n^o \text{ esperado de transições pelo estado } j}. \tag{22}$$

Para o caso do modelo λ ser contínuo, o novo modelo terá os parâmetros c_{jm}, μ_{jm} e U_{jm} ajustados por

$$\bar{c}_{jm} = \frac{n^o \text{ esperado de ocorrência da Gaussiana } m \text{ no estado } j}{n^0 \text{ esperado de ocorrências do estado } q_j}, \tag{23}$$

$$\bar{\mu}_{jm} = \frac{n^o \text{ esperado de ocorrência da Gaussiana } m \text{ em } q_j \text{ ponderada por } o_t}{n^o \text{ esperado de ocorrer } q_j \text{ e na mistura } m}, \tag{24}$$

$$\bar{U}_{jm} = \frac{n^o \text{ esperado de ocorrência de uma Gaussiana } m \text{ em } q_j \text{ ponderada pela matriz de covariância}}{n^o \text{ esperado de estar no estado } q_j \text{ e na mistura } m}. \tag{25}$$

O primeiro passo do algoritmo é substituir λ por $\bar{\lambda}$. O processo iterativo encerra-se quando não há mais melhorias significativas em $P(\mathbf{X}/\lambda)$.

No algoritmo de *Viterbi*, em vez de valores esperados, são usadas as somas das transições ocorridas e observações encontradas ao longo da melhor sequência de estados obtida para as

observações fornecidas. Os parâmetros a_{ij} são obtidos pela relação entre a contagem do número de transições do estado i para o estado j, e o número de transições feitas a partir de q_i. Para cada estado, após se agrupar os vetores de observações em M grupos através do algoritmo *K-means* modificado, obtém-se os parâmetros média, covariância e coeficiente de mistura. A média é estimada de todas as observações pertencentes a um dos M grupos de gaussianas de cada estado. O mesmo é feito para a covariância. O coeficiente de misturas será igual à relação entre o número de observações classificadas no grupo e o número total de observações classificadas naquele estado. Logo, os parâmetros reestimados são dados por

$$\bar{\mu}_{jm} = \frac{1}{N_{jm}} \sum_{i=1}^{N_{jm}} x_i, \tag{26}$$

$$\hat{U_{jm}} = \frac{1}{N_{jm}} \sum_{i=1}^{N_{jm}} (x_i - \mu_{jm})(x_i - \mu_{jm})^T, \tag{27}$$

$$c_{jm} = \frac{N_{jm}}{N_j}, \tag{28}$$

em que x_i é a i-ésima observação associada ao estado j e gaussiana m (que possui N_{jm} observações classificadas), e N_j é o número de observações no estado j e N_{jm}, o número de observações na m-ésima mistura do estado j (OLIVEIRA, 2001).

Como uma tentativa para solucionar os problemas de sensibilidade aos valores iniciais do modelo λ (observado nos dois algoritmos citados anteriormente), surgiu o algoritmo *Segmental K-means*. O primeiro passo desse algoritmo consiste em dividir as observações pelos estados (agrupamento) de maneira sequencial. Em seguida, o algoritmo de *Viterbi* é aplicado para a obtenção do modelo $\bar{\lambda}$. A partir daí, o algoritmo de *Baum-Welch* usa o modelo para reestimar todos os parâmetros. Obtidos os modelos por ambos os algoritmos, é feita uma comparação da verossimilhança entre eles. Caso o valor da verossimilhança exceda um limiar, substituem-se os valores anteriores pelos atuais e repete-se o treinamento. O processo encerra-se quando o modelo converge para um valor abaixo desse limiar, e os parâmetros passam a estar treinados. Mais detalhes sobre os algoritmos citados podem ser encontrados em (RABINER; JUANG, 1993).

6.3.4.5 Reconhecimento

A função do estágio de reconhecimento é decidir se uma dada elocução teste foi originada por um determinado locutor. Para o caso da VAL, calcula-se a verossimilhança da elocução $P(O|\lambda)$ ter sido gerado pelo modelo em questão, aceitando-se ou não, dependendo do limiar calculado para esse modelo.

No caso da IAL, essa verossimilhança é testada com todos os modelos do sistema, sendo aceito o que obtiver a maior. O algoritmo de *Viterbi* é o utilizado para o cálculo da

verossimilhança, fornecendo o caminho de máxima verossimilhança, da elocução teste, em pertencer ao modelo treinado. O valor obtido é então comparado a um limiar (se houver), por algum método de decisão. O método mais utilizado para esse fim é o método de Bayes (PARANAGUÁ, 1997).

7 SISTEMA PROPOSTO DE RAL

7.1 Introdução

Este capítulo é reservado ao sistema de RAL desenvolvido neste trabalho, cuja concepção foi motivada pelo *vocoder* proposto no Capítulo 5. Constatou-se que o MPFO, técnica utilizada na essência do seu projeto, apesar de simplificar, significativamente, a quantidade de amostras espectrais dos sinais de voz, preservava características fundamentais que possibilitavam o reconhecimento de maneira computacionalmente eficiente. Assim, esta nova ideologia de RAL herda parte da teoria proposta pelo *vocoder*, adaptando-na ao propósito da identificação de locutores. O principal objetivo é oferecer um compromisso entre complexidade e taxa de identificações corretas, podendo ser atrativo para aplicações em sistemas embarcados.

Para representar o vetor característico dos sinais de voz, o qual contém as informações relevantes dos locutores, o sistema não segue o padrão dos sistemas de RAL tradicionais. Ao invés disso, utiliza um novo procedimento que calcula a média das amplitudes dos tons de mascaramento por oitava, resultante da simplificação espectral pelo MPFO. Adicionalmente, para comparação do padrão extraído das características de cada locutor com as características extraídas das elocuções teste, substitui os classificadores padrão, baseados nos métodos citados no Capítulo 6, pela técnica simples de *template matching* via distância euclidiana entre os vetores.

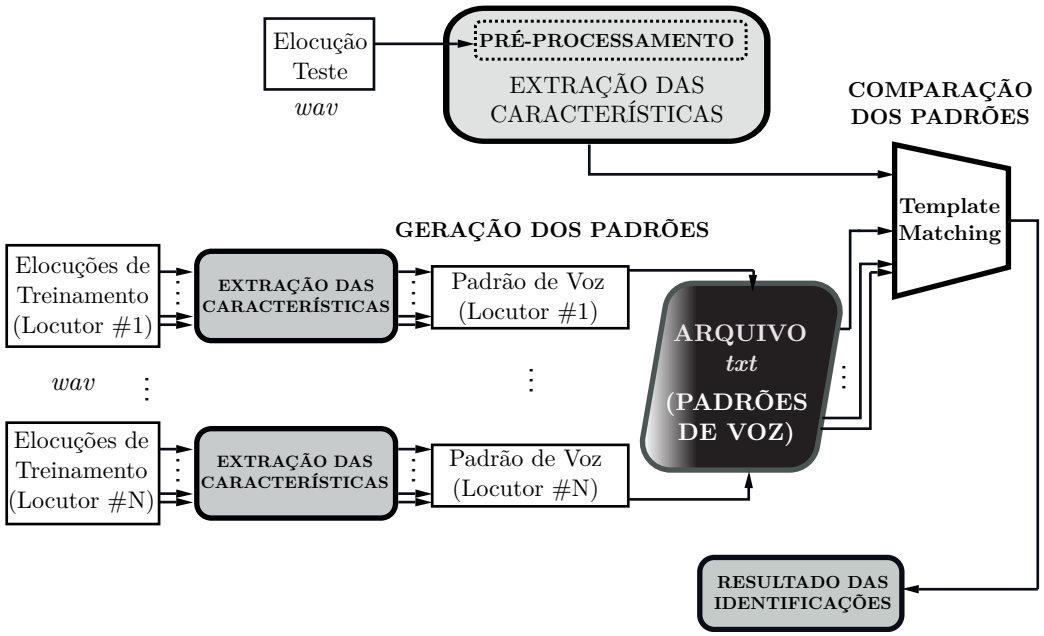
Ainda, são apresentados resultados de dois testes submetidos ao sistema de RAL proposto: com e sem dependência de texto. É investigada, também, a eficiência do sistema na ausência da pré-ênfase do sinal de voz.

7.2 Visão Geral do Sistema

Em geral, os sistemas de RAL são divididos em quatro fases principais: (i) gravações das elocuções; (ii) pré-processamento do sinal de voz; (iii) obtenção dos padrões dos locutores e (iv) comparação dos padrões. Nas gravações das elocuções, evita-se principalmente, que a variabilidade do canal de gravação (eg., microfone) e a introdução de ruído pelo ambiente afetem a qualidade das amostras adquiridas. No pré-processamento, além dos sinais serem segmentados e janelados, como no *vocoder* do Capítulo 6, serão, também, pré-enfatizados e submetidos a um detector de pontos extremos. Na obtenção dos padrões dos locutores, serão utilizados procedimentos desenvolvidos no Departamento de Eletrônica e Sistemas da UFPE. Na comparação dos padrões utiliza-se a técnica simples de menor distância euclidiana entre o vetor característico da elocução teste e os vetores padrão dos locutores. A Figura 35 apresenta

um diagrama de blocos simplificado desse sistema. A descrição detalhada é dada nas próximas seções.

Figura 35 – Diagrama de blocos simplificado do sistema de RAL proposto.



Fonte: Produzido pelo autor.

7.2.1 Implementação do Sistema

Pelas mesmas razões elencadas na implementação do sistema de codificação de voz do Capítulo 5, o sistema de RAL proposto foi, também, desenvolvido através de programas elaborados no MATLAB® e exibidos na Tabela 10.

Tabela 10 – Algoritmos do sistema de RAL proposto.

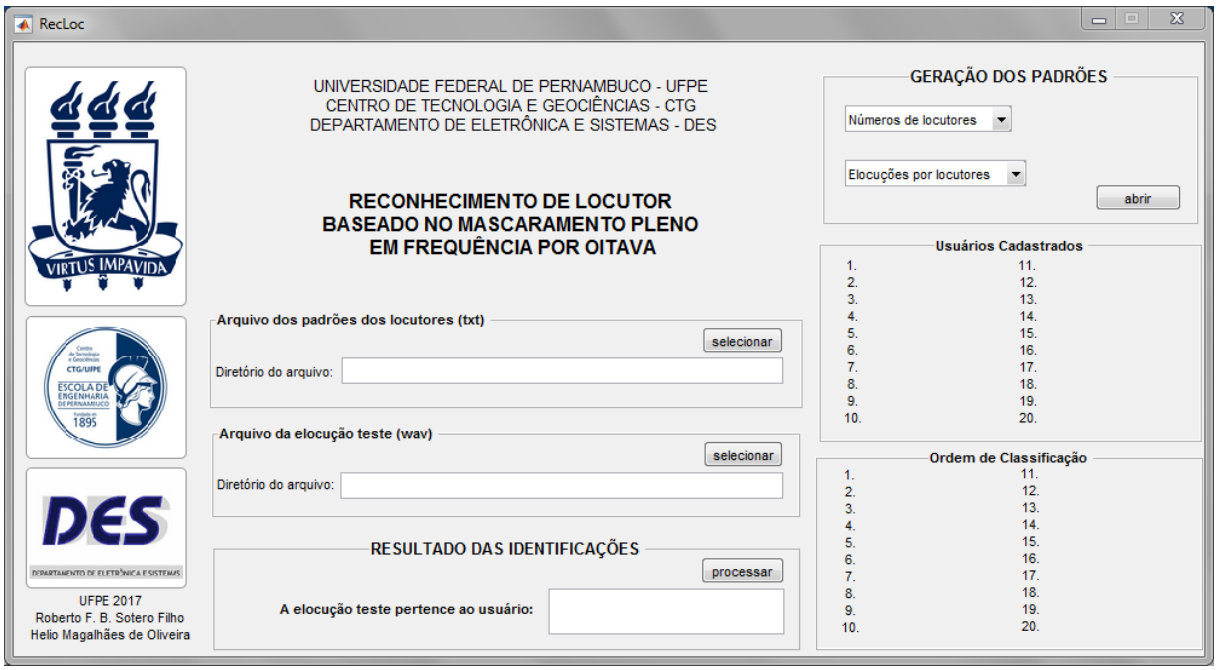
Algoritmo	*Função*
RecLoc	Identificação do locutor pretenso entre os cadastrados
PadraoLoc	Geração de arquivo *txt* com padrões dos locutores
vetorCaract	Obtenção do vetor característico de uma elocução
gerPadrao5eloc	Geração do padrão do locutor para 5 elocuções
gerPadraoEdit	Geração do padrão do locutor (editável pelo usuário)

Fonte: Produzido pelo autor.

O sistema é composto, basicamente, de um software principal, **RecLoc**, cuja interface gráfica (Figura 36), habilitada ao se digitar *RecLoc* na janela de comando do MATLAB®, permite ao usuário o cadastro de 5 até 20 locutores, com 5 elocuções de treinamento para geração dos

padrões. O número de elocuções de treinamento foi fixado em 5 para permitir uma interface gráfica mais amigável ao usuário. Entretanto, para os testes houve a necessidade de se incluir mais elocuções de treinamento, com o objetivo de se obter uma representação mais precisa dos padrões. Em casos como esse, o usuário deve seguir os procedimentos contidos no Anexo C, a fim de realizar alterações no algoritmo **gerPadraoEdit** e no arquivo *exemploPadrao.txt*, disponibilizados como modelo.

Figura 36 – Ilustração da interface gráfica do software de RAL proposto.

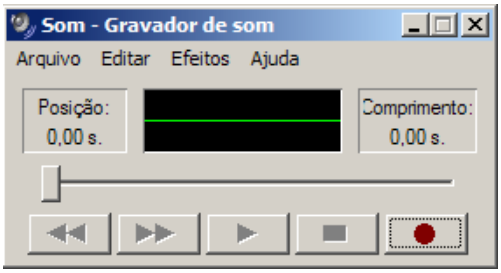


Fonte: Produzido pelo autor.

O cadastro dos locutores é realizado da seguinte forma: primeiramente o usuário seleciona, no painel "GERAÇÃO DOS PADRÕES", o número desejado de locutores e a quantidade de elocuções de treinamento para cada locutor Posteriormente, pressiona o botão "abrir" para que se habilite a interface gráfica de geração dos padrões dos locutores, **PadraoLoc** (Figura 38). Nela, o usuário pode carregar, nos campos correspondentes, todas as 5 elocuções de treinamento para cada locutor. Essas elocuções devem estar no formato de reprodução *wav* e podem ser obtidas, por exemplo, através do gravador de som disponível no Windows® (Figura 37).

O sistema, também, é composto dos algoritmos secundários **vetorCaract** e **gerPadrao5eloc**, que são utilizados pelos algoritmos principais para geração do vetor característico (treinamento e teste) e obtenção do padrão de cada locutor a partir de 5 elocuções de treinamento, respectivamente. Após o carregamento de todas as elocuções de treinamento, o usuário deve pressionar o botão "gerar" para que o sistema processe os dados alimentados e gere um arquivo *txt*, com nome salvo pelo usuário, contendo os padrões de voz de todos os locutores escolhidos. É este arquivo que o usuário deve carregar no **RecLoc** para finalizar o processo

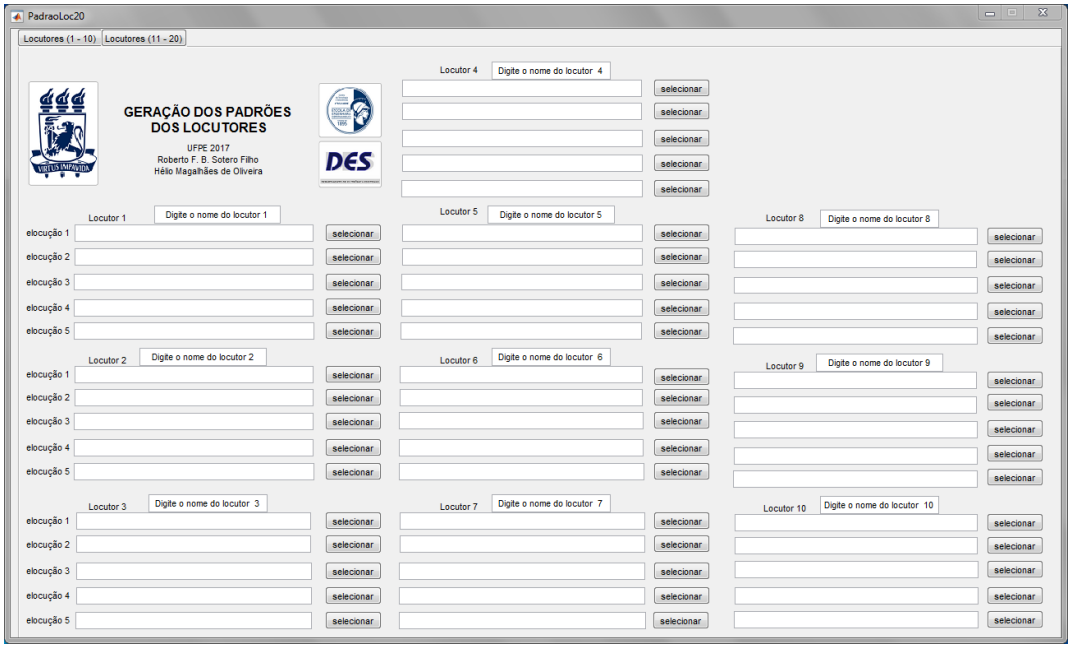
Figura 37 – Imagem do gravador de som do Windows®.



Fonte: Produzido pelo autor.

de armazenamento dos locutores no sistema. Ao se realizar isso, o programa exibirá no painel "Usuários Cadastrados" todos os locutores disponíveis para o reconhecimento, estando pronto para a etapa das identificações. Para se completar o processo de reconhecimento, o usuário seleciona a elocução teste e pressiona o botão "processar". O sistema, então, calcula distâncias entre o vetor característico da elocução teste e todos os padrões dos locutores cadastrados, e exibe na tela os nomes dos locutores ordenados pelos de menores distâncias. O locutor que encabeça a lista é apontado pelo sistema como sendo o detentor da elocução teste, e tem, também, o nome exibido em destaque no painel "RESULTADO DAS IDENTIFICAÇÕES". Essa ordenação em locutores mais prováveis pode ser utilizada para reduzir a base de dados para um número pré-estabelecido de locutores potenciais.

Figura 38 – Ilustração da interface gráfica do gerador de padrões de voz.



Fonte: Produzido pelo autor.

As etapas realizadas para o reconhecimento são dadas abaixo:

i. escolher o número de locutores e elocuções de treinamento;

ii. abrir programa de geração dos padrões (**PadraoLoc**);

iii. carregar todas as elocuções de treinamentos por locutor;

iv. gerar padrões dos locutores (arquivo *txt*);

v. alimentar arquivo de padrões no **RecLoc**;

vi. carregar elocução teste;

vii. processar o sistema.

As próximas seções trazem a descrição detalhada do sistema proposto.

7.3 Aquisição dos Sinais de Voz

O processo de identificação do locutor tem início com a gravação das elocuções para o processamento. Isso é realizado utilizando-se um microfone, cuja saída está conectada a uma placa de som instalada em um computador. Essa tem a função de converter o sinal analógico de voz em amostras igualmente espaçadas no tempo, a uma taxa que pode ser previamente escolhida.

Tipicamente, a energia de um sinal de voz é concentrada numa faixa de frequência de até 4 kHz, ainda que a realização (pronúncia) típica de fonemas fricativos (e.g. /s/) possua substancial parte da energia espectral acima desta frequência. No entanto, como isso ocorre apenas para sons de natureza ruidosa, eles contém pouca informação sobre o locutor (que se concentra mais nos sons vocálicos). Diante disso, o valor adotado para a taxa de amostragem do sistema foi de 8 kHz, utilizando 16 bits de resolução e 1 canal, mono. O software **RecLoc** é o responsável por transformar a taxa de amostragem das elocuções para esta taxa escolhida.

7.4 Pré-Processamento dos Sinais de Voz

Após a aquisição dos dados e sua conversão em amostras digitais, passa-se à fase do pré-processamento. Essa etapa compreende a pré-ênfase, a detecção de pontos extremos (*endpoints*), a segmentação dos dados em quadros (*frames*) e o janelamento.

7.4.1 Pré-Ênfase

Devido a características fisiológicas do sistema de produção da fala, o sinal de voz irradiado pelos lábios apresenta uma atenuação de aproximadamente 6 dB/oitava nas altas frequências. O filtro de pré-ênfase serve para compensar esta atenuação, antes da análise espectral, melhorando a eficiência da análise (RABINER; SCHAFER, 1978); sendo a audição menos sensível a frequências acima de 1 kHz do espectro, a pré-ênfase amplifica esta área do espectro, auxiliando os algoritmos de análise espectral na modelagem dos aspectos perceptualmente importantes do espectro da voz (SILVA, 2006). A resposta em frequência do filtro digital pode ser representada por

$$H(z) = 1 - az^{-1}. \tag{1}$$

Neste caso, a saída da pré-ênfase $y(n)$ está relacionada à entrada $x(n)$ pela equação diferença (PETRY, 2002)

$$y(n) = x(n) - a.x(n-1), \qquad 1 \leq n < M, \tag{2}$$

em que M é o número de amostras do sinal amostrado $x(n)$, $y(n)$ é o sinal pré-enfatizado e a constante *a* (o zero do filtro) é normalmente escolhida entre $0,9$ e 1. No trabalho adotou-se um valor de *a* igual a 0,95 (SILVA, 2006).

7.4.2 Detecção de Pontos Extremos (*Endpoints)*

Para se ter um bom desempenho no reconhecimento do locutor é de extrema importância que sejam determinados, de forma eficiente e precisa, o início e o final de uma locução, com a finalidade de excluir os silêncios, que não trazem nenhuma informação adicional sobre a locução a ser reconhecida. Este procedimento reduz o tempo de processamento e evita que o ruído de fundo, que ocorre antes e depois do sinal de voz, prejudique o reconhecimento (RABINER; JUANG, 1993).

Os pontos extremos são determinados pelo primeiro quadro onde o sinal de voz realmente se inicia e pelo último quadro do sinal de voz. Eles são importantes, pois evitam o processamento dos segmentos onde não há voz ativa, evitando carga computacional e economizando tempo, além de servir como marco de início e fim de um segmento de voz (ROSEMBERG, 1984). A determinação dos pontos extremos deve ser feita de forma cuidadosa, pois os mínimos erros nesta estimação podem degradar o reconhecimento. Ela é realizada através de um classificador de voz que pode diferenciar entre sons vocais, não-vocais ou silêncio. Neste trabalho, utiliza-se um classificador baseado nas características temporais do sinal e que foi proposto por Rabiner e Sambur em 1975, o VAD. Seu código fonte, em linguagem do MATLAB®, está disponível

no Anexo B. Ele foi desenvolvido através de um dos algoritmos mais empregados para essa finalidade e que utiliza duas medidas do sinal de voz: a energia do sinal e a taxa de cruzamento do zero, obtidas em janelas de 10 ms de duração. Nele, um intervalo de 100 ms no início da elocução (10 janelas) é utilizado para efetuar uma estatística do ruído de fundo (PELTON, 1993).

7.4.3 Segmentação dos Dados em Quadros e Janelamento

Após a detecção dos pontos extremos, o sinal de voz deve ser particionado em pequenos segmentos bem definidos (*frames*), com o propósito de se obter trechos de voz razoavelmente assumidos como estacionários. Sabe-se que as características dos sinais de voz mudam muito lentamente na voz contínua, e portanto muitas partes da onda acústica podem ser assumidas como estacionárias num intervalo de curtíssima duração (entre 10 e 40 ms). Este intervalo caracteriza o tamanho da janela a ser usada (BEZERRA, 1994). Neste trabalho, o tamanho da janela adotada (sem superposição) é de 20 ms, um valor típico de muitas aplicações envolvendo voz.

O janelamento do sinal tem o objetivo de amortecer o efeito do "fenômeno Gibbs" (RABINER; SCHAFER, 1978) (OPPENHEIM; SCHAFER; BUCK, 1999), que surge devido à descontinuidade das janelas (BEZERRA, 1994). Para o contexto da produção da voz, as características apresentadas, referentes ao janelamento de *Hamming*, mostram que este tipo de janela é mais eficiente quando comparada às janelas Retangular e de *Hanning*, como uma aproximação para a janela ideal (OPPENHEIM; SCHAFER; BUCK, 1999) . Por este motivo, essa é a janela utilizada neste trabalho.

7.5 Geração do Padrão do Locutor

Para obter o padrão de cada locutor, primeiramente é necessário extrair as características das elocuções teste. O procedimento é descrito a seguir.

7.5.1 Extração das Características dos Quadros de Voz

O processo inicial de extração das características é basicamente igual ao processo de simplificação do espectro via MPFO, descrito na subseção 5.4.2 do Capítulo 5. Entretanto, algumas sutis diferenças são observadas. A primeira consiste em adicionar pré-ênfase e detecção de pontos extremos no estágio de pré-processamento do sinal. A segunda está na inclusão das 3 primeiras oitavas, descartadas na análise do *vocoder*. A terceira reside no fato de que não há necessidade de se considerar as posições em que as amostras espectrais de mascaramento ocorreram, já que, nesse caso, não é necessário reconstruir o espectro simplificado. Como resultado da inclusão dessas diferenças citadas anteriormente, cada quadro de voz, agora, é representado, no domínio frequencial, por 7 amostras de mascaramento auditivo, uma para cada oitava. O total de 79 frequências oriundas da estimativa da DFT de comprimento 160 (mostrado na Tabela 11)

é reduzido para 7 amostras sobreviventes (retendo menos do que 5% das componentes espectrais).

Tabela 11 – Número de frequências estimadas pela DFT de comprimento 160 em cada oitava do espectro vocal.

Oitavas (Hz)	**# amostras espectrais/oitava**
32 - 64	1
64 - 128	1
128 - 256	3
256 - 512	5
512 - 1024	10
1024 - 2048	20
2048 - 4096	39
total	79

Fonte: Produzido pelo autor.

Definindo-se o vetor inicial de amostras espectrais, no i-ésimo quadro de voz, por $oct_j^{(i)}$ em que j representa o índice da oitava, tem-se:

$$oct_j^{(i)} = [a_{j,1}^{(i)}\ a_{j,2}^{(i)}\ a_{j,3}^{(i)}\ \ldots\ a_{j,N_j}^{(i)}], \qquad i = 1, \ldots, n; \quad j = 1, \ldots, 7, \tag{3}$$

em que $a_{j,k}^{(i)}$ é a amplitude do k-ésimo ponto da FFT, na janela i e oitava j e N_j é o número de amostras da j-ésima oitava. Aplicando-se o procedimento de busca da amostra espectral de maior magnitude, obtém-se um novo vetor $newoct_j^{(i)}$ sintetizado, contendo N_{j-1} zeros, e a única componente da amostra de mascaramento espectral correspondente ao $max(a_{j,k}^{(i)})$:

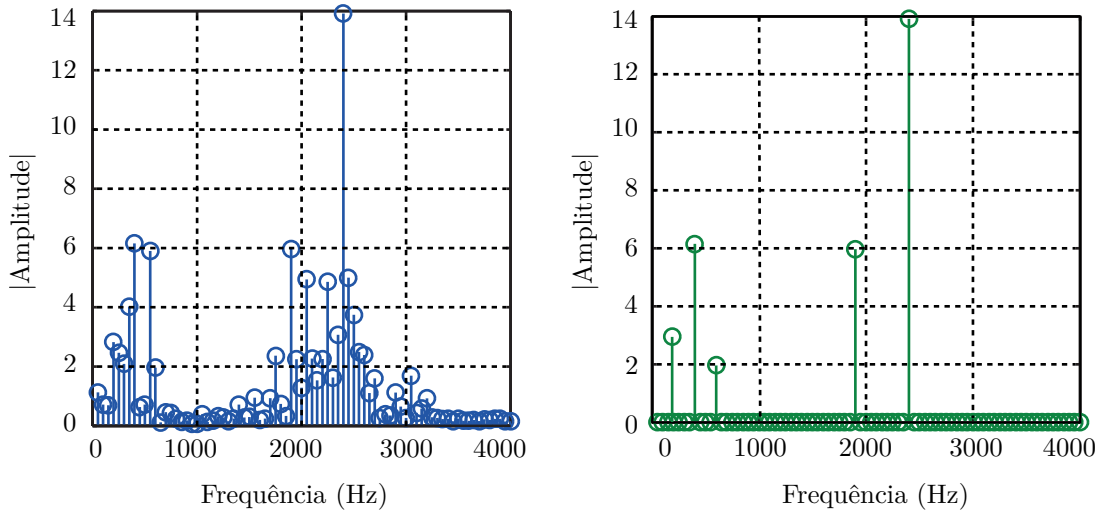
$$newoct_j^{(i)} = [0\ 0\ \ldots\ max(a_{j,k}^{(i)})\ \ldots\ 0], \tag{4}$$

com $k = 1, 2, \ldots, N_j$.

A Figura 39 exibe o módulo do espectro de um quadro, de 20 ms, de uma locução usada para teste, antes e depois da simplificação por tons de mascaramento psico-acústico.

Gerados todos os vetores $newoct_j^{(i)}$, o algoritmo cria, para cada oitava, uma matriz M_j cujas linhas são formadas por todos os n vetores $newoct_j^{(i)}$ do arquivo. Esse procedimento será

Figura 39 – Representação do espectro de frequência de um quadro de voz, antes e depois do processo de mascaramento auditivo.



Fonte: Produzido pelo autor.

útil para calcular as médias dos "tons" de mascaramento.

$$M_j = m_{j,k} = \begin{pmatrix} newoct_j^1 \\ newoct_j^2 \\ \vdots \\ newoct_j^n \end{pmatrix} = \begin{pmatrix} 0 & max(a_{j,k}^{(1)}) & \dots & 0 \\ 0 & 0 & \dots & max(a_{j,k}^{(2)}) \\ \vdots & \vdots & \vdots & \vdots \\ max(a_{j,k}^{(n)}) & 0 & \dots & 0 \end{pmatrix}. \quad (5)$$

Calculando-se a média de cada coluna da matriz M_j, obtém-se a participação média, $\overline{m_j}$, de cada amostra espectral de mascaramento (múltiplos de 50 Hz) no sinal de voz.

$$\overline{m_j} = [m_{j,1}\ m_{j,2} \dots m_{j,N_j}]. \quad (6)$$

em que $m_{j,k} = \frac{1}{n}\sum_{i=1}^{n} max(a_{j,k}^{(i)})$ e k representa o índice no qual existam amostras espectrais de mascaramento. Em seguida, todas as componentes do vetor m_j são somadas. Essa soma representará a participação média dos "tons" de mascaramento dentro de sua respectiva oitava.

$$s_j = \sum_{k=1}^{N_j} m_{j,k}. \quad (7)$$

Esses s_j assim definidos formarão o vetor

$$s_{total} = [s_1\ s_2\ \dots\ s_7]. \quad (8)$$

Os parâmetros obtidos pela etapa anterior são diretamente proporcionais aos níveis de energia dos sinais coletados, fator que pode deturpar a classificação incorretamente. Para realizar a normalização dessas amplitudes, o algoritmo **vetorCaract** faz a divisão do vetor s_{total} pela soma de todas as suas componentes, obtendo-se, enfim, o vetor característica do sinal de voz, com apenas 7 componentes, representantes do número de oitavas, e que será usado para a comparação com as elocuções testes no programa **RecLoc**:

$$s_{norm} = \frac{1}{\sum_{j=1}^{7} s_j}[s_1 \ s_2 \ \ldots s_7]. \tag{9}$$

7.5.2 Obtenção do Padrão dos Locutores

Ao contrário, por exemplo, dos HMM's que utilizam processos iterativos através de três algoritmos (*Baum-Welch*, *Viterbi* e *Segmental K-means*) para a geração dos padrões dos locutores, demandando uma carga computacional substancialmente alta, na presente proposta é utilizada a média de todos os vetores representantes das características do sinal de voz (os s_{norm}), das elocuções reservadas para o treinamento, como padrão p do locutor.

$$p = \frac{1}{n}\sum_{i=1}^{n} st_{norm}(i), \tag{10}$$

em que p é o vetor padrão e st_{norm} são os vetores $s_{norm}(i)$ de treinamento, com a soma sendo vetorial, ou seja, posição por posição. O algoritmo responsável pela obtenção desse padrão é o **gerPadrao5eloc**[1].

Pretende-se, nessa versão inicial do sistema, diminuir a complexidade computacional, utilizando abordagens simples. Fica a cargo de trabalhos futuros o aperfeiçoamento do método, através do uso de metodologias mais refinadas.

7.6 Comparação dos Padrões de Voz

Como última etapa do processo de identificação, tem-se a comparação entre dois vetores. A comparação é realizada através do cálculo da distorção entre eles. Há várias medidas de distorção entre vetores que podem ser utilizadas em reconhecimento de locutor. A medida de distorção mínima ou euclidiana, a medida mais conhecida, foi escolhida. Seja $\boldsymbol{p} = [p_1, p_2, \ldots, p_7]$ o vetor padrão e $\boldsymbol{x} = [x_1, x_2, \ldots, x_7]$ o vetor característico da elocução em teste, então a distância

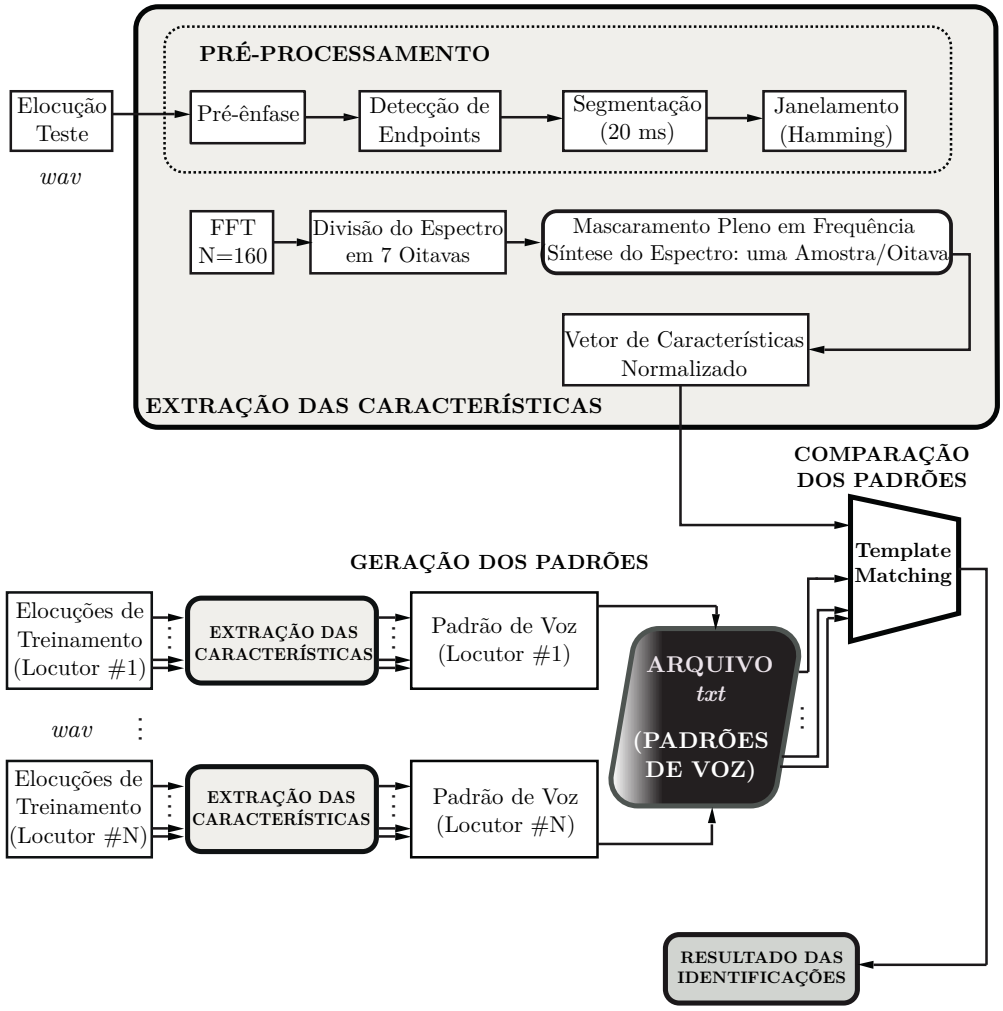
[1] pode também ser utilizado o algoritmo **gerPadraoEdit**, conforme procedimentos disponíveis no Anexo C

euclidiana entre eles, $d(\mathbf{x},\mathbf{p})$ será dada por

$$d(\boldsymbol{x},\boldsymbol{p}) = \sqrt{(x_1 - p_1)^2 + \cdots + (x_7 - p_7)^2} = \sqrt{\sum_{i=1}^{7}(x_i - p_i)^2}. \tag{11}$$

O padrão que obtiver o menor valor para $d(\boldsymbol{x},\boldsymbol{p})$ será o selecionado, e o locutor que for o detentor desse padrão será o escolhido. Simulações de desempenho pela alteração das métricas de comparação dos locutores precisam ser conduzidas, a fim de selecionar a mais adequada, i.e., aquela de melhor compromisso complexidade versus taxa de reconhecimento. Em vista disso, a seleção do locutor foi então realizada com base na técnica simples de *template matching* via distância euclidiana entre o vetor característico de uma elocução teste e os vetores "padrão de voz" armazenados para os locutores cadastrados. A Figura 40 ilustra o diagrama de blocos do sistema de RAL proposto neste trabalho.

Figura 40 – Diagrama de blocos do sistema de identificação proposto.



Fonte: Produzido pelo autor.

7.7 Testes e Resultados Obtidos

Para a obtenção dos resultados e taxa de precisão do método proposto, foram realizados dois tipos de testes padrão. No primeiro deles, a identificação dos locutores é realizada empregando-se uma mesma frase referência para todos os locutores (reconhecimento dependente de texto). No segundo caso, a identificação é realizada com textos escolhidos aleatoriamente no momento da gravação (reconhecimento independente de texto).

As elocuções empregadas para os testes foram todas adquiridas de gravações realizadas numa sala sem qualquer preparação especial destinada à redução de ecos ou mesmo à eliminação total de ruído de fundo, e a partir de um mesmo microfone.

Nos experimentos realizados, foi investigada, também, a eficiência do sistema na ausência da pré-ênfase.

7.7.1 IAL Dependente de Texto

Para a realização desse teste, faz-se necessário o conhecimento de textos ou frases previamente. Duas frases, por apresentarem grande quantidade de fonemas nasalados e vocalizados, são consideradas adequadas para reconhecimento de locutor (BEZERRA, 1994). São elas: “O prazo tá terminando” e “Amanhã ligo de novo”. Ambas as frases foram selecionadas para os testes audiométricos. Foram gravadas 40 repetições para 20 locutores diferentes (14 do sexo masculino e 6 do sexo feminino). 20 dessas elocuções foram utilizadas para a geração do padrão de cada locutor e outras 20 para a comparação dos padrões, totalizando 800 elocuções. Os resultados dos testes seguem nas Tabelas 12 e 13.

Tabela 12 – Resultado dos testes para o reconhecimento de locutor dependente de texto, frase “O prazo tá terminando”.

Pré-ênfase	Identificações corretas	Identificações incorretas	Eficiência
Sim	320	80	80,0%
Não	333	67	83,25%

Fonte: Produzido pelo autor.

Vê-se que, para ambas as frases, a taxa de identificações foi bem aceitável, dada a complexidade empregada pela metodologia de reconhecimento. Pode-se observar pelos resultados apresentados que na ausência da pré-ênfase o algoritmo tornou-se mais eficiente, indicando que, para a abordagem proposta, as componentes de alta frequência enfatizadas não foram cruciais para o reconhecimento.

Tabela 13 – Resultado dos testes para o reconhecimento de locutor dependente de texto, frase "Amanhã ligo de novo".

Pré-ênfase	Identificações corretas	Identificações incorretas	Eficiência
Sim	322	78	80,25%
Não	340	60	85,0 %

Fonte: Produzido pelo autor.

7.7.2 IAL Independente de Texto

Para esse teste, foram utilizados, para 12 locutores diferentes, 16 textos escolhidos aleatoriamente, com aproximadamente 10 segundos de duração. 8 desses textos foram destinados à geração do padrão de cada locutor e outros 8 foram destinados às comparações dos padrões. Os resultados são sumarizados na Tabela 14.

Tabela 14 – Resultado dos testes para o reconhecimento de locutor independente de texto.

Pré-ênfase	Identificações corretas	Identificações incorretas	Eficiência
Sim	78	18	81,25%
Não	82	14	85,41%

Fonte: Produzido pelo autor.

Pela Tabela 14, observa-se a mesma característica para o reconhecimento independente de texto: a pré-ênfase prejudicou um pouco a taxa de identificações corretas. Contradizendo a tendência normal dos sistemas de reconhecimento, nesse sistema proposto o RAL independente de texto foi mais eficiente do que o RAL dependente de texto. Suspeita-se que o maior tempo de gravação para as elocuções independentes de texto possam ter retido, com mais precisão, a característica de cada elocutor.

Com relação aos parâmetros extraídos, crê-se que as seguintes características desejáveis para um adequado reconhecimento sejam satisfeitas:

- São eficientes na representação de informação do locutor que a produziu;
- São fáceis de se medir;
- São estáveis no tempo;
- Ocorrência natural e frequente.

Por outro lado, outras características ainda precisam ser melhoradas:

- Pouca variação com o ambiente;
- Não susceptível a mímica.

7.7.3 Comparação com o Estado da Arte

Esta subseção discute sobre alguns importantes sistemas de RAL encontrados na literatura científica. Em 1995, Reynolds (REYNOLDS; ROSE, 1995) implementou um sistema de identificação baseado na variabilidade espectral, obtendo uma taxa de precisão de 96.80% com 49 locutores. Em 2009, Revathi, Ganapathy e Venkataramani (REVATHI; GANAPATHY; VENKATARAMANI, 2009) através de uma abordagem de agrupamento iterativo, PLP (Perceptual Linear Predictive cepstrum) e MF-PLP (Mel Frequency PLP), alcançaram taxa de precisão de 91% com 50 locutores escolhidos aleatoriamente no banco de dados TIMIT (TIMIT..., 1992). Em 2009 Chakroborty e Saha (CHAKROBORTY; SAH, 2009) combinando MFCC e IMFCC (MFCC Invertido) baseado no filtro gaussiano, alcançaram taxa de precisão de 97,42% com 131 locutores do banco de dados YOHO (YOHO..., 1994). Em 2010, Saeidi, Mowlaee, Kinnunen e Zheng-Hua (SAEIDI et al., 2009) através da divergência de Kullback-Leibler alcançaram uma taxa de precisão de 97% com 34 locutores. Em 2011, Gomez (GOMEZ, 2011) implementou um sistema de identificação baseado em uma nova rede neural paramétrica, atingindo uma precisão de 94% com 40 locutores. Em 2011, Rao, Prasada e Nagesh (RAO; PRASAD; NAGESH, 2010) fizeram um estudo comparando GMM, HMM e MFCC. A taxa de precisão obtida na melhor condição de teste foi de 99% com 200 indivíduos retirados do banco de dados TIMIT. A Tabela 15 exibe uma comparação das taxas de acertos entre as abordagens citadas e as desenvolvidas neste trabalho.

Tabela 15 – Comparação com o estado da arte.

Abordagem	Taxa de acertos
Reynolds e Rose (1995)	96,80%
Revathi, Ganapathy e Venkataramani (2009)	91%
Chakroborty e Sah (2009)	97,42%
Saeidi et al. (2009)	97%
Gomez (2011)	91%
Rao, Prasad e Nagesh (2010)	91%
Proposta (Independente de Texto)	85%
Proposta (Dependente de Texto)	83,25%

Fonte: Produzido pelo autor.

Acredita-se que os resultados preliminares apresentados são promissores. Mesmo que a taxa de reconhecimento correto nesta versão inicial seja inferior a 95%, restringindo seu uso imediato em algumas aplicações comerciais, aprimoramentos simples podem ser introduzidos (e.g., considerar mais de um sobrevivente em bandas de maior frequência) visando reduzir a taxa de falhas. Pode-se também tentar melhorar a taxa de identificações ao substituir o método de comparação de amostras por um de mais complexidade. Uma análise do comportamento do vetor de características para diferentes falantes, ou seja, quão bem ele consegue "espalhar" timbres diferentes no espaço de características (algo como a característica de decorrelação dos coeficientes mel-cepstrais), também pode ser realizado.

8 DISCUSSÕES E CONCLUSÕES

Como os temas abordados nessa dissertação versam sobre áreas distintas do PDSV, as discussões e conclusões de cada uma serão discutidas independentemente.

8.1 Síntese das Contribuições Pessoais

1. *Vocoder* Proposto
    - Técnica do MPFO;
    - Formato *voz*;
    - Preenchimento Espectral via Distribuição Beta.
2. Sistema de RAL Proposto
    - Técnica do MPFO;
    - *Template Matching*;
    - Análise de Desempenho.

8.2 Sistema de Codificação de Voz Proposto

Esse trabalho teve o objetivo de apresentar uma nova metodologia para codificação de sinais de voz com baixa taxa de bits e reduzida complexidade computacional, para fins de armazenamento ou para uma posterior implementação em tempo real. O intuito foi mostrar que a separação do espectro em bandas de frequências definidas por oitavas (sendo uma analogia às bandas críticas da audição) seguida, subsequentemente, de uma simplificação do espectro, considerando apenas uma amostra de mascaramento por oitava, não descaracterizava os sons vocais e não-vocais, a ponto de os sinais sintetizados não poderem ser compreendidos. Tentou-se através da técnica de preenchimento espectral via distribuição beta aprimorar a qualidade dos sinais gerados pelo sistema. Entretanto observou-se que a técnica ainda introduzia ruído ao sinal, tendo sido necessários artifícios incomuns (composição de sinais obtidos por variações do método proposto, janelamento extra ao sinal) com o propósito de reduzi-lo.

Os testes de todos as elocuções adquiridas foram realizados através da técnica subjetiva ACR a partir de uma classificação MOS. Para isso foi solicitado que pessoas leigas avaliassem a qualidade dos arquivos, produzidos pelo *vocoder* proposto, numa escala de 1(ruim) a 5 (excelente), de uma frase ou um trecho de voz. Esses arquivos foram divididos em quatro tipos:

1. Sinais produzidos sem o uso da técnica de preenchimento espectral;
2. Sinais produzidos com o uso da técnica de preenchimento espectral;
3. Sinais formados pela composição dos dois últimos;
4. Sinais do item 2, usando um janelamento de *Hamming* extra.

Pode-se comprovar que os resultados obtidos nos testes foram satisfatórios, haja vista a reduzida taxa de bits alcançada pelo método, 2.7 kbits/s, e a baixa complexidade empregada na sua implementação. Esses fatos não indicam uma melhoria em comparação aos sistemas já existentes, mas ratifica que o sistema proposto pode ser útil para economizar largura de banda em aplicações requerendo inteligibilidade. Em particular, o sistema é oferecido como uma opção para monitoramento de conversas de voz de longa duração, decorrentes de espionagem autorizada. Foi, também, de essencial relevância para o desenvolvimento do sistema de RAL proposto no trabalho.

8.3 Sistema de Reconhecimento Automático de Locutor Proposto

Nessa parte do trabalho, ficou comprovado que o mascaramento em frequência, além de ser útil em codificação de voz, também pode ser proveitoso no RAL. A síntese do sinal de voz proveniente do *vocoder* proposto, contendo apenas o espectro "ultra-simplificado" (com um único sobrevivente por oitava), fornece um sinal perfeitamente inteligível, a partir do qual se pode reconhecer o falante. Assim, a despeito da qualidade "metálica e artificial" da voz sintética, típica de *vocoders*, as informações suficientes para o reconhecimento não são destruídas. O processo descrito tem como atrativo a simplicidade, pois cada "padrão de voz" é resumido em um único vetor de sete componentes associadas às oitavas distintas. Adicionalmente, o classificador padrão usando os modelos de Markov escondidos é substituído pela simplória técnica de *template matching* via distância euclidiana entre os vetores.

Nos testes realizados, observou-se uma maior taxa de acertos do algoritmo para o reconhecimento independente de texto, para ambas as frases testadas, como normalmente não ocorre em sistemas de RAL. Também, de modo surpreendente para as expectativas iniciais, constatou-se que o filtro de pré-ênfase comprometeu um pouco a eficiência das identificações. De fato, ao enfatizar componentes espectrais mais sensíveis a distorções e ruído, obtém-se melhor qualidade e um sinal de voz mais natural. Porém, os resultados indicam que tais componentes não são cruciais no reconhecimento.

A técnica de mascaramento espectral pleno "lembra" a abordagem de estatística mínima suficiente (FERGUSON, 1967). É como se fossem descartadas as informações espectrais irrelevantes no processo de estimação. Detalhes práticos suplementares merecem investigação. A

transformada de comprimento N=160 usa bases mistas e visando simplicidade de implementação de *hardware* ou DSP, pode-se alterar a duração da janela. Com janelas de 32 ms (ou 16 ms) é possível usar o algoritmo de Cooley-Tukey de base 2 (OPPENHEIM; SCHAFER; BUCK, 1999), restando investigar o impacto na eficiência.

Uma comparação rigorosa entre a complexidade e o compromisso com o desempenho do algoritmo de reconhecimento do locutor entre diferentes técnicas IAL não foi realizada. Porém o principal mérito desta nova abordagem é oferecer uma taxa de reconhecimento razoável, porém demandando uma complexidade computacional substancialmente inferior àquela requerida por outras técnicas consagradas (e.g., HMM, redes neurais, quantização vetorial etc.). Outro aproveitamento possível deste algoritmo é nos casos em que a base de locutores é demasiadamente extensa. Este método rápido pode ser aplicado, selecionando um locutor provável, incluído em uma subclasse de locutores potenciais. Este é então eliminado da base original, repetindo o processo de forma a escolher um segundo locutor potencial. O procedimento é iterado até gerar um número pré-estabelecido de locutores potenciais (base reduzida). Esta aplicação prévia não requer taxas de acerto excessivamente altas, sendo 90% bastante razoável. Um método sofisticado (alto custo computacional e alta eficiência) é aplicado para identificar o locutor dentro desta base reduzida. Outra situação de potencial interesse para este método é no monitoramento em tempo real de telefonemas em prédios (empresas, repartições, etc.) que possuem centrais telefônicas. Com centenas de ligações simultâneas e diferentes ramais, como selecionar gravações (autorizadas) de conversações envolvendo indivíduo sob suspeição? Supõe-se disponível um trecho previamente gravado (e.g., primeiro contato de um sequestrador, chantagista, corrupto, terrorista, etc.) para constituir a informação de treinamento do locutor alvo. Neste caso, taxas de falsa aceitação e falsa rejeição aceitáveis podem ser maiores do que em aplicações comerciais típicas. Assim, situações em tempo real — nas quais há parca disponibilidade de recursos (como em sistemas embarcados) — esta técnica pode se tornar bastante atrativa.

8.4 Sugestões e Trabalhos Futuros

A seguir são apresentadas algumas sugestões para trabalhos futuros, que podem ser realizadas de modo a dar seguimento ao que foi exposto nesta dissertação. Muitas dessas sugestões são linhas de investigação previstas no início deste trabalho, outras são possibilidades que surgiram ao longo do estudo, mas que foram descartadas devido à limitação de tempo. São elas:

8.4.1 Para o *Vocoder*

- Aumentar um pouco a taxa de amostragem do sinal, analisando os efeitos desse procedimento;

- Considerar mais de uma amostra espectral de mascaramento;
- Tentar adaptar esse sistema para o funcionamento em tempo real;
- Empregar a superposição de janelas, para aumentar a correlação entre quadros adjacentes;
- Implementar o algoritmo de teste objetivo PESQ para uma melhor avaliação do codificador;
- Alterar o tamanho da janela de 20 ms para 32 ou 16 ms, podendo-se utilizar o algoritmo de Cooley-Tukey de base 2, reduzindo a complexidade computacional;
- Usar a técnica através de iterações sucessivas, conduzindo a aproximações melhores.

8.4.2 Para a Identificação Automática de Locutor

- Tentar melhorar a taxa de identificações, obtendo um vetor de características com mais componentes, usando também um método de comparação de amostras com mais complexidade;
- Alterar a duração da janela, visando simplicidade de implementação de hardware ou DSP, uma vez que a transformada de comprimento N=160 usa bases mistas. Com janelas de 32 ms (ou 16 ms) é possível usar o algoritmo algoritmo de Cooley-Tukey de base 2, restando investigar o impacto na eficiência;
- Talvez o uso de WAVELETS ao invés da FFT possa render resultados mais favoráveis aos já obtidos nessa dissertação;
- Implementar a Verificação Automática de Locutor com essa técnica proposta, analisando todos os prós e contras dessa nova abordagem;
- Fazer o casamento dessa técnica com outras já existentes, aproveitando as partes mais eficientes de cada uma.

REFERÊNCIAS

ANDRADE, G. A. A. *Wavelets Monocíclicas de Suporte Compacto Construídas a Partir de Distribução Beta*. Dissertação (Mestrado) — Programa de Pós-Graduação em Engenharia Elétrica, Universidade Federal de Pernambuco, Recife, Brasil, 2007. Citado na página 86.

ANDRADE, M. A. R. *Reconhecimento Automático de Comandos Conectados*. Dissertação (Mestrado) — Programa de Pós-Graduação em Engenharia Elétrica, Instituto Militar de Engenharia, Rio de Janeiro, Brasil, 1999. Citado na página 102.

ASSALEH, K. T.; CAMPBELL, W. M. Speaker Identification Using a Polynomial-Based Classifier. *in Proc. 5th International Symposium on Signal Processing and Its Applications (ISSPA'99)*, v. 1, p. 115–118, 1999. Citado na página 101.

ATAL, B. S. Automatic Recognition of Speakers from Their Voices. *Proceedings of the IEEE*, v. 64, p. 460–474, 1976. Citado na página 95.

BARBOSA, D. C. P. *Análise de Sistemas de Telefonia IP em Redes Par-a-Par Sobrepostas*. Dissertação (Mestrado) — Programa de Pós-Graduação em Engenharia Elétrica, Universidade Federal de Pernambuco, Recife, Brasil, 2008. Citado 13 vezes nas páginas 23, 24, 40, 41, 57, 59, 60, 62, 63, 67, 68, 69 e 70.

BEZERRA, M. R. *Reconhecimento Automático de Locutor para Fins Forenses, Utilizando Técnicas de Redes Neurais*. Dissertação (Mestrado) — Programa de Pós-Graduação em Engenharia Elétrica, Instituto Militar de Engenharia, Rio de Janeiro, Brasil, 1994. Citado 2 vezes nas páginas 115 e 120.

BHARIKTAR, S.; KYRIAKAKIS, C. *Immersive Audio Signal Processing*. New York: Springer, 2006. Citado 2 vezes nas páginas 25 e 30.

BRANCO NETO, W. C. *Sistema de Reconhecimento de Som para Afinação de Instrumentos*. Dissertação (Mestrado) — Programa de Pós-Graduação em Engenharia de Produção, Universidade Federal de Santa Catarina, Florianópolis, Brasil, 2000. Citado 3 vezes nas páginas 23, 24 e 25.

CAMPBELL, J. P. Speaker Recognition: A Tutorial. *Proceedings of the IEEE*, v. 85, p. 1437–1462, 1997. Citado 6 vezes nas páginas 20, 94, 95, 96, 97 e 101.

CAVALCANTI, D. L. *Uma Análise Comparativa dos Codificadores/Decodificadores de Voz para Comunicações Digitais*. Dissertação (Mestrado) — Programa de Pós-Graduação em Engenharia Elétrica, Universidade Federal de Pernambuco, Recife, Brasil, 2009. Citado 7 vezes nas páginas 49, 63, 64, 65, 68, 69 e 70.

CHAKROBORTY, S.; SAH, G. Improved Text-Independent Speaker Identification Using Fused MFCC and IMFCC Feature Sets Based on Gaussian Filter. *International Journal of Signal Processing*, v. 5, n. 1, p. 11–19, 2009. Citado na página 122.

CHU, W. C. *Speech Coding Algorithms: Foundation and Evolution of Standardized Coders*. Hoboken, New Jersey: John Wiley & Sons, Inc., 2003. Citado 11 vezes nas páginas 31, 35, 39, 40, 41, 43, 65, 68, 69, 70 e 81.

COOK, N. D. *Tone of Voice and Mind: The Connections Between Intonation, Emotion, Cognition and Consciousness*. Amsterdam: John Benjamins, 2002. Citado na página 25.

COSTA FILHO, A. C. *Análise e Síntese da Fala por Interpolação de Ondas*. Tese (Doutorado) — Programa de Pós-Graduação em Engenharia Elétrica, Universidade Federal de Uberlândia, Uberlândia, Brasil, 2005. Citado 5 vezes nas páginas 44, 45, 61, 63 e 64.

DAN, Z. et al. Speaker Recognition Based on LS-SVM. *The 3rd International Conference on Innovative Computing Information and Control*, p. 25–28, 2008. Citado na página 20.

DE OLIVEIRA, H. M. *Análise de Sinais para Engenheiros — Uma Abordagem via Wavelets*. Rio de Janeiro: Brasport, 2007. Citado na página 77.

FARRELL, K. R.; MAMMONE, R.; ASSALEH, K. Speaker Recognition Using Neural Networks and Conventional Classifiers. *IEEE Trans. Speech, and Audio Processing*, v. 2, n. 1, p. 194–205, 1994. Citado 2 vezes nas páginas 20 e 101.

FERGUSON, T. S. *Mathematical Statistics: A Decision Theoretic Approach*. New York: Academic Press, 1967. Citado na página 125.

FURUI, S. *Digital Speech Processing, Synthesis and Recognition*. 2. ed. New York: Marcel Dekker Inc., 2000. Citado 2 vezes nas páginas 39 e 40.

GOLDBERG, R.; RIEK, L. *A Practical Handbook of Speech Coders*. Boca Raton, FL.: CRC Press, 2000. Citado na página 69.

GOMEZ, P. A Text Independent Speaker Recognition System Using a Novel Parametric Neural Network. *Proceedings of International Journal of Signal Processing, Image Processing and Pattern Recognition*, p. 1–16, 2011. Citado na página 122.

GREENBERG, S. et al. *Speech Processing in the Auditory System*. New York: Springer-Verlag New York, Inc., 2004. Citado na página 43.

HOLMES, J.; HOLMES, W. *Speech Synthesis and Recognition*. 2. ed. Bristol, PA: Taylor & Francis, Inc., 2002. Citado 5 vezes nas páginas 30, 32, 35, 41 e 62.

HUANG, V. T. L. et al. A New Vocoder Based on AMR 7.4 kbits/s Mode in Speaker Dependent Coding System. *Proc. Int. Conf. Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing*, v. 6, p. 163–167, 2008. Citado na página 61.

ITU-T. *Coding Speech at 16 kbits/s Using Low-Delay Code Excited Linear Prediction*. Recommendation G.728, 1992. Citado na página 69.

ITU-T. *40, 32, 24, 16 kbits/s Adaptive Differential Pulse Code Modulation (ADPCM)*. Recommendation G.726, 1996. Citado na página 69.

ITU-T. *Objective Quality Measurement of Telephone-band (300-3400 Hz) Speech Codecs*. Recommendation P.861, 1998. Citado na página 66.

ITU-T. *Pulse Code Modulation (PCM) of Voice Frequencies*. Recommendation G.711, Geneva, 1993. Citado na página 68.

KAHRS, M.; BRANDENBURG, K. *Applications of Digital Signal Processing to Audio and Acoustics*. New York: Kluwer Academic Publishers, 2002. Citado 2 vezes nas páginas 37 e 38.

KONDOZ, A. M. *Digital Speech: Coding for Low Bit Rate Communication Systems*. 2. ed. Chichester, UK: John Wiley & Sons, 2004. Citado 15 vezes nas páginas 44, 45, 47, 50, 52, 53, 55, 56, 57, 61, 66, 68, 79, 80 e 99.

LATHI, B. P. *Modern Digital and Analog Communication Systems*. New York: Holt, Rinehart, and Winston, 1989. Citado na página 77.

LIPORACE, L. A. Maximum Likelihood Estimation for Multivariate Observations of Markov Sources. *IEEE Trans. Informat. Theory*, v. IT-28, n. 5, p. 729–734, 1982. Citado na página 102.

MINKER, W.; BENNACEF, S. *Speech and Human-Machine Dialog*. 1. ed. New York: Kluwer Academic Publishers, 2004. Citado na página 19.

MIRANDA, E. R. *The Informatics Handbook: A Guide to Multimedia Communications and Broadcasting*. 1. ed. Oxford, UK: Springer Science & Business Media, 1996. Citado na página 19.

MIRANDA, E. R. *Computer Sound Design: Synthesis Techniques and Programming*. 2. ed. Oxford, UK: Focal Press, 2002. Citado na página 19.

OLIVEIRA, M. P. B. *Verificação Automática de Locutor, Dependente de Texto, Utilizando Sistemas Híbridos MLP/HMM*. Dissertação (Mestrado) — Programa de Pós-Graduação em Engenharia Elétrica, Instituto Militar de Engenharia, Rio de Janeiro, Brasil, 2001. Citado 4 vezes nas páginas 102, 104, 105 e 107.

OPPENHEIM, A. V.; SCHAFER, R. W.; BUCK, J. R. *Discrete-Time Signal Processing*. 2. ed. Upper Saddle River, NJ: Prentice Hall, 1999. Citado 9 vezes nas páginas 55, 56, 58, 77, 79, 81, 82, 115 e 126.

PARANAGUÁ, E. D. S. *Reconhecimento de Locutor Utilizando Modelos de Markov Escondidos Contínuos*. Dissertação (Mestrado) — Programa de Pós-Graduação em Engenharia Elétrica, Instituto Militar de Engenharia, Rio de Janeiro, Brasil, 1997. Citado 7 vezes nas páginas 20, 93, 99, 100, 102, 104 e 108.

PEACOCKE, R. D.; GRAF, D. H. An Introduction to Speech and Speaker Recognition. *IEEE Computer Society Press*, v. 23 (8), p. 26–33, 1990. Citado na página 20.

PELTON, G. E. *Voice Processing*. New York: McGraw-Hill, Inc., 1993. Citado 4 vezes nas páginas 40, 41, 62 e 115.

PENHA, R. *Avaliação Audio–Vestibular in: Penha R.* Lisboa: Otorrinolaringologia, 1996. Citado 2 vezes nas páginas 26 e 27.

PETRY, A. *Reconhecimento Automático de Locutor Utilizando Medidas Invariantes Dinâmicas Não-Lineares*. Tese (Doutorado) — Programa de Pós-Graduação em Ciência da Computação, Universidade Federal do Rio Grande do Sul, Porto Alegre, Brasil, 2002. Citado 7 vezes nas páginas 93, 94, 95, 98, 99, 101 e 114.

POPE, S. P.; SOLBERG, B.; BRODERSEN, R. W. A Single-Ship Linear-Predictive-Coding Vocoder. *IEEE Journal of Solid-State Circuits*, v. 22, n. 3, p. 479–487, 1987. Citado na página 62.

RABINER, L.; JUANG, B.-H. *Fundamentals of Speech Recognition*. Englewood Cliffs, NJ: Prentice-Hall, 1993. Citado 11 vezes nas páginas 40, 43, 46, 99, 101, 103, 104, 105, 106, 107 e 114.

RABINER, L. R. Applications of Voice Processing to Telecommunications. *Proceedings of IEEE*, v. 82, n. 2, p. 199–228, 1994. Citado 2 vezes nas páginas 98 e 99.

RABINER, L. R.; SCHAFER, R. W. *Digital Processing of Speech Signals*. 1. ed. New Jersey: Prentice-Hall, 1978. Citado 7 vezes nas páginas 19, 39, 40, 77, 80, 114 e 115.

RABINER, L. R.; SCHAFER, R. W. Introduction to Digital Speech Processing. *Foundations and Trends in Information Retrieval*, v. 1, n. 1–2, p. 1–194, 2007. Citado 6 vezes nas páginas 43, 44, 45, 47, 50 e 58.

RAO, R. R.; PRASAD, V. K.; NAGESH, A. Performance Evaluation of Statistical Approaches for Text-Independent Speaker Recognition Using Source Feature. *InterJRI Computer Science and Networking)*, v. 2, n. 1, p. 8–13, 2010. Citado na página 122.

REVATHI, A.; GANAPATHY, R.; VENKATARAMANI, Y. Text Independent Speaker Recognition and Speaker Independent Speech Recognition Using Iterative Clustering Approach. *International Journal of Computer Science & Information Technology*, v. 1, n. 2, p. 30–42, 2009. Citado na página 122.

REYNOLDS, D. A.; ROSE, R. C. Robust Text-Independent Speaker Identification Using Gaussian Mixture Speaker Models. *IEEE Trans. Speech Audio Process*, v. 3, p. 72–83, 1995. Citado 2 vezes nas páginas 20 e 122.

RIX, A. W. et al. Perceptual Evaluation of Speech Quality (PESQ), an Objective Method for End-to-End Speech Quality Assessment of Narrowband Telephone Networks and Speech Codecs. *ITU-T Recommendation*, v. 862, 2001. Citado na página 66.

ROSEMBERG, A. E. Automatic Speaker Verification: A Review. *Proceedings of the IEEE*, v. 64, p. 475–487, 1984. Citado 2 vezes nas páginas 95 e 114.

RUMSEY, F.; MCCORMICK, T. *Sound and Recording: An Introduction*. 5. ed. Burlington, MA: Focal Press, 2006. Citado 6 vezes nas páginas 23, 24, 30, 32, 35 e 37.

SAEIDI, R. et al. Signal-to-Signal Ratio Independent Speaker Identification for Co-Channel Speech Signals. *Proceedings of International Conference on Pattern Recognition (ICPR 2009)*, p. 4565–4568, 2009. Citado na página 122.

SALAMI, R. et al. ITU-T G.729 Annex A: Reduced Complexity 8 kb/s CS-ACELP Codec for Digital Simultaneous Voice and Data. *IEEE Communications Magazine*, p. 56–63, 1997. Citado na página 70.

SCHROEDER, M. R. Dudley, Homer W.: A Tribute. *Signal Processing*, v. 3, p. 187–188, 1981. Citado na página 19.

SHAO, Y.; WANG, D. Robust Speaker Recognition Using Binary Time-Frequency Masks. *IEEE International Conference on Acoustic, Speech and Signal Processing*, I, p. 645–648, 2006. Citado na página 20.

SILVA, D. D. C. *Desenvolvimento de um IP Core de Pré-Processamento Digital de Sinais de Voz para Aplicações em Sistemas Embutidos*. Dissertação (Mestrado) — Programa de Pós-Graduação em Informática, Universidade Federal de Campina Grande, Campina Grande, Brasil, 2006. Citado 4 vezes nas páginas 77, 78, 80 e 114.

SMITH, S. W. *Digital Signal Processing – A Practical Guide for Engineers and Scientists*. 3. ed. Burlington, MA: Newnes, 2003. Citado 5 vezes nas páginas 24, 25, 27, 30 e 40.

SONG, F. K. et al. A Vector Quantization Approach to Speaker Recognition. *in Proc. Int. Conf. Acoustics, Speech and Signal Processing*, p. 387–390, 1985. Citado na página 101.

SOTERO FILHO, R. F. B.; DE OLIVEIRA, H. M. Reconhecimento de Locutor Baseado em Mascaramento Pleno em Frequência por Oitava. *7º Congresso de Engenharia de Áudio, São Paulo, SP. Anais do AES Brasil*, p. 61–66, 2009. Citado na página 72.

SOTERO FILHO, R. F. B.; DE OLIVEIRA, H. M.; CAMPELLO DE SOUZA, R. M. A Full Frequency Masking Vocoder for Legal Eavesdropping Conversation Recording. *XXXV Congresso Nacional de Matemática Aplicada e Computacional, Natal, RN, Brasil*, v. 3, 2014. Citado na página 72.

SOUZA, M. A. et al. Avaliação de Técnicas de Codificação de Voz para Voip. *XIV Encontro Regional de Informática, Guarapuava, PR*, 2007. Citado na página 67.

SPANIAS, A.; PAINTER, T.; ATTI, V. *Audio Signal Processing and Coding*. 1. ed. New Jersey: John Wiley & Sons, Inc., 2007. Citado 15 vezes nas páginas 19, 33, 34, 35, 36, 37, 38, 44, 45, 46, 60, 63, 64, 67 e 68.

TIMIT Speech Database. 1992. Disponível em: <http://www.ldc.upenn.edu>. Citado na página 122.

TISHBY, N. Z. On Application of Mixture AR Hidden Markov Models to Text Independent Speaker Recognition. *IEEE Trans. Signal Processing*, v. 39, n. 3, p. 563–570, 1991. Citado 2 vezes nas páginas 20 e 101.

VASEGHI, S. V. *Multimedia Signal Processing: Theory and Applications in Speech, Music and Communications*. Chichester, UK: John Wiley & Sons, Ltd, 2007. Citado 15 vezes nas páginas 25, 26, 27, 28, 29, 30, 31, 32, 35, 36, 39, 40, 43, 46 e 59.

WANG, N. et al. Robust Speaker Recognition using Both Vocal Source and Vocal Tract Features Estimated from Noisy Input Utterances. *IEEE International Symposium on Signal Processing and Information Technology*, 2007. Citado na página 20.

WANG, S.; SEKEY, A.; GERSHO, A. An Objective Measure for Predicting Subjective Quality of Speech Coders. *IEEE J. Select. Areas Commun*, v. 10, n. 5, p. 819–829, 1992. Citado na página 66.

WATKINSON, J. *The Art of Digital Audio*. 3. ed. Woburn, MA: Focal Press, 2001. Citado 3 vezes nas páginas 26, 29 e 35.

WOLF, J. J. Efficient Acoustic Parameters for Speaker Recognition. *Journal of the Acoustic Society of America 51*, v. 6 (Part 2), p. 2044–2056, 1972. Citado na página 98.

YOHO Speech Database. 1994. Disponível em: <http://www.ldc.upenn.edu>. Citado na página 122.

Apêndices

APÊNDICE A – ARTIGOS PUBLICADOS

Este trabalho está relacionado às seguintes publicações:

- SOTERO FILHO, R. F. B.; DE OLIVEIRA, H. M. Reconhecimento de Locutor Baseado em Mascaramento Pleno em Frequência por Oitava. 7^{o} *Congresso de Engenharia de Áudio, São Paulo, SP. Anais do AES Brasil*, p. 61–66, 2009.
- SOTERO FILHO, R. F. B.; DE OLIVEIRA, H. M.; CAMPELLO DE SOUZA, R. M. A Full Frequency Masking Vocoder for Legal Eavesdropping Conversation Recording. *XXXV Congresso Nacional de Matemática Aplicada e Computacional, Natal, RN, Brasil*, v. 3, 2014.

Sociedade de Engenharia de Áudio

Artigo de Congresso

Apresentado no 7º Congresso de Engenharia de Áudio
13ª Convenção Nacional da AES Brasil
26 a 28 de Maio de 2009, São Paulo, SP

Reconhecimento de Locutor baseado em Mascaramento Pleno em Freqüência por Oitavas

Sotero Filho, R. F. B. e de Oliveira, H. M.
Departamento de Eletrônica e Sistemas
Universidade Federal de Pernambuco - UFPE
Recife, Pernambuco, 50711-970, Brasil
rsotero@hotmail.com hmo@ufpe.br

RESUMO

Este artigo propõe um novo método de baixa complexidade computacional para reconhecimento de locutor, baseando-se em uma das propriedades-chave da percepção auditiva humana: o mascaramento acústico em freqüência. O vetor característico dos quadros do sinal de voz é representado pela média das amplitudes dos tons de mascaramento em cada oitava. Ambos os tipos de reconhecimento de locutor (de texto dependente e de texto independente) são estudados. Os resultados confirmam que o algoritmo proposto oferece um compromisso entre a complexidade e a taxa de identificações corretas, sendo atrativo para aplicações em sistemas embarcados.

ABSTRACT

This paper introduces a novel and low-complexity speaker identification technique. It is based on one of the key-properties of the human hearing perception: the auditory frequency masking. The feature vectors of voice frames are merely represented by the average amplitude of the greatest spectral samples within each octave. Both text-dependent and text-independent speaker recognition is investigated. Results corroborate a tradeoff between recognition efficiency and complexity of this kind of vocoder-based systems, which turns it attractive for embedded systems.

0 INTRODUÇÃO

Enquanto humanos, somos capazes de distinguir pessoas meramente ouvindo-as falar. Diferenças (ainda que sutis) de timbre, sotaque e/ou entonação, habilitam-nos a distinguir uma pessoa de outra apenas pela sua voz. Geralmente, curtos trechos de fala (2 a 3 segundos) são largamente suficientes para o reconhecimento de uma voz familiar.

A área de processamento de voz, que torna possível o reconhecimento de pessoas pela voz por meio de máquinas é chamada de "reconhecimento automático de locutor" (RAL). No RAL, determina-se a identidade de uma pessoa através da voz, com o propósito de controlar/restringir o acesso a redes, computadores, bases de dados, bem como restringir a disponibilização de informações confidenciais para pessoas não autorizadas, dentre várias outras aplicações [1].

Um sistema que trabalha com RAL calcula (por algum critério específico) a similaridade entre as características da voz do locutor que se deseja reconhecer, com as características de voz de um conjunto de locutores previamente armazenadas pelo sistema de reconhecimento.

O RAL divide-se em Verificação Automática de Locutor (VAL) e Identificação Automática de Locutor (IAL). Na VAL, faz-se uso de uma máquina para verificar

a identidade da voz de uma pessoa que a reivindicou [2]. Na literatura há outras denominações para a VAL, incluindo-se: verificação de voz, autenticação de locutor e autenticação de voz. Na VAL pode haver erros de dois tipos: a falsa aceitação (FA) de um locutor impostor, ou a falsa rejeição (FR) de um locutor verídico, [3], [4].

Na IAL não há a reivindicação de autenticidade. O sistema é que deverá decidir, dentre um determinado número *N* de locutores, qual o usuário correto ou se o mesmo é desconhecido dentre *N* possíveis locutores cadastrados [2]. A IAL pode ser implementada com rejeição ou sem rejeição. No primeiro caso, é estabelecido um limiar para cada usuário. Para o locutor ser considerado autêntico, a similaridade entre as características de sua elocução teste e as características extraídas de seu padrão deverá superar esse limiar. Em caso negativo, o locutor é considerado um impostor. Este trabalho é focado exclusivamente na Identificação Automática de Locutor sem rejeição.

O reconhecimento de locutor pode ser feito através do uso de um texto conhecido ou pode ser feito através de um texto arbitrário. No primeiro caso (reconhecimento dependente de texto), o texto ou frase é previamente conhecido pelo sistema que o utilizará para teste e para o treinamento. No segundo (reconhecimento independente de texto), não há especificação de texto. A tarefa de verificação é realizada com a comparação de um texto falado no momento do reconhecimento, com outro texto distinto, previamente gravado pelo sistema.

Recentes pesquisas na área de reconhecimento de locutor visam reduzir a complexidade computacional de métodos já existentes, e que invariavelmente requerem grande carga computacional para o processamento. O trabalho publicado recentemente, [5], baseado em LS-SVM (*The Least Square Support Vector Machine*), transforma um problema de programação quadrática, do convencional *Support Vector Machine* (SVM), num problema de programação linear, reduzindo assim a complexidade computacional. Outras publicações recentes procuram aprimorar o desempenho dos métodos de reconhecimento em ambientes ruidosos, como em [6] e [7].

Visando trabalhar com uma técnica de baixa complexidade e com alta simplicidade de implementação, este trabalho apresenta os resultados obtidos utilizando-se técnicas de processamento digital de sinais para a identificação automática de pessoas pela voz, baseado em uma técnica nomeada de "mascaramento em freqüência por oitava".

Inicialmente são introduzidas as técnicas adotadas para a realização do pré-processamento do sinal e extração das características representativas do sinal pré-processado. Posteriormente, o processo de reconhecimento é descrito. Concluindo, são analisados os resultados obtidos, com a implementação prática das técnicas descritas neste artigo para o reconhecimento de falantes.

1 AQUISIÇÃO DE SINAIS DE VOZ

O processo de identificação do locutor tem início com a gravação das elocuções para o processamento. Isso é realizado utilizando um microfone, cuja saída está conectada a uma placa de som instalada em um computador. Essa tem a função de converter o sinal analógico de voz em amostras igualmente espaçadas no tempo, a uma taxa que pode ser previamente escolhida.

Do teorema da amostragem de Shannon [8], sabe-se que para não haver perda de informação, o sinal banda limitada em f_m Hz deve ser amostrado a uma taxa de pelo menos $2f_m$ amostras eqüiespaçadas por segundo. Tipicamente, a energia de um sinal de voz é concentrada numa faixa de freqüência de até 5 kHz, ainda que a realização (pronúncia) típica de fonemas fricativos (e.g. /s/) possua substancial parte da energia espectral acima desta freqüência. No entanto, como isso ocorre apenas para sons de natureza ruidosa, eles contêm pouca informação sobre o locutor (que se concentra mais nos sons vocálicos). Diante disso, em concordância com o Teorema da amostragem, um valor aceitável para amostragem de um sinal de voz típico na aplicação em vista deveria ser em torno de 10 kHz [9]. O valor escolhido nesse trabalho foi o de 8 kHz, utilizando 16 bits de resolução e 1 canal, *Mono*.

2 PRÉ-PROCESSAMENTO DO SINAL DE VOZ

Após adquirirem-se os dados e convertê-los em amostras digitais, passa-se à fase do pré-processamento dos mesmos. Essa etapa compreende a pré-ênfase, a detecção de pontos extremos (*endpoints),* segmentação dos dados em quadros (*frames*) e janelamento.

2.1 Pré-ênfase

Devido a características fisiológicas do sistema de produção da fala, o sinal de voz irradiado pelos lábios apresenta uma atenuação de aproximadamente 6 dB/ oitava nas altas freqüências. O filtro de pré-ênfase serve para compensar esta atenuação, antes da análise espectral, melhorando a eficiência da análise [10]; sendo a audição menos sensível a freqüências acima de 1 kHz do espectro, a pré-ênfase amplifica esta área do espectro, auxiliando os algoritmos de análise espectral na modelagem dos aspectos perceptualmente importantes do espectro da voz [11]. A resposta em freqüência do filtro pode ser representada por:

$$H(z) = 1 - az^{-1}. \quad (1)$$

Neste caso, a saída da pré-ênfase *y(n)* está relacionada à entrada *x(n)* pela equação diferença [12]:

$$y(n) = x(n) - a.x(n-1) \quad (2)$$

para $1 \leq n < M$, em que *M* é o número de amostras do sinal amostrado *x(n)*, *y(n)* é o sinal pré-enfatizado e a constante "*a*" é normalmente escolhido entre 0,9 e 1. No trabalho foi adotado um valor de "*a*" igual a 0,95 [11].

2.2 Detecção de pontos extremos (*endpoints*)

A fim de reduzir o tempo de processamento, e evitar que o ruído de fundo que ocorra antes e depois do sinal de voz prejudique o desempenho do reconhecimento [13], far-se-á o uso de um algoritmo (*voice activity detection* – VAD), que detecta os pontos extremos do sinal. Esse algoritmo baseia-se na metodologia criada por Rabiner e Sambur em 1975 e faz uso de duas medidas do sinal de voz: a energia e a taxa de cruzamento do zero obtidas em janelas de 10 ms

de duração do sinal. Um intervalo de 100 ms no início da elocução (10 janelas) é utilizado para efetuar uma estatística do ruído de fundo [14].

2.3 Segmentação dos dados em quadros e Janelamento

Após a detecção dos pontos extremos, o sinal de voz deve ser particionado em pequenos segmentos (*frames)* bem definidos, com o propósito de se obter trechos de voz razoavelmente assumidos como estacionários. Isso porque, sendo o sinal de voz um processo estocástico, e sabendo-se que o trato vocal muda de forma muito lentamente na voz contínua, muitas partes da onda acústica podem ser assumidas como estacionárias num intervalo de curtíssima duração (entre 10 e 40 ms). Este intervalo caracteriza o tamanho da janela a ser usada [15]. Neste trabalho, o tamanho da janela adotada será de 20 ms, um valor típico de muitas aplicações envolvendo voz.

O janelamento do sinal tem o objetivo de amortecer o efeito do "fenômeno Gibbs" [10], [16] que surge devido à descontinuidade das janelas [15].

Para o contexto da produção da voz, as características apresentadas, referentes ao janelamento de *Hamming*, mostram que este tipo de janela é mais eficiente quando comparada às janelas Retangular e de *Hanning*, com uma aproximação da janela ideal [16]. Assim sendo, essa foi a janela utilizada neste trabalho.

3 METODOLOGIA EMPREGADA

A idéia proposta baseou-se em umas das propriedades psico-acústicas da audição humana: o mascaramento auditivo ou "audibilidade diminuída de um som devido à presença de outro", podendo este ser em freqüência – foco do nosso trabalho – ou no tempo. O mascaramento auditivo em freqüência ocorre quando um som que normalmente poderia ser ouvido é mascarado por outro, de maior intensidade, que se encontra em uma freqüência próxima. Ou seja, o limiar de audição é modificado (aumentado) na região próxima à freqüência do som que causa a ocorrência do mascaramento, sendo que isto se deve à limitação da percepção de freqüências do sistema auditório humano.

Em função deste comportamento, o que método de reconhecimento proposto fará, *a priori*, é identificar casos de mascaramento em freqüência no espectro do sinal particionado em oitavas, e descartar sinais que "não seriam audíveis" devido a este fenômeno.

A tendência predominante dos padrões de reconhecimento existentes em utilizar coeficientes cepstrais e mel-cepstrais [16] para caracterizar um quadro de voz, não será aqui adotada. Em vez disso, utilizaremos a fração média das amplitudes das freqüências de mascaramento por oitava, como uma representação do padrão de voz. Essa nova abordagem reduz significativamente o volume de dados para processamento.

As algoritmos desenvolvidos para a extração das características do quadro de voz, geração e comparação dos padrões foram todos escritos na linguagem MATLAB® por ser uma linguagem muito difundida nos meios acadêmicos e de fácil implementação.

A seguir a metodologia abordada é descrita.

3.1 Extração das características do quadro de voz

O sinal gravado e amostrado (a uma taxa de 8 kHz) passará pelas etapas descritas no item 2, ou seja, da pré-ênfase, detecção dos pontos extremos, segmentação e janelamento. Posteriormente, para cada segmento do arquivo de voz janelado, será aplicada uma FFT de comprimento 160 (número de amostras contidas em um quadro de 20 ms de voz), obtendo-se assim a representação no domínio da freqüência do sinal, para cada quadro. Subseqüentemente, o espectro da magnitude do sinal é dividido em oitavas. A primeira oitava correspondendo à faixa de freqüências de 32 Hz – 64 Hz, a segunda indo de 64 Hz – 128 Hz, e assim por diante, até a sétima que corresponde à faixa de 2048 Hz a 4096 Hz.

Como se está fazendo uso de uma taxa de amostragem de 8 kHz, cada amostra da magnitude do espectro corresponderá a uma amostra espectral múltipla de 50 Hz, sendo que a primeira amostra irá representar a componente DC de cada quadro de voz. Já que as raias espectrais caminham a passos de 50 Hz, a primeira oitava (de 32 Hz a 64 Hz), será representada pela amostra espectral de 50 Hz, a segunda oitava (64 Hz a 128 Hz) pela amostra de 100 Hz, a terceira (de 128 Hz a 256 Hz) pelas amostras de 150 Hz, 200 Hz e 250 Hz, e assim por diante.

Tabela 1 – Número de freqüências estimadas pela DFT de comprimento 160 em cada oitava do espectro vocal.

Oitava (Hz)	*# amostras espectrais/oitava*
32 - 64	1
64 - 128	1
128 - 256	3
256 - 512	5
512 - 1024	10
1024 - 2048	20
2048 - 4096	39

Terminado esse procedimento inicial, o algoritmo irá agora buscar em cada oitava, em todos os sete sub-bandas de voz do sinal, o ponto da FFT de maior magnitude, i.e., aquele que irá (potencialmente) mascarar os demais. Essa amostra espectral passará a ser o único representante dentro de cada oitava (por opção de complexidade reduzida). As demais serão descartadas, assumindo valor espectral nulo. O total de 80 freqüências oriundas da estimativa da DFT com N=160 é reduzido para 7 sobreviventes (retendo menos do que 5% das componentes espectrais). Portanto, cada quadro, agora, será representado, no domínio freqüencial, por 7 tons puros de mascaramento auditivo, um para cada oitava. Esta técnica é denominada aqui de mascaramento pleno de freqüência.

Definindo o vetor inicial de amostras espectrais, no i-ésimo quadro de voz, por $oct_j^{(i)}$ em que j representa o índice da oitava, tem-se:

$$oct_j^{(i)} = \left[a_{j,1}^{(i)}\, a_{j,2}^{(i)} a_{j,3}^{(i)} \ldots a_{j,N_j}^{(i)}\right], \quad \begin{cases} i = 1,2 \ldots, n \\ j = 1,2\ldots,7 \end{cases} \qquad (3)$$

sendo, $a_{j,k}^{(i)}$ a amplitude do k-ésimo ponto da FFT, na janela i e oitava j e N_j o número de amostras da j-ésima oitava.

Aplicando-se o procedimento de busca da amostra espectral de maior magnitude, vamos obter um novo vetor

$new\_oct_j^{(i)}$ sintetizado contendo N_j-1 zeros, e a única componente da amostra de mascaramento espectral correspondente ao *max* ($a_{j,k}^{(i)}$):

$$new\_oct_j^{(i)} = [0\ 0\ \ldots\ max\ (a_{j,k}^{(i)})\ldots\ 0],\ k = 1,2\ldots N_j. \quad (4)$$

A Figura 1 mostra o módulo do espectro de um quadro, de 20 ms, de uma locução usada para teste, antes e depois da simplificação por tons de mascaramento psico-acústico.

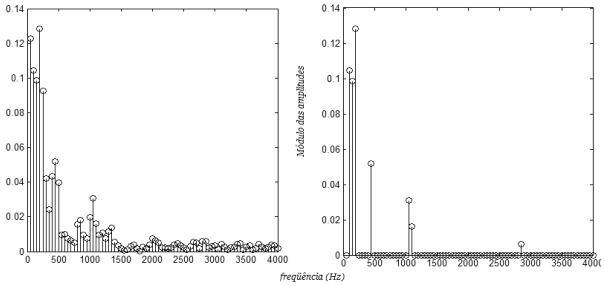


Figura 1 Representação do espectro de freqüência de um quadro de voz, para antes e depois do processo de mascaramento auditivo.

Obtidos todos os vetores $new\_oct_j^{(i)}$, o algoritmo obtém, para cada oitava, uma matriz M_j cujas linhas são formadas por todos os *n* vetores $new\_oct_j^{(i)}$ do arquivo. Esse procedimento será útil para calcular as médias dos "tons" de mascaramento.

$$M_j = (m_{j,k}) = \begin{bmatrix} new\_oct_j^{(1)} \\ new\_oct_j^{(2)} \\ new\_oct_j^{(3)} \\ \vdots \\ new\_oct_j^{(n)} \end{bmatrix} \quad (5)$$

$$= \begin{bmatrix} 0 & 0 & \max(a_{j,k}^{(1)}) & \ldots & 0 \\ 0 & 0 & 0 & \vdots & \max(a_{j,k}^{(2)}) \\ \max(a_{j,k}^{(3)}) & 0 & 0 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ldots & \vdots \\ 0 & \max(a_{j,k}^{(N_j)}) & 0 & \ldots & 0 \end{bmatrix}$$

Calculando-se a média de cada coluna da matriz M_j, obtém-se a participação média de cada amostra espectral de mascaramento (múltiplos de 50 Hz), no sinal de voz, resultando no vetor:

$$\overline{m_j} = [m_{j,1}\ m_{j,2}\ \ldots m_{j,Nj}], \quad (6)$$

em que, $m_{j,k} = \frac{1}{n}\sum_{i=1}^{n} \max(a_{j,k}^{(i)})$, e *k* representa o índice no qual existam amostras espectrais de mascaramento.

Em seguida, todas as componentes do vetor, $\overline{m_j}$ são somadas. Essa soma representará a participação média dos "tons" de mascaramento dentro de sua respectiva oitava.

$$s_j = \sum_{k=1}^{N_j} m_{j,k}\ . \quad (7)$$

Esses s_j assim definidos formarão o vetor s_{total}:

$$s_{total} = [s_1\ s_2 \ldots s_7]. \quad (8)$$

Os parâmetros obtidos pela etapa anterior são diretamente proporcionais aos níveis de energia dos sinais coletados, fator que pode deturpar a classificação incorretamente. Para realizar a normalização dessas amplitudes, faz-se a divisão do vetor s_{total} pela soma de todas as suas componentes.

Normalizando, o vetor s_{total} encontra-se, enfim, o vetor característica do sinal de voz, com apenas 7 componentes, representantes do número de oitavas, o qual será usado para a comparação com as locuções testes:

$$s_{norm} = \frac{1}{\sum_{j=1}^{7} s_j}[s_1\ s_2\ \ldots s_7]. \quad (9)$$

3.2 Geração dos padrões de locutores

A geração do padrão de cada locutor é feita obtendo a média de todos os vetores representantes das características do sinal de voz, das elocuções reservadas para o treinamento.

3.3 Comparação dos padrões de voz

Como última etapa do processo de identificação, tem-se a comparação entre dois vetores. A comparação é realizada através do cálculo da distorção entre eles. Há várias medidas de distorção entre vetores que podem ser utilizadas em reconhecimento de locutor. A medida de distorção mínima ou euclidiana, a medida mais conhecida, foi aquela utilizada. Simulações de desempenho pela alteração das métricas de comparação dos locutores precisam ser conduzidas, a fim de selecionar a mais adequada, i.e., aquela de melhor compromisso complexidade versus taxa de reconhecimento. A seleção do locutor é realizada com base na técnica simples de *template matching* via distância euclidiana entre o vetor de característica e os vetores armazenados para os locutores cadastrados. A Figura 2 no anexo ilustra o algoritmo de reconhecimento de locutor proposto neste trabalho.

4 RESULTADOS

Foram realizados dois tipos de testes. No primeiro deles, a identificação dos locutores é feita fazendo uso de uma mesma frase padrão para todos os locutores (reconhecimento dependente de texto). No segundo caso, a identificação é feita com textos escolhidos aleatoriamente no momento da gravação (reconhecimento independente de texto). Todas as gravações foram realizadas com o mesmo microfone, numa sala que não teve nenhuma preparação especial destinada à redução de ecos ou mesmo a eliminação total de ruído de fundo. Nos experimentos realizados a eficiência do algoritmo foi também testada na ausência da pré-ênfase. Os resultados são comentados a seguir.

4.1 IAL Dependente de Texto

Para a realização desse teste faz-se necessário o pré-conhecimento de textos ou frases. Duas frases são consideradas adequadas para reconhecimento de locutor, por apresentarem grande quantidade de fonemas nasalados e vocalizados [15]. São elas: "O prazo tá terminando" e "Amanhã ligo de novo". A segunda opção foi à selecionada para realização dos testes audiométricos.

Foram gravadas 40 repetições para 10 locutores diferentes (7 do sexo masculino e 3 do sexo feminino), das quais 20 serão utilizadas para a geração do padrão de cada locutor e outros 20 serão utilizados para a comparação dos padrões, totalizando 400 elocuções. Os resultados dos testes seguem na Tabela 2.

Tabela 2 – Resultado dos testes para o reconhecimento de locutor dependente de texto.

Pré-ênfase	*Identificações corretas*	*Identificações incorretas*	*Eficiência*
Sim	174	26	87,0 %
Não	183	17	91,5%

Como se pode observar pela Tabela 2, na ausência da pré-ênfase o algoritmo tornou-se mais eficiente.

4.2 IAL Independente de Texto

Nesse teste, utilizaram-se oito textos, escolhidos aleatoriamente, de aproximadamente 10 segundos de duração, para 12 locutores diferentes. Quatro desses textos foram usados para a geração do padrão de cada locutor. Os outros quatro textos foram utilizados para as comparações dos padrões. Os resultados são sumarizados na Tabela 3.

Tabela 3 – Resultado dos testes para o reconhecimento de locutor independente de texto.

Pré-ênfase	*Identificações corretas*	*Identificações incorretas*	*Eficiência*
Sim	39	9	81,25 %
Não	44	4	91,66 %

5 DISCUSSÃO E CONCLUSÕES

Ficou constatado nesse artigo que o mascaramento em freqüência fazendo uso de um único ponto da FFT sobrevivente por oitava pode ser útil no reconhecimento de locutor. A síntese do sinal de áudio proveniente de um vocoder contendo apenas o espectro "ultra-simplificado" (com único sobrevivente por oitava, e.g. Fig.1) fornece um sinal perfeitamente inteligível, a partir do qual se reconhece facilmente o falante. Assim, a despeito da qualidade "metálica e artificial" da voz sintética (vide arquivo anexo *sotero-reconhecimento-2.wav*), típica de vocoders, as informações suficientes para o reconhecimento não são destruídas. O processo descrito tem como atrativo a simplicidade, pois cada "padrão de voz" é resumido em um único vetor de sete componentes associadas às oitavas distintas. Adicionalmente, o classificador padrão usando cadeias de Markov escondidas (HMM) é substituído pela técnica simples de *template matching* via distância euclidiana entre os vetores. Foi observada uma maior taxa de acertos do algoritmo para o reconhecimento dependente de texto. De modo surpreendente para as expectativas iniciais, constatou-se que o filtro de pré-ênfase comprometeu um pouco a eficiência das identificações. De fato, ao enfatizar componentes espectrais mais sensíveis a distorções e ruído, obtém-se melhor qualidade e um sinal de voz mais natural. Porém, os resultados indicam que tais componentes não são cruciais no reconhecimento. Os resultados preliminares apresentados são promissores. Mesmo que a taxa de reconhecimentos corretos nesta versão inicial seja inferior a 95% − restringindo seu uso imediato em algumas aplicações comerciais– aprimoramentos simples podem ser introduzidos (e.g. considerar mais de um sobrevivente em bandas de maior freqüência) visando reduzir a taxa de falhas. Este tópico encontra-se atualmente sob investigação, além de uma análise do comportamento do vetor de características para diferentes falantes, ou seja, quão bem ele consegue "espalhar" timbres diferentes no espaço de características (algo como a característica de decorrelação dos coeficientes MFCC).

A técnica de mascaramento espectral pleno "lembra" a abordagem de estatística mínima suficiente [17]. É como se fossem descartadas as informações espectrais irrelevantes no processo de estimação. Detalhes práticos suplementares merecem investigação. A transformada de comprimento N=160 usa bases mistas e visando simplicidade de implementação de *hardware* ou DSP, pode-se alterar a duração da janela. Com janelas de 32 mseg (ou 16 mseg) é possível usar o algoritmo *butterfly* (radix-2) [16], restando investigar o impacto na eficiência.

Uma comparação rigorosa entre a complexidade e o compromisso com o desempenho do algoritmo de reconhecimento do locutor entre diferentes técnicas IAL não foi realizada. Porém o principal mérito desta nova abordagem é oferecer uma taxa de reconhecimento razoável, porém demandando uma complexidade computacional substancialmente inferior àquela requerida por outras técnicas consagradas (e.g., HMM, redes neurais, quantização vetorial etc.). Vale lembrar que as complexidades (por janela de 20 ms) exigidas pela FFT (N=160) e algoritmo de seleção do maior elemento de uma lista (Tabela 1) são desprezíveis para os comprimentos requeridos. A adaptação do método para uso de wavelets discretas [8], tornando-o mais atrativo, também se encontra em investigação. Outro aproveitamento possível deste algoritmo é nos casos em que a base de locutores é demasiadamente extensa. Este método rápido pode ser aplicado, selecionando um locutor provável, incluído em uma subclasse de locutores potenciais. Este é então eliminado da base original, repetindo o processo de forma a escolher um segundo locutor potencial. O procedimento é iterado até gerar um número pré-estabelecido de locutores potenciais (base reduzida). Esta aplicação prévia não requer taxas de acerto excessivamente altas, 90% é bastante razoável. Um método sofisticado (alto custo computacional e alta eficiência) é aplicado para identificar o locutor dentro desta base reduzida. Outra situação de potencial interesse para este método é no monitoramento em tempo real de telefonemas em prédios (empresas, repartições, etc.) que possuem centrais telefônicas. Com centenas de ligações simultâneas e diferentes ramais, como selecionar gravações (autorizadas) de conversações envolvendo indivíduo sob suspeição? Supõe-se disponível um trecho previamente gravado (e.g., primeiro contato de um seqüestrador, chantagista, corrupto, terrorista etc.) para constituir a informação de treinamento do locutor alvo. Neste caso, taxas de FA e FR aceitáveis podem ser maiores do que em aplicações comerciais típicas. Assim, situações em tempo real – nas quais há parca disponibilidade de recursos (como em sistemas embarcados) – esta técnica pode se tornar bastante atrativa.

AGRADECIMENTOS- Os autores agradecem a revisores anônimos por sugestões valiosas para aperfeiçoar a apresentação deste trabalho.

6 REFERÊNCIAS

[1] Oliveira, M.P.B., "*Verificação Automática de locutor, Dependente do Texto, Utilizando Sistemas Híbridos MLP/HMM*" Dissertação de Mestrado – Instituto Militar de Engenharia / IME - 2001.

[2] Campbell Jr, J.P., "Speaker Recognition: A Tutorial", *Proceedings of the IEEE*, September, vol.85, n 9. (1997).

[3] Atal, B.S. "Automatic Recognition of Speakers from Theirs Voices", *Proceedings of the IEEE*, April, vol 64, n 64, pp 460-475 (1976).

[4] Rosemberg, A.E. "Automatic Speaker Verification: A Review", *Proceedings of the IEEE*, April vol. 64, n 4, pp. 475-487 (1976).

[5] Dan, Z. Zheng, S. Sun S. and Dong, R. "Speaker Recognition based on LV-SVM" – *The 3rd International Conference on Innovative Computing Information and Control* (ICICIC'08), 2008.

[6] Wang, N. Ching, P.C. Zheng N.H. and Tan Lee – "Robust Speaker Recognition Using Both Vocal Source and Vocal Tract Features Estimated from Noisy Input Utterances", *IEEE International Symposium on Signal Processing and Information Technology*, 2007.

[7] Shao Y. and Wang D., "Robust Speaker Recognition Using Binary Time-Frequency Masks"- *IEEE International Conference on Acoustic, Speech and Signal Processing* 2006 (ICASSP 2006).

[8] De Oliveira, H.M., *Análise de sinais para Engenheiros – Uma abordagem via Wavelets*, Brasport, 2007.

[9] Diniz, S.S. "*Uso de Técnicas Neurais para o Reconhecimento de Comandos à Voz*". Dissertação de Mestrado, IME, Rio de Janeiro, 1997.

[10] Rabiner, L.R.; Schafer, R.W. *Digital processing of speech signals*. New Jersey: Prentice Hall, 1978.

[11] Silva, D.D.C, "*Desenvolvimento de um IP Core de Pré-Processamento Digital de Sinais de Voz para Aplicações em Sistemas Embutidos*", Dissertação de Mestrado, UFCG, Campina Grande, 2006.

[12] Petry, A., Zanuz, A. e Barone, D.A.C., "Reconhecimento Automático de Pessoas pela Voz usando técnicas de Processamento Digital de Sinais. *SEMAC*, *Semana de Computação da UNESP*, 2000.

[13] Rabiner, L.; Juang, B.H., *Fundamentals of Speech Recognition*. New Jersey: Prentice Hall, 1993. 507p.

[14] Paranaguá, E.D.S., "*Reconhecimento de Locutor Utilizando Modelos de Markov Escondidos Contínuos*", Dissertação de Mestrado, IME, Rio de Janeiro-RJ, 1997.

[15] Bezerra, M.R. "*Reconhecimento Automático de Locutor para Fins Forenses, Utilizando Técnicas de Redes Neurais*", Dissertação de Mestrado, IME, Rio de Janeiro, 2001.

[16] Oppenheim, A.V. & Schafer, R.W. *Digital-Time Signal Processing,* Prentice-Hall, Inc, Englewood Cliffs, New Jersey, 1989.

[17] Ferguson, T., *Mathematical Statistics: a Decision Theoretic Approach*, New York, Academic Press, 1967.

ANEXO

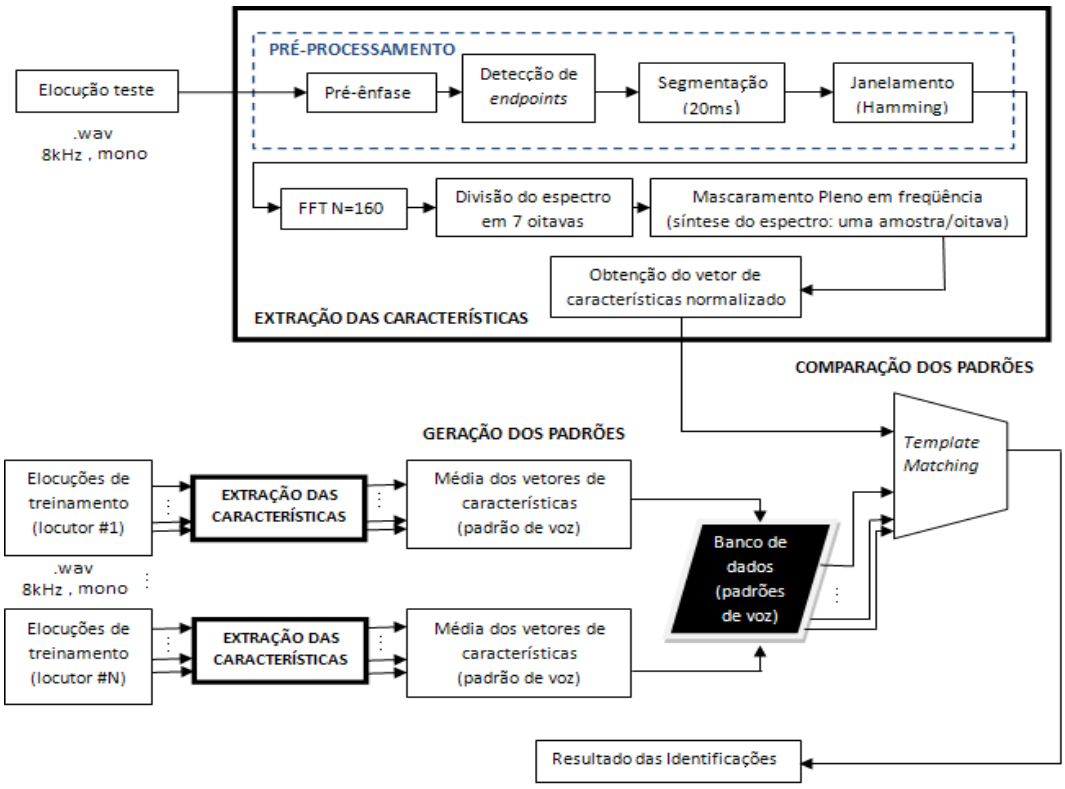


Figura 2 Diagrama de blocos de um sistema de reconhecimento de locutor com base no mascaramento de freqüências por oitava.

A Full Frequency Masking Vocoder for Legal Eavesdropping Conversation Recording

R. F. B. Sotero Filho, H. M. de Oliveira, R. Campello de Souza

Signal Processing Group, Federal University of Pernambuco - UFPE
E-mail: rsotero@hotmail.com.br, {hmo,ricardo}@ufpe.br

***Abstract**: This paper presents a new approach for a vocoder design based on full frequency masking by octaves in addition to a technique for spectral filling via beta probability distribution. Some psycho-acoustic characteristics of human hearing - inaudibility masking in frequency and phase - are used as a basis for the proposed algorithm. The results confirm that this technique may be useful to save bandwidth in applications requiring intelligibility. It is recommended for the legal eavesdropping of long voice conversations.*

The purpose of the voice compression is to obtain a concise representation of the signal, which allows efficient storage and transmission of voice data [1]. With proper processing, a voice signal can be analyzed and encoded at low data rates and then resynthesized. In many applications, the digital coding of voice is needed to introduce encryption algorithms (for security) or error correction techniques (to mitigate the noise of the transmission channel). Often, the available bandwidth for the transmission of digitized voice is a few kilohertz [2]. In such conditions of scarce bandwidth, it is necessary to adopt coding schemes that reduce the bit rate in such a way that information can be properly transmitted. However, these coding systems at low bit rate cannot reproduce the speech waveform in its original format. Instead, a set of parameters are extracted from the voice, transmitted and used to generate a new waveform at the receiver. This waveform may not necessarily recreate the original waveform in appearance, but it should be perceptually similar to it [3]. This type of encoder – called the vocoder (a contraction from voice encoder), a term also used broadly to refer to encoding analysis / synthesis in general, will use perceptually relevant features of the voice signal to represent it in a more efficient way, without compromising much on its quality [3]. The vocoder was first described by Homer Dudley at Bell Telephone Laboratory in 1939, and consisted of a voice synthesizer operated manually [4]. Generally speaking, the vocoders are based on the fact that the vocal tract changes slowly and its state and configuration may be represented by a set of parameters. Typically, these parameters are extracted from the spectrum of the voice signal and updated every 10-25 ms [5]. In general, given its low complexity in the process of generating the synthesized voice, the modeling, and the nature of simplifications carried out by vocoders, they introduce losses and/or distortions that ultimately make the voice quality below those obtained by waveform encoders [5]. Two properties of voice communication are heavily exploited by vocoders. The first is the limitation of the human auditory system [6]. This restriction makes the listeners hearing rather insensitive to various flaws in the process of voice reproduction. The second concerns the physiology of the voice generation process that places strong constraints on the type of signal that can occur, and this fact can be exploited to model some aspects of the production of the human voice [3,5]. The vocoder also find wide acceptance as an essential principle for handling audio files. For example, audio effects like time stretching or pitch transposition are easily achieved by a vocoder [7]. Since then, a series of modifications and improvements to this technology have been published [5]. In this article we present an innovative technique, which combines simplicity of implementation, low computational complexity, low bit rate and acceptable quality of generated voice files. In our approach, the stage of analysis of the voice signal is based on full frequency masking, recently published in [8] and explained in detail in Section III. In the resynthesis stage of the signal, we present a new approach based on spectral filling by a beta probability distribution.

The first stage of the proposed vocoder is a pre-signal processing. This is often required in speech processing, since the characteristics of voice signals have peculiarities that need to be worked with, previously. Because vocoders are designed for voice signals, which have most of their energy concentrated in a limited range of frequencies (typically between 300 Hz and less than 4 kHz), it is required to limit the bandwidth of the signals within this range, by a low-pass filter. Then a sampling rate that meets the Shannon sampling theorem condition must be taken. According to this theorem [9], there is no loss of information in the sampling process when a signal band limited to f_m Hz is sampled at a rate of at least $2f_m$ equally spaced samples per second. *Voice Segmentation and Windowing*-A signal is said to be stationary when its statistical features do not vary with time [9]. Since the voice signal is an stochastic process, and knowing that the vocal tract changes its shape very slowly in a continuous speech, many parts of the acoustic waveform can be assumed as stationary within a short duration range (typically between 10 and 40 ms). Segmentation is the partition of the speech signal into pieces (frames), selected by windows of duration perfectly defined. The size of these segments is chosen within the bounds of stationarity of the signal [10]. The use of windowing is a way of achieving increased spectral information from a sampled signal [11]. This "increase" of information is due to the minimization of the margins of transition in truncated waveforms and a better separation of the signal of small amplitude from a signal of high amplitude with frequencies very close to each other. Many different types of windows can be used. The Hamming window was chosen due to the fact that it presents interesting spectral characteristics and softness at the edges [12]. *Pre-emphasis*- The pre-emphasis aims to reduce a spectral inclination of approximately -6dB/octave, radiated from the lips during speech. This spectral distortion can be eliminated by ap-plying a filter response approximately +6 dB / octave, which causes a flattening of the spectrum [13]. The hearing is less sensitive to frequencies above 1 kHz of the spectrum; pre-emphasis amplifies this area of the spectrum, helping spectral analysis algorithms for modeling the perceptually aspects of the spectrum of voice [6,11]. Equation (1) describes the pre-emphasis performed on the signal that is obtained by differentiating the input.

$$y(n)= x(n)-a.x(n-1), \quad (1)$$

for $1 \leq n < M$, where M is the number of samples of $x(n)$, $y(n)$ is the emphasized signal and the constant "a" is normally set between 0.9 and 1. In this paper the adopted value was "a" equals to 0.95 [13]. The algorithms developed for implementation of this vocoder were written in MATLAB^{TM} platform, owing to the fact that it is a widespread language in the academic world and it is easy to implement. In the following, details of the approach are described. As in most efficient speech coding systems, vocoders may exploit certain properties of the human auditory system, taking advantage of them to reduce the bit rate. The technique proposed in this article for implementation of the vocoder is founded on two important characteristics: the masking in frequency and the insensitivity to phase. The function of the stage of analysis is, a priori, to identify the frequency masking in the spectrum of the signal (obtained by an FFT of blocklength 160), partitioned into octave bands, discard signals that "would not be audible," due to the phenomenon of masking in frequency [14], and totally disregard the signal phase.
Psycho-Acoustics of the Human Auditory System- Because it is of great importance for the understanding of the proposed method, a few characteristics of human auditory system are briefly discussed [6,14].

• Frequency Masking: Masking in frequency or "reduced audibility of a sound due to the presence of another" is one of the main psycho-acoustic characteristics of human hearing. The auditory masking (which may be in frequency or in time) occurs when a sound, that could be heard, is often masked by another, more intense, which is in a nearby frequency. In general, the presence of a tone cannot be detected if the power of the noise is more than a few dB above this tone. Due to the effect of masking, the human auditory system is not sensitive to detailed structure of the spectrum of a sound within this band [3,5].

• Insensitivity to the phase: The human ear has little sensitivity to the phase of signals. The process can be explained by examining how sound propagates in an environment.

Any sound that propagates reaches our ears through various obstacles and travels distinct paths. Part of the sound gets lagged, but this difference is hardly felt by the ear [15]. The information in the human voice is mostly concentrated in "bands of frequencies". Based on this fact, the proposed vocoder discards the phase characteristics of the spectrum.

Simplification of the spectrum via the frequency masking- Equipped with the pre-processed signals, we can start the stage of signal analysis, which is described in the sequel. For each voice segment of the file, an FFT of blocklength 160 (number of samples contained in a frame of 20 ms of voice) is applied, thus obtaining the spectral representation of each voice frame. Only the magnitude of the spectrum is considered. After that, the spectrum is segmented into regions of influence (octaves). The range of frequencies between 32 and 64 Hz is removed from the analysis. The first pertinent octave corresponds to the frequency range 64 Hz-128 Hz, the second covering the band 128 Hz-512 Hz, and so on. The sixth (last octave band) matches the range of 2048 Hz-4000 Hz (remarking that from here the spectrum produced by the FFT begins to repeat). Since the sampling rate is 8 kHz, each spectral sample corresponds to a multiple of 50 Hz, and the first sample represents the DC component of each frame of speech. Because this sample has no information, it is promptly disregarded from the analysis. Since the spectral lines have a step of 50 Hz, the first octave (from 64 Hz to 128 Hz) is represented by the spectral sample of 100 Hz, the second octave (from 128 Hz to 256 Hz) by samples at 150 Hz, 200 Hz and 250 Hz, with the remaining octaves following a similar reasoning. After this preliminary procedure, we search at each octave, in all relevant sub-bands of the voice signal, for the DFT component of greatest magnitude, i.e., that one that (potentially) can mask the others. There are 80 spectral lines (dc is not shown). This component is taken as the sole representative tone in each octave (as an option of reducing the complexity). The other spectral lines are discarded, assuming a zero spectral value. A total of 79 frequencies coming from the estimation of the DFT with N=160 is then reduced to only 4 survivors (holding less than 5% of the spectral components). Therefore, each frame is now represented in the frequency domain by 4 pure (masking) tones. This technique is called full frequency masking [8]. These simplified frames are encoded and used by a synthesizer to retrieve the voice signal. Now a signal synthesis is described on the basis of a spectral filling via a distribution of probability. The beta distribution is a continuous probability distribution defined over the interval $0 \leq x \leq 1$, characterized by a pair of parameters α and β, according to Equation [16]:

$$P(x)=1/\mathrm{B}(\alpha,\beta)\ x^{(\alpha-1)}\ (1-x)^{(\beta-1)}, \qquad 1<\alpha,\beta<+\infty, \tag{2}$$

whose normalized factor is $\mathrm{B}(\alpha,\beta)=(\Gamma(\alpha)\Gamma(\beta))/(\Gamma(\alpha+\beta))$, where $\Gamma(.)$ is the generalized Euler factorial function and B(.,.) is the Beta function. The point where the maximum of the density is achieved is the mode and can be computed by the following equation [16]:

$$mode=(\alpha-1)/(\alpha+\beta-2). \tag{3}$$

Octave (Hz)	# spectral samples/octave
32-64	1
64-128	1
128-256	3
256-512	5
512-1024	10
1024-2048	20
2048-4096	39

Table 1. Number of Spectral Lines per Octave Estimated by a DFT of Length N=160 with a Sample Rate 8 kHz.

The purpose of the synthesis stage is to retrieve the voice signal from data provided by the parsing stage. As mentioned, the full frequency masking was adopted to simplify the spectrum of each frame of voice. Such a simplification results in a very vague and spaced sample configuration in the spectrum. To improve this representation, the synthesizer can use the

spectral filling technique via beta distribution, so as to smooth the abrupt transition between adjacent samples in octaves, assigning interpolated values to lines with zero magnitude, thus filling up the spectrum completely. Each octave has its own distribution and these are updated with each new frame. The peak of each of these distributions is equal to the survivor spectral sample after the full masking simplification. In what follows, the methodology of spectral filling, via beta distribution, is described. Since the beta distribution is defined over the interval [0,1], see Fig.1, it is necessary to scale and translate the original expression of the distribution, so that their range encompass the transition from one octave to another. Moreover, the value of the mode should assume the same value of the survivor spectral sample within the octave. Based on the original expression of the beta distribution, given by Eq. (2), there is a suitable scaling of the curve so that the upper limit is equivalent to the difference between the normalized cutoff frequency exceeding (f_M) and lower (f_m) of each octave, i.e., f_M - f_m. The cutoff frequencies need to be normalized, since the limiting frequency of octaves (64-128 Hz, 128 – 256 Hz, etc.) are not multiples of 50 Hz, which is the value of the spectrum step while sampling at 8 kHz. Later, the curve must be translated so that the lower and upper limits become f_m and f_M, respectively. By making the fitting, it is also necessary to adjust the value of the mode, which becomes

$$new_{mode} = (\alpha\text{-}1)/(\alpha+\beta\text{-}2)\ (f_M - f_m) + f_m. \quad (4)$$

From this expression and after some mathematical manipulations, we find a relation between α and β, which is useful in representing the adjusted expression of the distribution:

$$\beta\text{-}1 = (\alpha\text{-}1).Q, \quad (5)$$

where:

$$Q := (f_M - f_c)/(f_c - f_m). \quad (6)$$

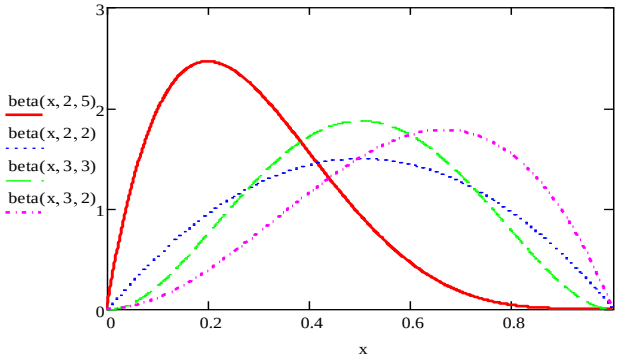


Figure 1. Envelope shape of the survivor tone is shown for a few parameters α and β.

The final expression, one that is used to fulfill the spectral algorithm each frame, is given by:

$$P(x) = 1/(f_M - f_m)^{(\alpha+\beta-2)}\ (x - f_m)^{(\alpha-1)}\ (f_M - x)^{(\beta-1)}. \quad (7)$$

The value of α in Eq. (7) represents a parameter of expansion/compression of the interpolation curve. The higher its value, the narrower it becomes. The values of α were octave-dependent. Fig. 2 shows the magnitude of the spectrum of a frame of a file (test voice file), (a) before simplifying by masking, (d) after simplifying and (c) after the fulfilling via beta distribution. A few audio files generated by this vocoder are available at the URL http://www2.ee.ufpe.br/codec/vocoder.html Given the symmetry of the DFT, it is also necessary to fulfill the half mirror portion of the spectrum for proper signal restoration. Otherwise a signal in time domain, complex in nature, will be incorrectly generated. As one of the last stages of reconstruction of the voice signal, there is the transformation from the frequency domain to the time domain of all voice frames. Such a transformation is achieved through the inverse fast Fourier transform (IFFT) of the same blocklength of a frame. In doing so, the frames are glued one by one, resetting the pre-emphasized signal. An inverse pre-emphasis filter is used to de-emphasize the signal, thus finalizing the process of recovering the voice signal. For each frame, spectral samples survivors and the positions of each are then quantized and encoded, and saved in a binary format (.voz) and used later by a synthesizer.

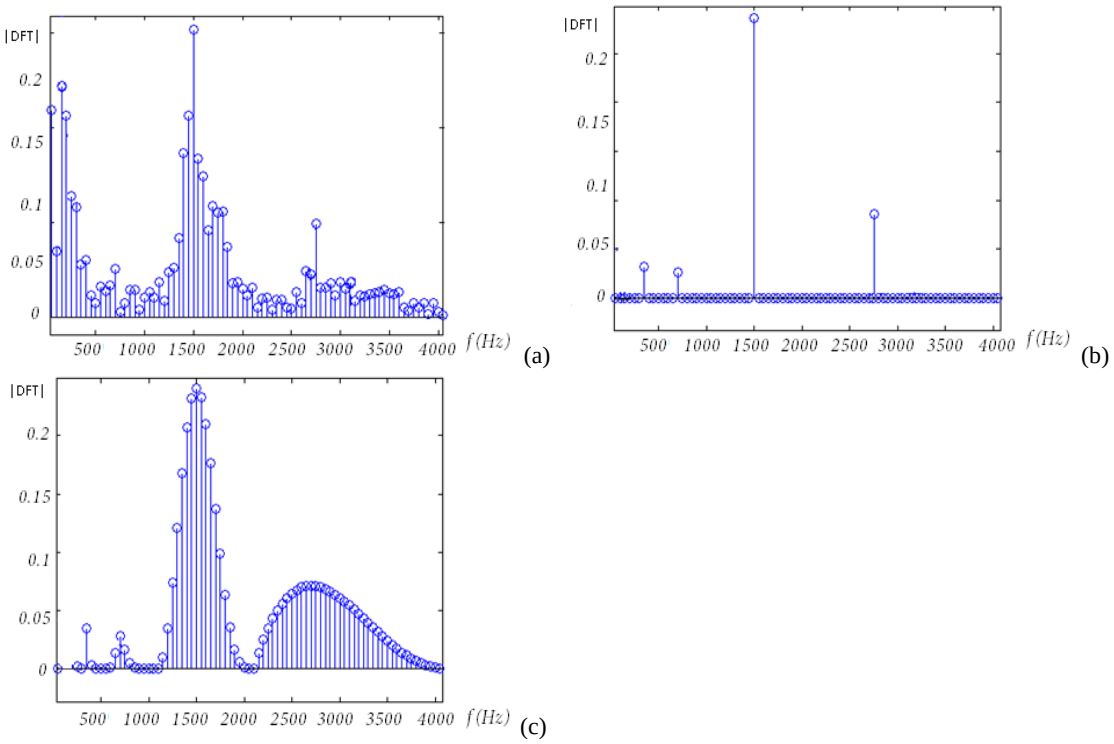


Figure 2. Steps of the procedure of analysis/synthesis of a frame of tested voice signal. The spectrum of a voice frame computed by the FFT is shown: a) Original spectrum, b) Simplified spectrum using full masking, c) Spectrum fulfilled by the beta distribution.

The quantization and coding procedures (allocation of bits per frame) are shown in the sequel. The most common method was used in the quantization of frames: the uniform quantization. A number of levels coincident with a power of 2 was adopted to simplify the binary encoding. The maximum excursion of the signal (greater magnitude of the full spectrum of the voice signal) was thus divided into 256 intervals of equal length, each represented by one byte. Since there are no negative samples to be quantized (the magnitude of the spectrum does not assume negative values), the quantizer cannot be bipolar.

Relevant octave	#possible survivor components	Bits A+P
#1 (256-512 Hz)	5	8 + 3
#2 (512-1024 Hz)	10	8 + 4
#3 (1024-2048 Hz)	20	8 + 5
#4 (2048-4096 Hz)	39	8 + 6

Table 2. Bit allocation in a voice frame (20 ms). The required number of bits is expressed as A + P, where A is the number of bits for spectral line amplitude and P the number of bits to express the relative position within the OCTAVE.

A MATLAB routine is specifically designed for this purpose. The quantization of the positions was not necessary, since they are integer-valued. In order to reduce the number of bits needed for encoding voice frames, the bit allocation algorithm took into consideration the bandwidth of each octave. A lower octave reduces by half the bandwidth and therefore fewer bits are needed for proper co-ing of positions in which the spectral masking occurred. Positions in successive octaves (spectrum towards high frequencies) need an extra bit for its correct representation. For example, a tone masking which occurs in the first octave (256-512 Hz) has 5 possible occurrences (position 7 to position 11 of DFT), thereby requiring a 3-bit codeword. In the next octave, (512 - 1024 Hz), the maximum position that the tone masking may occur is 21, which can be encoded by a 4-bit codeword. In the subsequent two octaves, the peak may be at 41th

position (5-bit codeword) and 80th (6-bit codeword), respectively. For the maximum values of the spectral masking samples, one byte is reserved for their representation. The number of bits allocated to each of these parameters is shown in Table 2. As mentioned, the phase information of the spectrum is disregarded. It is seen that each voice frame needs only 50 bits (18 for identifying positions and 32 for identifying masking tones), leading to a rate of 50 bits/20 ms=2.5 kbps. *The binary format .voz* - The bit allocation in each frame, summarized in Table II, suggests the concatenation of encoded frames. The representation of a voice frame in this format (extension .voz) is shown in Fig.3. The 50 bits are distributed into four sub blocks (one for each octave), indicating the value of the spectral sample followed by its respective position in the spectrum. The voice files registered in the .wav format are all converted to this binary format, by a Matlab routine. In the decoder, the reconstruction algorithm of the synthesized spectrum, can recover the voice signal by converting it back into the .wav format.

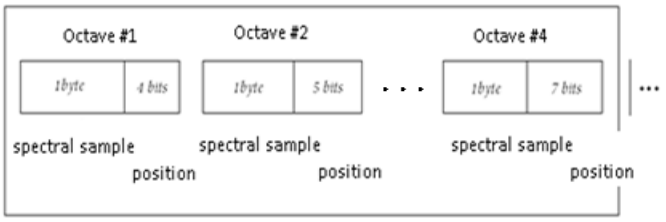


Figure 3. Frame of files in the format .voz (20 ms).

Simulation results usually focus on intelligibility and voice quality versus bit rate [17]. Fifty-eight subjects, of whom eight were trained, were accessed for this study. Voice quality is estimated using the "Mean Opinion Score (MOS)" and "degradation Mean Opinion Score (DMOS)" tests. During the tests, "MOS"-listeners were asked to rate voice quality of the output files considering an absolute scale 1-5, with 1 meaning very poor quality and 5 being excellent. The main obstacle for the MOS testing was that ordinary people were not familiar with low bit rate vocoders and got confused between a sound disharmonies, stuffy, with tinnitus, and the nasal quality of speech and noise added after encoding. To overcome this limitation, DMOS tests were conducted. In this test, listeners were asked to rate the quality of sentences encoded and spread over time on the output of the vocoder MELP pattern [15]. Preliminary tests were conducted and voice signals tested using four different techniques of synthesis.

Evaluated techniques.

1. Synthesized signals with no spectral filling.
2. Vocoder signals reconstructed via beta spectral filling technique.
3. Synthesized voice signals combining 1 and 2 techniques (linear combination).
4. Voice signals from item 2, but with an extra Hamming windowing.

Results are summarized in Table III. They were reasonable, given the low bit rate (2.5 kbits/s) and low implementation complexity of the vocoder. Indeed, the comparison is "unfair" to those coders, since the MOS values obtained for them were much more insightful and performed with a wide range of listeners, or even using objective methods such as PESQ [17]. It can be observed from Table 3 that noise is still a factor that impairs such an assessment, reflecting a lower MOS score for noisy signals (produced by the technique of spectral filling).

Vocoder technique	MOS score
1	3.0
2	2.5
3	2.8
4	3.0

Table 3. MOS scores for the voice signals synthesized by four different techniques.

We introduced a new vocoder that can represent a voice signal using fewer samples of the spectrum. Our initial results suggest that this approach has the potential to transmit voice, with acceptable quality, at a rate of a few kbits/s. A new technique of spectral filling was also presented, which is based on the beta distribution of probability. Surprisingly, this was not helpful in improving the voice quality, although it improved the naturalness of the speech generated by this vocoder. This vocoder can be useful for the transmission of maintenance voice channels in large plants. It was successfully applied in a recent speaker recognition system. In particular, it is offered as a technique for monitoring long voice conversation stemming from authorized eavesdropping.

References

[1] Schroeder, M.R., A Brief History of Synthetic Speech, Speech Comm., vol.13, pp.231-237, (1993).

[2] Pope, S.P., Solberg, B., Brodersen, R.W, A Single-Chip Linear-Predictive-Coding Vocoder, IEEE J. of Solid-State Circuits, vol. SC-22, (1987).

[3] Holmes, J., Holmes, W. Speech Synthesis and Recognition, Taylor & Francis, 2001.

[4] Schroeder, M.R., Homer Dudley: A tribute, Signal Processing, vol.3, pp.187-188, (1981).

[5] Spanias, A., Speech Coding: A tutorial Review, Proc. of IEEE, vol. 82, pp.1541-1582, (1994).

[6] Greenwood, D., Auditory Masking and the Critical Band, J. Acoust. Soc. Am., vol. 33, pp. 484-502, (1961).

[7] Zoelzer, U. Digital Audio Effects', Wiley & Sons, pp.201-298, 2002.

[8] Sotero Filho, R.F.B, de Oliveira, H., Reconhecimento de Locutor Baseado no Mascaramento Pleno em Frequência por Oitava, Audio Engineering Congress, AES2009, São Paulo,2009.

[9] Lathi, B.P. Modern Digital and Analog Communication Systems. Oxford Univ. Press, NY, 1998.

[10] Rabiner, L.R.; Schafer, R.W., Digital Processing of Speech Signals. Prentice Hall, NJ, 1978.

[11] Turk, O., Arsan, L.M., Robust Processing Techniques for Voice Conversation, Computer Speech & Language, vol. 20, pp.441-467, (2006).

[12] Taubin, G., Zhang, T., Golub, G., Optimal Surface Smoothing as Filter Design, Lecture Notes on Computer Science, vol. 1064, pp. 283-292, (1996).

[13] Schnell, K., Lacroix, A., Time-varying pre-emphasis and inverse filtering of Speech, Proc. Interspeech, Antwerp, 2007.

[14] Wegel, R.L., Lane, C.E., Auditory masking of one pure tone by another and its probable relation to the dynamics of the inner ear, Physical Review, vol. 23, pp. 266-285, (1924).

[15] Smith, S.W., Digital Signal Processing – A Practical Guide for Engineers and Scientists, Newnes, 2003.

[16] de Oliveira, H.M., Araújo G.A.A., Compactly Supported One-cyclic Wavelets Derived from Beta Distributions, Journal of Communication and Information Systems, vol. 20, pp.27-33, (2005).

[17] Kreiman, J., Gerrat, B.R., Validity of rating scale measures of voice quality, J. Acoust. Soc. Am., vol. 104, pp.1598-1608, (1998).

Anexos

ANEXO A – CÓDIGOS FONTE (*VOCODER*)

Requerido MATLAB® 2013 ou versões mais recentes.

VocCod.m

```
function varargout = VocCod(varargin)
%VOCCOD          Codificador wav -> voz baseado no Mascaramento Pleno em
%                Frequencia por Oitava (MPFO).
%                Reune simplificidade de implementacao e baixa taxa de bits
%                (2,7 kbits/s). Aceita como entrada arquivos no formato wav
%                e produz arquivo binario,  codificado no formato voz, para
%                posterior decodificacao no VocDec.
%
%Autores:        Roberto F. B. Sotero Filho e Helio Magalhaes de Oliveira
%E-mail:         rsotero@hotmail.com e hmo@ufpe.br
%Universidade:   Universidade Federal de Pernambuco
%Data:           23/05/2017

% Codigo de inicializacao do programa - nao editavel
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @VocCod_OpeningFcn, ...
                   'gui_OutputFcn',  @VocCod_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% Chamada dos logos na tela.
function ufpe_CreateFcn(~, ~, ~)
A = imread('ufpectgdes_80.png');
imshow(A);
% ------------ Rotinas e funcoes internas do programa --------------------
```

```
function VocCod_OpeningFcn(hObject, ~, handles, varargin)
handles.output = hObject;
guidata(hObject, handles);

function varargout = VocCod_OutputFcn(~, ~, handles)
varargout{1} = handles.output;

function edit1_Callback(~, ~, ~)
function edit1_CreateFcn(hObject, ~, ~)
if ispc && isequal(get(hObject,'BackgroundColor'), ...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit2_Callback(~, ~, ~)
 function edit2_CreateFcn(hObject, ~, ~)
if ispc && isequal(get(hObject,'BackgroundColor'),...
         get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit4_Callback(~, ~, ~)
function edit4_CreateFcn(hObject, ~, ~)
if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
% --- Acoes executadas ao se pressionar o botao "selecionar" ------------.
function togglebutton2_Callback(~, ~, handles)
[nome,diretorio] = uigetfile('*.wav',...
'Selecione o arquivo .wav para processamento'); % Seleciona arquivo wav.
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')     % mensagem de cancelar.
else
    arquivowav = strcat(diretorio,nome);    % leitura do arquivo wav.
    set(handles.edit4,'string',arquivowav); % escreve arquivo no diretorio.
end

% --------------------- ROTINAS PRINCIPAIS DO PROGRAMA -------------------
% --- Acoes executadas ao se pressionar o botao "processar" -------------
function pushbutton1_Callback(hObject, ~, handles)
guidata(hObject, handles)
arquivowav = get(handles.edit4,'string'); %leitura do arquivo wav (caminho)
[y_in, fs_in] = audioread(arquivowav); % leitura atributos do arquivo wav.
fs = 8000;                             % tx amostragem utilizada.
y8000=resample(y_in,fs,fs_in);         % conversao da tx de amostragem.
                                       % para 8 kHz.
```

```
N = length(y8000);                          % comprimento dos sinais de voz.
c = 160;                                    % numero de amostras em 20ms.
n = round(N/c);
y8000 = [y8000;0*ones(n*c-N,1)];            % preenche amostras arquivo de voz
                                            % com zeros caso seu tamanho n seja
                                            % multiplo de 160.
HAM = hamming(c);                           % cria janela de Hamming de c = 160.

% Bloco de montagem espectro de frequencia simplificado (1 amostra/oitava)|
for i=0:n-1
  spectrum_bloco(c*i+1:c*i+c)= ...
  abs(fft(HAM(1:c).*y8000(c*i+1:c*i+c)));   % cria bloco de FFTs janeladas
                                            % com apenas as amplitudes.
end

for k =0:n-1;
  spectrum_bloco(c*k+1:c*k+6)=0;
  spectrum_bloco(c*k+156:c*k+160)=0;        %zera amostras n consideradas.
  spectrum_bloco(c*k+81)=0;

%-------------------------------------------------------------------------
%                       Oitava 1 (256 - 512 Hz)
%-------------------------------------------------------------------------
  int1 = c*k+7:c*k+11;                          % intervalo da oitava 1.
  y(int1)= spectrum_bloco(int1);                % forma bloco de amostras.
  spectrum_bloco(find(y ~= max(y)& y~=0))= 0; % zera amplitudes n maximas.
  valormaximo1 = spectrum_bloco(find(y == ... % acha amplitude maxima.
                                  max(y)));
  posicao1 = find(y==max(y));                   % posicao da amplitude max.
  clear y;
%-------------------------------------------------------------------------
%                       Oitava 2 (512 - 1024 Hz)
%-------------------------------------------------------------------------
 int2 = c*k+12:c*k+21;                          % intervalo da oitava 2.
 z(int2)=spectrum_bloco(int2);                  % forma bloco de amostras.
 spectrum_bloco(find(z ~= max(z)& z~=0))=0;     % zera amplitudes n maximas.
 valormaximo2 = spectrum_bloco(find(z == ...    % acha amplitude maxima.
                                max(z)));
 posicao2 = find(z==max(z));                    % posicao da amplitude max.
 clear z;
%-------------------------------------------------------------------------
%                       Oitava 3 (1024 - 2048 Hz)
%-------------------------------------------------------------------------
 int3 = c*k+22:c*k+41;                          % intervalo da oitava 3.
 h(int3)=spectrum_bloco(int3);                  % forma bloco de amostras.
 spectrum_bloco(find(h ~= max(h)& h~=0))=0;     % zera amplitudes n maximas.
 valormaximo3 = spectrum_bloco(find(h ==...     % acha amplitude maxima.
```

```
                                    max(h)));
 posicao3 = find(h==max(h));                    % posicao da amplitude max.
 clear h;
%-------------------------------------------------------------------------
%                       Oitava 4 (2048 - 4096 Hz)
%-------------------------------------------------------------------------
 int4 = c*k+42:c*k+80;                          % intervalo da oitava 4.
 j(int4)=spectrum_bloco(int4);                  % forma bloco de amostras.
 spectrum_bloco(find(j ~= max(j)&j~=0))=0;      % zera amplitudes n maximas.
 valormaximo4 = spectrum_bloco(find(j ==...     % acha amplitude maxima.
                                    max(j)));
 posicao4 = find(j==max(j));                    % posicao da amplitude max.
 clear j;
 spectrum_bloco(c*k+1:c*k+c) =...              % montagem do espectro completo
                                  [spectrum_bloco(c*k+1:c*k+80)...
                                   0 spectrum_bloco(c*k+80:-1:2+c*k)];

                                             %conversao p/binario das posicoes
  seq_posi_bin(22*k+1:22*k+22) = [dec2binvec(mod(posicao1,c),4)...
                                 dec2binvec(mod(posicao2,c),5)...
                                  dec2binvec(mod(posicao3,c),6)...
                                  dec2binvec(mod(posicao4,c),7)];
end
%-------------------------------------------------------------------------|
% Bloco de quantizacao (quantizacao uniforme de 256 niveis) das amplitudes|
% das amostras dominantes do espectro                                     |
%-------------------------------------------------------------------------|
ind_max_raia = find(spectrum_bloco==...
                    max(spectrum_bloco));     %indice da amostra maxima
maxraia = spectrum_bloco(ind_max_raia);       %valor da amostra maxima
valormaxraia = maxraia(1);
Q = valormaxraia/257;                         %passo de quantizacao
cont1=0;

for cont = 1:length(spectrum_bloco)
    q=0;
    while spectrum_bloco(cont) >= q*Q && spectrum_bloco(cont) ~= 0
           q = q+1;
    end

           if q>255
               q=255;
           end

           if q > 0
               cont1 = cont1+1;
               seq_bin_total(8*(cont1-1)+1:8*(cont1-1)+8)= dec2binvec(q,8);
```

```
            end
end

for i=0:n-1
                                    %montagem do bloco de amplitudes em binario
    seq_bin(32*i+1:32*i+32) = seq_bin_total(64*i+1:64*i+32);

end

%---------Bloco de formacao do arquivo binario (.voz) de saida -----------
for i=1:n
                                    %montagem dos blocos do formato .voz
 saidaBinaria(54*(i-1)+1:54*(i-1)+54)=...
                                       [seq_bin((32*(i-1)+1:32*(i-1)+8))...
                                       seq_posi_bin(22*(i-1)+1:22*(i-1)+4) ...
                                       seq_bin((32*(i-1)+9:32*(i-1)+16))...
                                       seq_posi_bin(22*(i-1)+5:22*(i-1)+9) ...
                                       seq_bin((32*(i-1)+17:32*(i-1)+24))...
                                       seq_posi_bin(22*(i-1)+10:22*(i-1)+15)...
                                       seq_bin((32*(i-1)+25:32*(i-1)+32))...
                                       seq_posi_bin(22*(i-1)+16:22*(i-1)+22)];

end

                                       % abre tela para salvar arquivo de saida
[nome_out,diretorio_out] = uiputfile('*.voz','Salvar arquivo .voz como');
if isequal(nome_out,0) || isequal(diretorio_out,0)
     disp('Usuario pressionou cancelar')
else
arquivo_out = strcat(diretorio_out,nome_out);
fid = fopen(arquivo_out,'w');
fwrite(fid,saidaBinaria');        %escreve os dados para o arquivo binario
aviso_sucesso                     %habilita tela de sucesso no processamento
end
```

VocDec.m

```
function varargout = VocDec(varargin)
%VOCDEC          Decodificador voz -> waz baseado no Mascaramento Pleno em
%                Frequencia por Oitava (MPFO).
%                Aceita como entrada arquivos codificados no formato voz e
%                produz arquivos wav, atraves de 4 tipos de recomposicao do
%                sinal: (i) utilizando apenas o %MPFO, (ii) MPFO acrescido
%                da tecnica de preenchimento espectral, iii) utilizando uma
%                composicao  dos  sinais  em  (i) e (ii), e (iv) utilizando
%                sinais de (ii) com um janelamento de Hamming extra.
%
%Autores:        Roberto F. B. Sotero Filho e Helio Magalhaes de Oliveira
%E-mail:         rsotero@hotmail.com e hmo@ufpe.br
%Universidade:   Universidade Federal de Pernambuco
%Data:           25/05/2017

% Codigo de inicializacao do programa - nao editavel
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @VocDec_OpeningFcn, ...
                   'gui_OutputFcn',  @VocDec_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% Chamada dos logos na tela.
function ufpe_CreateFcn(hObject, eventdata, handles)
A = imread('ufpectgdes.png');
imshow(A);
%------------ Rotinas e funcoes internas do programa --------------------
function VocDec_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;
guidata(hObject, handles);

function varargout = VocDec_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;
```

```
function edit1_Callback(hObject, eventdata, handles)
function edit1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit2_Callback(hObject, eventdata, handles)
function edit2_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit4_Callback(hObject, eventdata, handles)
function edit4_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
% --- Acoes executadas ao se pressionar o botao "selecionar" ------------
function togglebutton2_Callback(hObject, eventdata, handles)
guidata(hObject, handles)
[nome,diretorio] = uigetfile('*.voz',...
'Selecione o arquivo .voz para processamento'); % Seleciona arquivo voz.
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')      % mensagem de cancelar.
else
    arquivovoz= strcat(diretorio,nome);      % leitura do arquivo voz
    set(handles.edit4,'string',arquivovoz);  % escreve arquivo no diretorio
end
% ------------------------------------------------------------------------
% -------------------- ROTINAS PRINCIPAIS DO PROGRAMA --------------------
% --- Acoes executadas ao se pressionar o botao "processar" -------------
function pushbutton1_Callback(hObject, eventdata, handles)
guidata(hObject, handles)
arquivovoz = get(handles.edit4,'string');
valor4 = get(handles.radiobutton4,'Value');
valor5 = get(handles.radiobutton5,'Value');
valor6 = get(handles.radiobutton6,'Value');
valor7 = get(handles.radiobutton7,'Value');

fs = 8000;                                  % tx amostragem utilizada.
c = 160;                                    % numero de amostras em 20ms.
fid = fopen(arquivovoz);                    % ler identificador arquivo voz
arq_interm_bin = fread(fid)';               % ler o arquivo binario voz
```

```
% CONVERSAO DO SINAL DE VOZ DE BINARIO PARA DECIMAL
for i=1:length(arq_interm_bin)/54
      arq_interm(8*(i-1)+1:8*(i-1)+8) =...
          [binvec2dec(arq_interm_bin((54*(i-1)+1:54*(i-1)+8)))...
          binvec2dec(arq_interm_bin(54*(i-1)+9:54*(i-1)+12))+160*(i-1)...
          binvec2dec(arq_interm_bin((54*(i-1)+13:54*(i-1)+20)))...
          binvec2dec(arq_interm_bin(54*(i-1)+21:54*(i-1)+25))+160*(i-1),...
          binvec2dec(arq_interm_bin((54*(i-1)+26:54*(i-1)+33))) ...
          binvec2dec(arq_interm_bin(54*(i-1)+34:54*(i-1)+39))+160*(i-1)...
          binvec2dec(arq_interm_bin((54*(i-1)+40:54*(i-1)+47))) ...
          binvec2dec(arq_interm_bin(54*(i-1)+48:54*(i-1)+54))+160*(i-1)];
end
n = length(arq_interm)/8;

%------Bloco de Recomposicao do espectro simplificado pelo MPFO ---------|
for k =0:n-1;
    spectrum_bloco(c*k+1:c*k+6)=0;
    spectrum_bloco(c*k+156:c*k+160)=0;      %zera amostras n consideradas.
    spectrum_bloco(c*k+81)=0;
%-------------------------------------------------------------------------
%                       Oitava 1 (256 - 512 Hz)
%-------------------------------------------------------------------------
      spectrum_bloco(arq_interm(8*k+1+1))=arq_interm(8*k+1);
%-------------------------------------------------------------------------
%                       Oitava 2 (512 - 1024 Hz)
%-------------------------------------------------------------------------
       spectrum_bloco(arq_interm(8*k+3+1))=arq_interm(8*k+2+1);
%-------------------------------------------------------------------------
%                       Oitava 3 (1024 - 2048 Hz)
%-------------------------------------------------------------------------
       spectrum_bloco(arq_interm(8*k+5+1))=arq_interm(8*k+4+1);
%-------------------------------------------------------------------------
%                       Oitava 4 (2048 - 4096 Hz)
%-------------------------------------------------------------------------
      spectrum_bloco(arq_interm(8*k+7+1))=arq_interm(8*k+6+1);
%-------------------------------------------------------------------------

%CONSTRUCAO DO ESPECTRO SIMPLIFICADO COMPLETO
        spectrum_bloco(c*k+1:c*k+c) = [spectrum_bloco(c*k+1:c*k+80)...
                                       0 spectrum_bloco(c*k+80:-1:2+c*k)];
%OBTENCAO DO SINAL DE VOZ NO TEMPO ATRAVES DA IFFT DO ESPECTRO SIMPLIFICADO
         arq(c*k+1:c*k+c) = ifft(spectrum_bloco(c*k+1:c*k+c));
end
% DEFINICAO DO ARQUIVO DE SAIDA
     saida = arq';

%---------------------TIPOS DE RECOMPOSICAO DO SINAL--------------------
```

```
% (i) MPFO apenas
if  valor4 ==1
    [nome_out,diretorio_out] = uiputfile('*.wav',...
        'Salvar arquivo .wav como');
    if isequal(nome_out,0) || isequal(diretorio_out,0)
       disp('Usuario pressionou cancelar')
    else
       arquivo_out = strcat(diretorio_out,nome_out);
       audiowrite(arquivo_out,0.35*saida,fs)
       aviso_sucesso                              % habilita tela de sucesso
    end
end
% (ii) MPFO + Preenchimento espectral
if  valor5 ==1
    out = SpecStuffing(saida);             % chama rotina de preenchimento
    [nome_out,diretorio_out] = uiputfile('*.wav',...
                                         'Salvar arquivo .wav como');
    if isequal(nome_out,0) || isequal(diretorio_out,0)
       disp('Usuario pressionou cancelar')
    else
       arquivo_out = strcat(diretorio_out,nome_out);
       audiowrite(arquivo_out,0.35*out,fs)
       aviso_sucesso                              % habilita tela de sucesso
    end
end
% (iii) Composicao de (i) com (ii)
if  valor6 ==1
    out = SpecStuffing(saida);             % chama rotina de preenchimento
    out = out(1:length(saida));
    out_new = (saida+out)/2;                % calcula media de (i) e (ii)
    [nome_out,diretorio_out] = uiputfile('*.wav',...
                                         'Salvar arquivo .wav como');
    if isequal(nome_out,0) || isequal(diretorio_out,0)
       disp('Usuario pressionou cancelar')
    else
       arquivo_out = strcat(diretorio_out,nome_out);
       audiowrite(arquivo_out,0.35*out_new,fs)
       aviso_sucesso                           % habilita tela de sucesso
    end
end
% (iv) MPFO + Preenchimento espectral + Janelamento extra
if  valor7 ==1
    out = SpecStuffWindow(saida); % chama rotina de preenchimento
                                              % e janelamento extra
    [nome_out,diretorio_out] = uiputfile('*.wav',...
                                         'Salvar arquivo .wav como');
```

```
    if isequal(nome_out,0) || isequal(diretorio_out,0)
       disp('Usuario pressionou cancelar')
    else
       arquivo_out = strcat(diretorio_out,nome_out);
       audiowrite(arquivo_out,0.35*out,fs)
       aviso_sucesso                              % habilita tela de sucesso
    end
end
```

SpecStuffing.m

```
function [out] = SpecStuffing(in)
%SPECSTUFFING    Tecnica de Preenchimento Espectral via Distribuicao Beta
%                de  Probabilidade.
%
%Autores:        Roberto F. B. Sotero Filho e Helio Magalhaes de Oliveira
%E-mail:         rsotero@hotmail.com e hmo@ufpe.br
%Universidade:   Universidade Federal de Pernambuco
%Data:           10/2008

fs = 8000;                                  % frequencia amostragem utilizada.
N = length(in);                             % comprimento arquivo de entrada
c = fs*20e-3;                               % numero de amostras em 20 ms.
%calculo do numero de blocos no arquivo de voz
if rem(N,c) ~= 0
    n = round(N/c + 0.5);
    in = [in; 0*ones(n*c-N,1)];
else
    n = N/c;
end
HAM = hamming(c);                          %criando janela de Hamming
% --------   Frequencias de corte normalizadas de cada oitava   -----------
% Oitava 0 fm0=64/50 +1    fM0=2*64/50 +1
fm0=2.28; fM0=3.56;
% Oitava 1 fm1=128/50 +1    fM1=2*128/50 +1
fm1=3.56; fM1=6.12;
% Oitava 2 fm2=256/50 +1    fM2=2*256/50 +1
fm2=6.12; fM2=11.24;
% Oitava 3 fm3=512/50 +1    fM3=2*512 +1
fm3=11.24; fM3=21.48;
% Oitava 4 fm4=1024/50 +1   fM4=2*1024/50+1
fm4=21.48; fM4=41.96;
% Oitava 5 fm5=2048/50 +1   fM5=2*2048/50 +1
fm5=41.96; fM5=82.92;
%--------------BLOCO DE MONTAGEM DO NOVO ESPECTRO DE FREQUENCIAS----------|
for i=0:n-1
   % cria o bloco de transformadas rapidas
   spectrum_block(c*i+1:c*i+c)= abs(fft(in(c*i+1:c*i+c)));
end
for k =0:n-1
        spectrum_block(c*k+1:c*k+2)=0;    % anular oitava de dc-64 Hz
        spectrum_block(c*k+81)=0;
%-------------------------------------------------------------------------
%                       Oitava 0 (64 - 128 Hz)
alfa0=2; %2
```

```
int0 = c*k+3:c*k+3;
        ft0(int0)= spectrum_block(int0);
  % Encontrando o tom dominante original do programa E0= sqrt(sum(ft0.^2));
        E0= sqrt(sum(ft0.^2));
        spectrum_block(find(ft0 ~= max(ft0)& ft0~=0))= 0;
        surv0 = spectrum_block(find(int0 == max(int0)));
        survivor0=spectrum_block(c*k+3:c*k+3);  % unico tom dominante
        nota0=find(survivor0 ~=0);       % indice da nota sobrevivente
        f0oct0=find(survivor0 ~=0)+2;    % posicao do tom na oitava
        unnormfoct0=(f0oct0-1)*50;       %freq do tom dominante na oitava 0
        % spectrum stuffing via beta distribution
        Q0=(fM0-f0oct0)/(f0oct0-fm0);
        K0=0;
        if (Q0 ~=0)
            K0=1/(((f0oct0-fm0)^(alfa0-1))*(fM0-f0oct0)^(Q0*(alfa0-1)));
        end
       if (nota0 ~=0)
           spectrum_block(c*k+3)=...
                  abs(E0*K0*((3-fm1)^(alfa0-1))*((fM1-3)^(Q0*(alfa0-1))));
       end
       clear peso(3:3);
       clear ft0;
%-------------------------------------------------------------------------
%                       Oitava 1 (128 - 256 Hz)
alfa1=3; %3
int1 = c*k+4:c*k+6;
       ft1(int1)= spectrum_block(int1);
       % Encontrando o tom dominante
        E1= sqrt(sum(ft1.^2));
        spectrum_block(find(ft1 ~= max(ft1)& ft1~=0))= 0;
        surv1 = spectrum_block(find(int1 == max(int1)));
        survivor1=spectrum_block(c*k+4:c*k+6);  % unico tom dominante
        nota1=find(survivor1 ~=0);       % indice da nota sobrevivente
        f0oct1=find(survivor1 ~=0)+3;    % posicao do tom na oitava
        unnormfoct1=(f0oct1-1)*50;       % freq do tom dominante na oitava 1
        % spectrum stuffing via beta distribution
        Q1=(fM1-f0oct1)/(f0oct1-fm1);
        K1=0;
        if (Q1 ~=0)
        K1=1/(((f0oct1-fm1)^(alfa1-1))*(fM1-f0oct1)^(Q1*(alfa1-1)));
        end
        for dummy=4:6
           peso(dummy)=K1*((dummy-fm1)^(alfa1-1))*((fM1-dummy)^...
                       (Q1*(alfa1-1)));  % stuffing
       end
       peso(4:6);
       if (nota1 ~=0)
```

```
            spectrum_block(c*k+4:c*k+6)=E1*peso(4:6);
        end
        clear peso(4:6);
        clear ft1;
%-------------------------------------------------------------------------
%                       Oitava 2 (256 - 512 Hz)
alfa2=4; %4
int2 = c*k+7:c*k+11;
      ft2(int2)= spectrum_block(int2);
      % Encontrando o tom dominante
      E2= sqrt(sum(ft2.^2));
      spectrum_block(find(ft2 ~= max(ft2)& ft2~=0))= 0;
      surv2 = spectrum_block(find(int2 == max(int2)));
      survivor2 = spectrum_block(c*k+7:c*k+11);  % unico tom dominante
      nota2 = find(survivor2 ~=0);       % indice da nota sobrevivente
      f0oct2 = find(survivor2 ~=0)+6;    % posicao do tom na oitava
      unnormfoct2=(f0oct2-1)*50;         % freq do tom dominante na oitava 2
      % spectrum stuffing via beta distribution
      Q2=(fM2-f0oct2)/(f0oct2-fm2);
      K2=0;
      if (Q2 ~= 0)
          K2=1/(((f0oct2-fm2)^(alfa2-1))*(fM2-f0oct2)^(Q2*(alfa2-1)));
      end
      % o dummy corresponde ao x da equacao e o Beta - 1 corresponde ao
      %Q2*(alfa - 1) por manipulacoes na forma da distribuicao beta pegando
      %o valor da moda.
       for dummy=7:11
           peso(dummy)=K2*((dummy-fm2)^(alfa2-1))*((fM2-dummy)^...
                       (Q2*(alfa2-1))); % stuffing
       end
       peso(7:11);
       if (nota2 ~=0)
           spectrum_block(c*k+7:c*k+11)=E2*peso(7:11);
       end
       clear peso(7:11);
       clear ft2;
%-------------------------------------------------------------------------
%                       Oitava 3 (512 - 1024 Hz)
alfa3=4; %4
int3 = c*k+12:c*k+21;
      ft3(int3)=spectrum_block(int3);
      % Encontrando o tom dominante
      E3 = sqrt(sum(ft3.^2));
      spectrum_block(find(ft3 ~= max(ft3)& ft3~=0))=0;
       surv3 = spectrum_block(find(int3 == max(int3)));
       survivor3=spectrum_block(c*k+12:c*k+21);  % unico tom dominante
      nota3=find(survivor3 ~=0);         % indice da nota sobrevivente
```

```
      f0oct3=find(survivor3 ~=0)+11;     % posicao do tom na oitava
      unnormfoct3=(f0oct3-1)*50;         % freq do tom dominante na oitava 3
      % spectrum stuffing via beta distribution
      Q3=(fM3-f0oct3)/(f0oct3-fm3);
      K3=0;
      if (Q3 ~=0)
          K3=1/(((f0oct3-fm3)^(alfa3-1))*(fM3-f0oct3)^(Q3*(alfa3-1)));
      end
      for dummy=12:21
           peso(dummy)=K3*((dummy-fm3)^(alfa3-1))*((fM3-dummy)^...
                      (Q3*(alfa3-1)));   % stuffing
      end
       peso(12:21);
       if (nota3 ~=0)
           spectrum_block(c*k+12:c*k+21)=E3*peso(12:21);
       end
       clear peso(12:21);
       clear ft3;
%-------------------------------------------------------------------------
%                        Oitava 4 (1024 - 2048 Hz)
alfa4=7; %7
int4 = c*k+22:c*k+41;
      ft4(int4)=spectrum_block(int4);
      % Encontrando o tom dominante
      E4= sqrt(sum(ft4.^2));
      spectrum_block(find(ft4 ~= max(ft4)& ft4~=0))=0;
      surv4 = spectrum_block(find(int4 == max(int4)));
      survivor4=spectrum_block(c*k+22:c*k+41);  % unico tom dominante
      nota4=find(survivor4 ~=0);         % indice da nota sobrevivente
      f0oct4=find(survivor4 ~=0)+21;     % posicao do tom na oitava
      unnormfoct4=(f0oct4-1)*50;         % freq do tom dominante na oitava 4
      % spectrum stuffing via beta distribution
      Q4=(fM4-f0oct4)/(f0oct4-fm4);
      K4=0;
      if (Q4 ~=0)
          K4=1/(((f0oct4-fm4)^(alfa4-1))*(fM4-f0oct4)^(Q4*(alfa4-1)));
      end
       for dummy=22:41
           peso(dummy)=K4*((dummy-fm4)^(alfa4-1))*((fM4-dummy)^...
                      (Q4*(alfa4-1)));  % stuffing
       end

       if (nota4 ~=0)
           spectrum_block(c*k+22:c*k+41)=E4*peso(22:41);
       end
       clear peso(22:41);
       clear ft4;
```

```
%-------------------------------------------------------------------------
%                        Oitava 5 (2048 - 4006 Hz)
alfa5=7; %7
        int5 = c*k+42:c*k+80;
        ft5(int5)=spectrum_block(int5);

        % Encontrando o tom dominante
        E5= sqrt(sum(ft5.^2));
        spectrum_block(find(ft5 ~= max(ft5)&ft5~=0))=0;
        surv5 = spectrum_block(find(int5 == max(int5)));
        survivor5=spectrum_block(c*k+42:c*k+80);  % unico tom dominante
        nota5=find(survivor5 ~=0 );     % indice da nota sobrevivente
        f0oct5=find(survivor5 ~=0 )+41; % posicao do tom na oitava
        unnormfoct5=(f0oct5-1)*50;      % freq do tom dominante na oitava 5
        % spectrum stuffing via beta distribution
        Q5=(fM5-f0oct5)/(f0oct5-fm5);
        K5=0;
       if (Q5 ~=0)
            K5=1/(((f0oct5-fm5)^(alfa5-1))*(fM5-f0oct5)^(Q5*(alfa5-1)));
       end
       for dummy=42:80
           peso(dummy)=K5*((dummy-fm5)^(alfa5-1))*((fM5-dummy)^...
                       (Q5*(alfa5-1))); % stuffing
       end
       if (nota5 ~=0)
           spectrum_block(c*k+42:c*k+80)=E5*peso(42:80);
       end
       clear peso(42:80);
       clear ft5;
%lista dos tons sobreviventes por oitava, para cada quadro k
spectrum_block(c*k+1:c*k+c) = [spectrum_block(c*k+1:c*k+80)...
                               0 spectrum_block(c*k+80:-1:2+c*k)];
end
%---------- BLOCO DE MONTAGEM DO NOVO ARQUIVO DE VOZ SINTETIZADO  --------|
% concatenacao de blocos
aux=~isnan(spectrum_block);
spectrum_block(find(aux == 0)) = 0;
for i=0:n-1
arquivoVozSintetizado(c*i+1:c*i+c)= ifft(spectrum_block(c*i+1:c*i+c));
end
arquivoVozSintetizado = arquivoVozSintetizado(1:N)';
  for i=0:n-2
  arquivoVozSintetizado(c*i+1:c*i+c)=...
                          (HAM(1:c)).*arquivoVozSintetizado(c*i+1:c*i+c);
  end
out = arquivoVozSintetizado;
end
```

SpecStuffWindow.m

```
function [out] = SpecStuffWindow(in)
%SPECSTUFFINGWINDOW    Tecnica de Preenchimento Espectral via Distribuicao
%                      Beta de  Probabilidade com janelamento extra.
%
%Autores:          Roberto F. B. Sotero Filho e Helio Magalhaes de Oliveira
%E-mail:           rsotero@hotmail.com e hmo@ufpe.br
%Universidade:     Universidade Federal de Pernambuco
%Data:             10/2008


arquivoVozSintetizado=SpecStuffing(in);  % chamada do SpecStuffing
fs = 8000;                               % frequencia amostragem utilizada.
N = length(in);                          % comprimento arquivo de entrada
c = fs*20e-3;                            % numero de amostras em 20 ms.
n = N/c;

HAM = hamming(c);                       %criando janela de Hamming

  for i=0:n-2
  %janelamento extra
   arquivoVozSintetizado(c*i+1:c*i+c)=...
                          (HAM(1:c)).*arquivoVozSintetizado(c*i+1:c*i+c);
  end
out = arquivoVozSintetizado;
end
```

ANEXO B – CÓDIGOS FONTE (*RAL*)

Requerido MATLAB® 2013 ou versões mais recentes.

RecLoc.m

```
function varargout = RecLoc(varargin)
%RECLOC          Software de Reconhecimento de Locutor baseado no Mascara-
%                mento Pleno em Frequencia por Oitava. Eh possivel o cadas-
%                tro de 5(cinco) a 20(vinte) usuarios distintos atraves de
%                5 elocucoes de treinamento por usuario. Aceita como entra-
%                da arquivo com extensao txt, produzido pelo software
%                PadraoLoc, que contem os padroes dos usuarios a serem
%                cadastrados no sistema. Retorna como saida o resultado da
%                IAL, exibindo o usuario detentor da elocucao em teste. O
%                sistema, tambem, exibe uma classificacao para os outros
%                usuarios, ordenando-nos pelos de padrao de voz mais seme-
%                lhante ao do detentor da elocucao em teste.
%
%Autores:        Roberto F. B. Sotero Filho e Helio Magalhaes de Oliveira
%E-mail:         rsotero@hotmail.com e hmo@ufpe.br
%Universidade:   Universidade Federal de Pernambuco
%Data:           04/06/2017

% Codigo de inicializacao do programa - nao editavel
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @RecLoc_OpeningFcn, ...
                   'gui_OutputFcn',  @RecLoc_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% Chamada dos logos  na tela.
function ufpe_CreateFcn(hObject, eventdata, handles)
A = imread('figuras\ufpectgdes.png');
```

```
imshow(A);
% ------------ Rotinas e funcoes internas do programa --------------------
function RecLoc_OpeningFcn(hObject, ~, handles, varargin)
handles.output = hObject;
guidata(hObject, handles);

function varargout = RecLoc_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;
% campo de texto para eloucacao teste
function textoteste_Callback(hObject, eventdata, handles)
function textoteste_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
                   get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
% campo de texto para resultado das indentificacoes
function texto
Callback(hObject, eventdata, handles)
function textoident_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
                   get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function popupmenu1_Callback(hObject, eventdata, handles)
 contents1 = cellstr(get(hObject,'String'));
 b1 = contents1{get(hObject,'Value')};

function popupmenu1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function popupmenu2_Callback(hObject, eventdata, handles)
contents2 = cellstr(get(hObject,'String'));
 b2 = contents2{get(hObject,'Value')};
 contents1 = get(handles.popupmenu1,'string');
 b1 = contents1{get(hObject,'Value')};

function popupmenu2_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
% Acoes executadas ao pressionar botao "abrir" geracao dos padroes.
function abrirpadrao_Callback(hObject, eventdata, handles)
```

```
v = get(handles.popupmenu1,'Value');
v1 = get(handles.popupmenu2,'Value');
indice1 = v-1;
indice2 = v1-1;
    if isequal(indice1,0) || isequal(indice2,0)
       disp('Preencher todos os atributos')   %mensagem preencher atributos
    end
    if isequal(indice1,1) && ~isequal(indice2,0)
         PadraoLoc5                           %chama tela de padrao 5 loc
    end
    if isequal(indice1,2) && ~isequal(indice2,0)
         PadraoLoc6                           %chama tela de padrao 6 loc
    end
     if isequal(indice1,3) && ~isequal(indice2,0)
         PadraoLoc7                           %chama tela de padrao 7 loc
     end
     if isequal(indice1,4) && ~isequal(indice2,0)
         PadraoLoc8                           %chama tela de padrao 8 loc
     end
    if isequal(indice1,5) && ~isequal(indice2,0)
         PadraoLoc9                           %chama tela de padrao 9 loc
    end
     if isequal(indice1,6) && ~isequal(indice2,0)
        PadraoLoc10                           %chama tela de padrao 10 loc
     end
     if isequal(indice1,7) && ~isequal(indice2,0)
        PadraoLoc11                           %chama tela de padrao 11 loc
     end
     if isequal(indice1,8) && ~isequal(indice2,0)
        PadraoLoc12                           %chama tela de padrao 12 loc
     end
     if isequal(indice1,9) && ~isequal(indice2,0)
        PadraoLoc13                           %chama tela de padrao 13 loc
     end
     if isequal(indice1,10) && ~isequal(indice2,0)
        PadraoLoc14                           %chama tela de padrao 14 loc
     end
     if isequal(indice1,11) && ~isequal(indice2,0)
        PadraoLoc15                           %chama tela de padrao 15 loc
     end
     if isequal(indice1,12) && ~isequal(indice2,0)
        PadraoLoc16                           %chama tela de padrao 16 loc
     end
     if isequal(indice1,13) && ~isequal(indice2,0)
        PadraoLoc17                           %chama tela de padrao 17 loc
     end
     if isequal(indice1,14) && ~isequal(indice2,0)
```

```
        PadraoLoc18                              %chama tela de padrao 18 loc
     end
     if isequal(indice1,15) && ~isequal(indice2,0)
        PadraoLoc19                              %chama tela de padrao 19 loc
     end
     if isequal(indice1,16) && ~isequal(indice2,0)
        PadraoLoc20                              %chama tela de padrao 20 loc
     end
%campo de texto do caminho do arquivo txt padroes de voz
function textopadrao_Callback(hObject, eventdata, handles)
function textopadrao_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
%campo de texto dos usuarios cadastrados
function textocadastrados_CreateFcn(hObject, eventdata, handles)
%--Acoes executadas ao pressionar botao "selecionar padrao dos locutores"--
function selepadrao_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.txt',...
    'Selecione o arquivo .txt do padrao dos locutores');      %selecionar
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')                       %cancelar
else
%formacao da string com diretorio e nome do arquivo selecionado('C:\x.txt')
  arquivovoz= strcat(diretorio,nome);
  set(handles.textopadrao,'string',arquivovoz); %envia 'C:\x.txt' p/textbox
  set(handles.textocadastrados2,'string',''); %limpa texto cadastrados 2
  set(handles.textclassificacao,'string',''); %limpa texto classificacao
  set(handles.textclassificacao2,'string','');%limpa texto classificacao2
    tabelacomp = readtable(arquivovoz);          %ler tabela dos padroes
    tabelacell1 = table2cell(tabelacomp);        % transforma em celula
    [linha coluna] = size(tabelacell1);   %tamanho das linhas e colunas

 % ------ leitura das strings dos locutores na matriz padroes de voz ------
    stringloc1=tabelacell1(1,1); stringloc1char=char(stringloc1);
    stringloc2=tabelacell1(2,1); stringloc2char=char(stringloc2);
    stringloc3=tabelacell1(3,1); stringloc3char=char(stringloc3);
    stringloc4=tabelacell1(4,1); stringloc4char=char(stringloc4);
    stringloc5=tabelacell1(5,1); stringloc5char=char(stringloc5);
if linha>=6
    stringloc6=tabelacell1(6,1); stringloc6char=char(stringloc6);
end
if linha>=7
    stringloc7=tabelacell1(7,1); stringloc7char=char(stringloc7);
end
if linha>=8
```

```
    stringloc8=tabelacell1(8,1); stringloc8char=char(stringloc8);
end
if linha>=9
    stringloc9=tabelacell1(9,1); stringloc9char=char(stringloc9);
end
if linha>=10
    stringloc10=tabelacell1(10,1); stringloc10char=char(stringloc10);
end
if linha>=11
    stringloc11=tabelacell1(11,1); stringloc11char=char(stringloc11);
end
if linha>=12
   stringloc12=tabelacell1(12,1); stringloc12char=char(stringloc12);
end
if linha>=13
    stringloc13=tabelacell1(13,1); stringloc13char=char(stringloc13);
end
if linha>=14
    stringloc14=tabelacell1(14,1); stringloc14char=char(stringloc14);
end
if linha>=15
    stringloc15=tabelacell1(15,1); stringloc15char=char(stringloc15);
end
if linha>=16
stringloc16=tabelacell1(16,1); stringloc16char=char(stringloc16);
end
if linha>=17
    stringloc17=tabelacell1(17,1); stringloc17char=char(stringloc17);
end
if linha>=18
    stringloc18=tabelacell1(18,1); stringloc18char=char(stringloc18);
end
if linha>=19
    stringloc19=tabelacell1(19,1); stringloc19char=char(stringloc19);
end
if linha==20
    stringloc20=tabelacell1(20,1); stringloc20char=char(stringloc20);
end
%------------------  5 LOCUTORES CADASTRADOS  -------------------------
if linha==5
    %forma vetor de strings
    vetorstring={stringloc1char;stringloc2char;stringloc3char;...
                 stringloc4char;stringloc5char};
end
%------------------- 6 LOCUTORES CADASTRADOS  -------------------------
if linha==6
    %forma vetor de strings
```

```
    vetorstring={stringloc1char;stringloc2char;stringloc3char;...
                 stringloc4char;stringloc5char;stringloc6char};
end
%------------------  7 LOCUTORES CADASTRADOS  -------------------------
if linha==7
    %forma vetor de strings
    vetorstring={stringloc1char;stringloc2char;stringloc3char;...
                 stringloc4char;stringloc5char;stringloc6char;...
                 stringloc7char};
end
%------------------  8 LOCUTORES CADASTRADOS  -------------------------
if linha==8
    %forma vetor de strings
    vetorstring={stringloc1char;stringloc2char;stringloc3char;...
                 stringloc4char;stringloc5char;stringloc6char;...
                 stringloc7char;stringloc8char};
end
%------------------  9 LOCUTORES CADASTRADOS  -------------------------
if linha==9
    %forma vetor de strings
    vetorstring={stringloc1char;stringloc2char;stringloc3char;...
                 stringloc4char;stringloc5char;stringloc6char;...
                 stringloc7char;stringloc8char;stringloc9char};
end
%------------------  10 LOCUTORES CADASTRADOS  ------------------------
if linha==10
    %forma vetor de strings
    vetorstring={stringloc1char;stringloc2char;stringloc3char;...
                 stringloc4char;stringloc5char;stringloc6char;...
                 stringloc7char;stringloc8char;stringloc9char;...
                 stringloc10char};
end
%------------------  11 LOCUTORES CADASTRADOS  ------------------------
if linha==11
    %forma vetor de strings
    vetorstring={stringloc1char;stringloc2char;stringloc3char;...
                 stringloc4char;stringloc5char;stringloc6char;...
                 stringloc7char;stringloc8char;stringloc9char;...
                 stringloc10char};
    vetorstring2={stringloc11char};
end
%------------------  12 LOCUTORES CADASTRADOS  ------------------------
if linha==12
    %forma vetor de strings
    vetorstring={stringloc1char;stringloc2char;stringloc3char;...
                 stringloc4char;stringloc5char;stringloc6char;...
                 stringloc7char;stringloc8char;stringloc9char;...
```

```
                stringloc10char};
    vetorstring2={stringloc11char;stringloc12char};
end
%------------------  13 LOCUTORES CADASTRADOS  -------------------------
if linha==13
    %forma vetor de strings
    vetorstring={stringloc1char;stringloc2char;stringloc3char;...
                stringloc4char;stringloc5char;stringloc6char;...
                stringloc7char;stringloc8char;stringloc9char;...
                stringloc10char};
    vetorstring2={stringloc11char;stringloc12char;stringloc13char};
end
%------------------  14 LOCUTORES CADASTRADOS  -------------------------
if linha ==14
     %forma vetor de strings
    vetorstring={stringloc1char;stringloc2char;stringloc3char;...
                stringloc4char;stringloc5char;stringloc6char;...
                stringloc7char;stringloc8char;stringloc9char;...
                stringloc10char};
    vetorstring2={stringloc11char;stringloc12char;stringloc13char;...
                 stringloc14char};
end
%------------------  15 LOCUTORES CADASTRADOS  -------------------------
if linha==15
    vetorstring={stringloc1char;stringloc2char;stringloc3char;...
                stringloc4char;stringloc5char;stringloc6char;...
                stringloc7char;stringloc8char;stringloc9char;...
                stringloc10char};
    vetorstring2={stringloc11char;stringloc12char;stringloc13char;...
                 stringloc14char;stringloc15char};
end
%------------------  16 LOCUTORES CADASTRADOS  -------------------------
if linha==16
    vetorstring={stringloc1char;stringloc2char;stringloc3char;...
                stringloc4char;stringloc5char;stringloc6char;...
                stringloc7char;stringloc8char;stringloc9char;...
                stringloc10char};
    vetorstring2={stringloc11char;stringloc12char;stringloc13char;...
                 stringloc14char;stringloc15char;stringloc16char};
end
%------------------  17 LOCUTORES CADASTRADOS  -------------------------
if linha==17
    vetorstring={stringloc1char;stringloc2char;stringloc3char;...
                stringloc4char;stringloc5char;stringloc6char;...
                stringloc7char;stringloc8char;stringloc9char;...
                stringloc10char};
    vetorstring2={stringloc11char;stringloc12char;stringloc13char;...
```

```
                stringloc14char;stringloc15char;stringloc16char;...
                stringloc17char};
end
%------------------  18 LOCUTORES CADASTRADOS  -------------------------
if linha==18
    vetorstring={stringloc1char;stringloc2char;stringloc3char;...
                stringloc4char;stringloc5char;stringloc6char;...
                stringloc7char;stringloc8char;stringloc9char;...
                stringloc10char};
    vetorstring2={stringloc11char;stringloc12char;stringloc13char;...
                stringloc14char;stringloc15char;stringloc16char;...
                stringloc17char;stringloc18char};
end
%------------------  19 LOCUTORES CADASTRADOS  -------------------------
if linha==19
    vetorstring={stringloc1char;stringloc2char;stringloc3char;...
                stringloc4char;stringloc5char;stringloc6char;...
                stringloc7char;stringloc8char;stringloc9char;...
                stringloc10char};
    vetorstring2={stringloc11char;stringloc12char;stringloc13char;...
                stringloc14char;stringloc15char;stringloc16char;...
                stringloc17char;stringloc18char;stringloc19char};
end
%------------------  20 LOCUTORES CADASTRADOS  -------------------------
if linha==20
    vetorstring={stringloc1char;stringloc2char;stringloc3char;...
                stringloc4char;stringloc5char;stringloc6char;...
                stringloc7char;stringloc8char;stringloc9char;...
                stringloc10char};
    vetorstring2={stringloc11char;stringloc12char;stringloc13char;...
                stringloc14char;stringloc15char;stringloc16char;...
                stringloc17char;stringloc18char;stringloc19char;...
                stringloc20char};
end
%envia strings para tela em "Usuarios Cadastrados"
    set(handles.textocadastrados,'string',vetorstring);
    if linha>=11
        set(handles.textocadastrados2,'string',vetorstring2);
    end
end
%Acoes executadas ao pressionar botao "selecionar elocucao teste (wav)"
function selelocteste_Callback(~, ~, handles)
[nome,diretorio] = uigetfile('*.wav',...
    'Selecione o arquivo .wav de teste para processamento'); %selecionar
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')                      %cancelar
else
```

```
    arquivovoz= strcat(diretorio,nome);
    set(handles.textoteste,'string',arquivovoz);
end
%Acoes executadas ao se pressionar o botao "processar"
function resultado_Callback(~, ~, handles)
eloctestestring = get(handles.textoteste,'string'); %le caminho da elocucao
elocteste = audioread(eloctestestring);        %le arquivo wav
vetorCaracteristico=vetorCaract(elocteste);    %monta vetor caracteristico
string_padrao = get(handles.textopadrao,'string');  %le caminha arquivo txt
matrizpadrao = readtable(string_padrao);       %le matriz padrao
matrizpadraocell = table2cell(matrizpadrao);   %gera celular da matriz
[linha coluna] = size(matrizpadraocell);       %tamanho da matriz
tabelacomp = readtable(string_padrao);
tabelacell1 = table2cell(tabelacomp);

if linha>=5
    %leitura, no arquivo dos padroes, do padrao de cada locutor
    locutor1=matrizpadraocell(1,2:8); locutor1mat=cell2mat(locutor1);
    locutor2=matrizpadraocell(2,2:8); locutor2mat=cell2mat(locutor2);
    locutor3=matrizpadraocell(3,2:8); locutor3mat=cell2mat(locutor3);
    locutor4=matrizpadraocell(4,2:8); locutor4mat=cell2mat(locutor4);
    locutor5=matrizpadraocell(5,2:8); locutor5mat=cell2mat(locutor5);
    %calculos dos erros
    erroloc1 = vetorCaracteristico - locutor1mat;
    erroloc2 = vetorCaracteristico - locutor2mat;
    erroloc3 = vetorCaracteristico - locutor3mat;
    erroloc4 = vetorCaracteristico - locutor4mat;
    erroloc5 = vetorCaracteristico - locutor5mat;
    %calculo das normas
    norm_loc1 = norm(erroloc1,2);
    norm_loc2 = norm(erroloc2,2);
    norm_loc3 = norm(erroloc3,2);
    norm_loc4 = norm(erroloc4,2);
    norm_loc5 = norm(erroloc5,2);
    %leitura, no arquivo padroes dos locutores, do nome de cada locutor
    stringloc1=tabelacell1(1,1); sloc1char=char(stringloc1);
    stringloc2=tabelacell1(2,1); sloc2char=char(stringloc2);
    stringloc3=tabelacell1(3,1); sloc3char=char(stringloc3);
    stringloc4=tabelacell1(4,1); sloc4char=char(stringloc4);
    stringloc5=tabelacell1(5,1); sloc5char=char(stringloc5);
end
if linha>=6
    locutor6=matrizpadraocell(6,2:8); locutor6mat=cell2mat(locutor6);
    erroloc6 = vetorCaracteristico - locutor6mat;
    norm_loc6 = norm(erroloc6,2);
    stringloc6=tabelacell1(6,1); sloc6char=char(stringloc6);
end
```

```
if linha>=7
    locutor7=matrizpadraocell(7,2:8); locutor7mat=cell2mat(locutor7);
    erroloc7 = vetorCaracteristico - locutor7mat;
    norm_loc7 = norm(erroloc7,2);
    stringloc7=tabelacell1(7,1); sloc7char=char(stringloc7);
end
if linha>=8
    locutor8=matrizpadraocell(8,2:8); locutor8mat=cell2mat(locutor8);
    erroloc8 = vetorCaracteristico - locutor8mat;
    norm_loc8 = norm(erroloc8,2);
    stringloc8=tabelacell1(8,1); sloc8char=char(stringloc8);
end
if linha>=9
    locutor9=matrizpadraocell(9,2:8); locutor9mat=cell2mat(locutor9);
    erroloc9 = vetorCaracteristico - locutor9mat;
    norm_loc9 = norm(erroloc9,2);
    stringloc9=tabelacell1(9,1); sloc9char=char(stringloc9);
end
if linha>=10
    locutor10=matrizpadraocell(10,2:8); locutor10mat=cell2mat(locutor10);
    erroloc10 = vetorCaracteristico - locutor10mat;
    norm_loc10 = norm(erroloc10,2);
    stringloc10=tabelacell1(10,1); sloc10char=char(stringloc10);
end
if linha>=11
    locutor11=matrizpadraocell(11,2:8); locutor11mat=cell2mat(locutor11);
    erroloc11 = vetorCaracteristico - locutor11mat;
    norm_loc11 = norm(erroloc11,2);
    stringloc11=tabelacell1(11,1); sloc11char=char(stringloc11);
end
if linha>=12
    locutor12=matrizpadraocell(12,2:8); locutor12mat=cell2mat(locutor12);
    erroloc12 = vetorCaracteristico - locutor12mat;
    norm_loc12 = norm(erroloc12,2);
    stringloc12=tabelacell1(12,1); sloc12char=char(stringloc12);
end
if linha>=13
    locutor13=matrizpadraocell(13,2:8); locutor13mat=cell2mat(locutor13);
    erroloc13 = vetorCaracteristico - locutor13mat;
    norm_loc13 = norm(erroloc13,2);
    stringloc13=tabelacell1(13,1); sloc13char=char(stringloc13);
end
if linha>=14
    locutor14=matrizpadraocell(14,2:8); locutor14mat=cell2mat(locutor14);
    erroloc14 = vetorCaracteristico - locutor14mat;
    norm_loc14 = norm(erroloc14,2);
    stringloc14=tabelacell1(14,1); sloc14char=char(stringloc14);
```

```
end
if linha>=15
    locutor15=matrizpadraocell(15,2:8); locutor15mat=cell2mat(locutor15);
    erroloc15 = vetorCaracteristico - locutor15mat;
    norm_loc15 = norm(erroloc15,2);
    stringloc15=tabelacell1(15,1); sloc15char=char(stringloc15);
end
if linha>=16
    locutor16=matrizpadraocell(16,2:8); locutor16mat=cell2mat(locutor16);
    erroloc16 = vetorCaracteristico - locutor16mat;
    norm_loc16 = norm(erroloc16,2);
    stringloc16=tabelacell1(16,1); sloc16char=char(stringloc16);
end
if linha>=17
    locutor17=matrizpadraocell(17,2:8); locutor17mat=cell2mat(locutor17);
    erroloc17 = vetorCaracteristico - locutor17mat;
    norm_loc17 = norm(erroloc17,2);
    stringloc17=tabelacell1(17,1); sloc17char=char(stringloc17);
end
if linha>=18
    locutor18=matrizpadraocell(18,2:8); locutor18mat=cell2mat(locutor18);
    erroloc18 = vetorCaracteristico - locutor18mat;
    norm_loc18 = norm(erroloc18,2);
    stringloc18=tabelacell1(18,1); sloc18char=char(stringloc18);
end
if linha>=19
    locutor19=matrizpadraocell(19,2:8); locutor19mat=cell2mat(locutor19);
    erroloc19 = vetorCaracteristico - locutor19mat;
    norm_loc19 = norm(erroloc19,2);
    stringloc19=tabelacell1(19,1); sloc19char=char(stringloc19);
end
if linha>=20
    locutor20=matrizpadraocell(20,2:8); locutor20mat=cell2mat(locutor20);
    erroloc20 = vetorCaracteristico - locutor20mat;
    norm_loc20 = norm(erroloc20,2);
    stringloc20=tabelacell1(20,1); sloc20char=char(stringloc20);
end
%--------ROTINA PARA ORDENACAO DOS 5 LOCUTORES PELOS DE MENOR NORMA--------
if linha==5
    for i=1:4
        if norm_loc1<norm_loc2
            naux = norm_loc2; norm_loc2=norm_loc1; norm_loc1 = naux;
            stringlocaux=sloc2char; sloc2char=sloc1char;
            sloc1char=stringlocaux;
        end
        if norm_loc2<norm_loc3
            naux = norm_loc3; norm_loc3=norm_loc2; norm_loc2 = naux;
```

```
            stringlocaux=sloc3char; sloc3char=sloc2char;
            sloc2char=stringlocaux;
        end
        if norm_loc3<norm_loc4
            naux = norm_loc4; norm_loc4=norm_loc3; norm_loc3 = naux;
            stringlocaux=sloc4char; sloc4char=sloc3char;
            sloc3char=stringlocaux;
        end
        if norm_loc4<norm_loc5
            naux = norm_loc5; norm_loc5=norm_loc4; norm_loc4 = naux;
            stringlocaux=sloc5char; sloc5char=sloc4char;
            sloc4char=stringlocaux;
        end
        i=i+1;
    end
%montagem do vetor com string ordenada dos locutores
vetor_stringsort = {sloc5char;sloc4char;sloc3char;sloc2char;sloc1char};
%envia string de elocutores ordenados para tela
set(handles.textclassificacao,'string',vetor_stringsort);
%indicar locutor pretenso no visor
set(handles.textoident,'string',sloc5char);
end
%--------ROTINA PARA ORDENACAO DOS 6 LOCUTORES PELOS DE MENOR NORMA--------
if linha==6
    for i=1:5
        if norm_loc1<norm_loc2
            naux = norm_loc2; norm_loc2=norm_loc1; norm_loc1 = naux;
            stringlocaux=sloc2char; sloc2char=sloc1char;
            sloc1char=stringlocaux;
        end
        if norm_loc2<norm_loc3
            naux = norm_loc3; norm_loc3=norm_loc2; norm_loc2 = naux;
            stringlocaux=sloc3char; sloc3char=sloc2char;
            sloc2char=stringlocaux;
        end
        if norm_loc3<norm_loc4
            naux = norm_loc4; norm_loc4=norm_loc3; norm_loc3 = naux;
            stringlocaux=sloc4char; sloc4char=sloc3char;
            sloc3char=stringlocaux;
        end
        if norm_loc4<norm_loc5
            naux = norm_loc5; norm_loc5=norm_loc4; norm_loc4 = naux;
            stringlocaux=sloc5char; sloc5char=sloc4char;
            sloc4char=stringlocaux;
        end
        if norm_loc5<norm_loc6
            naux = norm_loc6; norm_loc6=norm_loc5; norm_loc5 = naux;
```

```
            stringlocaux=sloc6char; sloc6char=sloc5char;
            sloc5char=stringlocaux;
        end
        i=i+1;
    end
%montagem do vetor com string ordenada dos locutores
vetor_stringsort = {sloc6char;sloc5char;sloc4char;sloc3char;...
                    sloc2char;sloc1char};
%envia string de elocutores ordenados para tela
set(handles.textclassificacao,'string',vetor_stringsort);
%indicar locutor pretenso no visor
set(handles.textoident,'string',sloc6char);
end

%------ROTINA PARA ORDENACAO DOS 7 LOCUTORES PELOS DE MENOR NORMA----------
if linha==7
    for i=1:6
        if norm_loc1<norm_loc2
            naux = norm_loc2; norm_loc2=norm_loc1; norm_loc1 = naux;
            stringlocaux=sloc2char; sloc2char=sloc1char;
            sloc1char=stringlocaux;
        end
        if norm_loc2<norm_loc3
            naux = norm_loc3; norm_loc3=norm_loc2; norm_loc2 = naux;
            stringlocaux=sloc3char; sloc3char=sloc2char;
            sloc2char=stringlocaux;
        end
        if norm_loc3<norm_loc4
            naux = norm_loc4; norm_loc4=norm_loc3; norm_loc3 = naux;
            stringlocaux=sloc4char; sloc4char=sloc3char;
            sloc3char=stringlocaux;
        end
        if norm_loc4<norm_loc5
            naux = norm_loc5; norm_loc5=norm_loc4; norm_loc4 = naux;
            stringlocaux=sloc5char; sloc5char=sloc4char;
            sloc4char=stringlocaux;
        end
        if norm_loc5<norm_loc6
            naux = norm_loc6; norm_loc6=norm_loc5; norm_loc5 = naux;
            stringlocaux=sloc6char; sloc6char=sloc5char;
            sloc5char=stringlocaux;
        end
        if norm_loc6<norm_loc7
            naux = norm_loc7; norm_loc7=norm_loc6; norm_loc6 = naux;
            stringlocaux=sloc7char; sloc7char=sloc6char;
            sloc6char=stringlocaux;
        end
```

```
        i=i+1;
    end
%montagem do vetor com string ordenada dos locutores
vetor_stringsort = {sloc7char;sloc6char;sloc5char;sloc4char;...
                    sloc3char;sloc2char;sloc1char};
%envia string de elocutores ordenados para tela
set(handles.textclassificacao,'string',vetor_stringsort);
%indicar locutor pretenso no visor
set(handles.textoident,'string',sloc7char);
end
%-------ROTINA PARA ORDENACAO DOS 8 LOCUTORES PELOS DE MENOR NORMA---------
if linha==8
    for i=1:7
        if norm_loc1<norm_loc2
            naux = norm_loc2; norm_loc2=norm_loc1; norm_loc1 = naux;
            stringlocaux=sloc2char; sloc2char=sloc1char;
            sloc1char=stringlocaux;
        end
        if norm_loc2<norm_loc3
            naux = norm_loc3; norm_loc3=norm_loc2; norm_loc2 = naux;
            stringlocaux=sloc3char; sloc3char=sloc2char;
            sloc2char=stringlocaux;
        end
        if norm_loc3<norm_loc4
            naux = norm_loc4; norm_loc4=norm_loc3; norm_loc3 = naux;
            stringlocaux=sloc4char; sloc4char=sloc3char;
            sloc3char=stringlocaux;
        end
        if norm_loc4<norm_loc5
            naux = norm_loc5; norm_loc5=norm_loc4; norm_loc4 = naux;
            stringlocaux=sloc5char; sloc5char=sloc4char;
            sloc4char=stringlocaux;
        end
        if norm_loc5<norm_loc6
            naux = norm_loc6; norm_loc6=norm_loc5; norm_loc5 = naux;
            stringlocaux=sloc6char; sloc6char=sloc5char;
            sloc5char=stringlocaux;
        end
        if norm_loc6<norm_loc7
            naux = norm_loc7; norm_loc7=norm_loc6; norm_loc6 = naux;
            stringlocaux=sloc7char; sloc7char=sloc6char;
            sloc6char=stringlocaux;
        end
        if norm_loc7<norm_loc8
            naux = norm_loc8; norm_loc8=norm_loc7; norm_loc7 = naux;
            stringlocaux=sloc8char; sloc8char=sloc7char;
            sloc7char=stringlocaux;
```

```
        end
        i=i+1;
    end
%montagem do vetor com string ordenada dos locutores
vetor_stringsort = {sloc8char;sloc7char;sloc6char;sloc5char;sloc4char;...
                    sloc3char;sloc2char;sloc1char};
%envia string de elocutores ordenados para tela
set(handles.textclassificacao,'string',vetor_stringsort);
%indicar locutor pretenso no visor
set(handles.textoident,'string',sloc8char);
end
%--------ROTINA PARA ORDENACAO DOS 9 LOCUTORES PELOS DE MENOR NORMA--------
if linha==9
    for i=1:8
        if norm_loc1<norm_loc2
            naux = norm_loc2; norm_loc2=norm_loc1; norm_loc1 = naux;
            stringlocaux=sloc2char; sloc2char=sloc1char;
            sloc1char=stringlocaux;
        end
        if norm_loc2<norm_loc3
            naux = norm_loc3; norm_loc3=norm_loc2; norm_loc2 = naux;
            stringlocaux=sloc3char; sloc3char=sloc2char;
            sloc2char=stringlocaux;
        end
        if norm_loc3<norm_loc4
            naux = norm_loc4; norm_loc4=norm_loc3; norm_loc3 = naux;
            stringlocaux=sloc4char; sloc4char=sloc3char;
            sloc3char=stringlocaux;
        end
        if norm_loc4<norm_loc5
            naux = norm_loc5; norm_loc5=norm_loc4; norm_loc4 = naux;
            stringlocaux=sloc5char; sloc5char=sloc4char;
            sloc4char=stringlocaux;
        end
        if norm_loc5<norm_loc6
            naux = norm_loc6; norm_loc6=norm_loc5; norm_loc5 = naux;
            stringlocaux=sloc6char; sloc6char=sloc5char;
            sloc5char=stringlocaux;
        end
        if norm_loc6<norm_loc7
            naux = norm_loc7; norm_loc7=norm_loc6; norm_loc6 = naux;
            stringlocaux=sloc7char; sloc7char=sloc6char;
            sloc6char=stringlocaux;
        end
        if norm_loc7<norm_loc8
            naux = norm_loc8; norm_loc8=norm_loc7; norm_loc7 = naux;
            stringlocaux=sloc8char; sloc8char=sloc7char;
```

```
            sloc7char=stringlocaux;
        end
        if norm_loc8<norm_loc9
            naux = norm_loc9; norm_loc9=norm_loc8; norm_loc8 = naux;
            stringlocaux=sloc9char; sloc9char=sloc8char;
            sloc8char=stringlocaux;
        end
        i=i+1;
    end
%montagem do vetor com string ordenada dos locutores
vetor_stringsort = {sloc9char;sloc8char;sloc7char;sloc6char;...
                    sloc5char;sloc4char;sloc3char;sloc2char;sloc1char};
%envia string de elocutores ordenados para tela
set(handles.textclassificacao,'string',vetor_stringsort);
%indicar locutor pretenso no visor
set(handles.textoident,'string',sloc9char);
end
%--------ROTINA PARA ORDENACAO DOS 10 LOCUTORES PELOS DE MENOR NORMA-------
if linha==10
    for i=1:9
        if norm_loc1<norm_loc2
            naux = norm_loc2; norm_loc2=norm_loc1; norm_loc1 = naux;
            stringlocaux=sloc2char; sloc2char=sloc1char;
            sloc1char=stringlocaux;
        end
        if norm_loc2<norm_loc3
            naux = norm_loc3; norm_loc3=norm_loc2; norm_loc2 = naux;
            stringlocaux=sloc3char; sloc3char=sloc2char;
            sloc2char=stringlocaux;
        end
        if norm_loc3<norm_loc4
            naux = norm_loc4; norm_loc4=norm_loc3; norm_loc3 = naux;
            stringlocaux=sloc4char; sloc4char=sloc3char;
            sloc3char=stringlocaux;
        end
        if norm_loc4<norm_loc5
            naux = norm_loc5; norm_loc5=norm_loc4; norm_loc4 = naux;
            stringlocaux=sloc5char; sloc5char=sloc4char;
            sloc4char=stringlocaux;
        end
        if norm_loc5<norm_loc6
            naux = norm_loc6; norm_loc6=norm_loc5; norm_loc5 = naux;
            stringlocaux=sloc6char; sloc6char=sloc5char;
            sloc5char=stringlocaux;
        end
        if norm_loc6<norm_loc7
            naux = norm_loc7; norm_loc7=norm_loc6; norm_loc6 = naux;
```

```
            stringlocaux=sloc7char; sloc7char=sloc6char;
            sloc6char=stringlocaux;
        end
        if norm_loc7<norm_loc8
            naux = norm_loc8; norm_loc8=norm_loc7; norm_loc7 = naux;
            stringlocaux=sloc8char; sloc8char=sloc7char;
            sloc7char=stringlocaux;
        end
        if norm_loc8<norm_loc9
            naux = norm_loc9; norm_loc9=norm_loc8; norm_loc8 = naux;
            stringlocaux=sloc9char; sloc9char=sloc8char;
            sloc8char=stringlocaux;
        end
        if norm_loc9<norm_loc10
            naux = norm_loc10; norm_loc10=norm_loc9; norm_loc9 = naux;
            stringlocaux=sloc10char; sloc10char=sloc9char;
            sloc9char=stringlocaux;
        end
        i=i+1;
    end
%montagem do vetor com string ordenada dos locutores
vetor_stringsort = {sloc10char;sloc9char;sloc8char;sloc7char;sloc6char;...
                    sloc5char;sloc4char;sloc3char;sloc2char;sloc1char};
%envia string de elocutores ordenados para tela
set(handles.textclassificacao,'string',vetor_stringsort);
%indicar locutor pretenso no visor
set(handles.textoident,'string',sloc10char);
end
%--------ROTINA PARA ORDENACAO DOS 11 LOCUTORES PELOS DE MENOR NORMA-------
if linha==11
    for i=1:10
        if norm_loc1<norm_loc2
            naux = norm_loc2; norm_loc2=norm_loc1; norm_loc1 = naux;
            stringlocaux=sloc2char; sloc2char=sloc1char;
            sloc1char=stringlocaux;
        end
        if norm_loc2<norm_loc3
            naux = norm_loc3; norm_loc3=norm_loc2; norm_loc2 = naux;
            stringlocaux=sloc3char; sloc3char=sloc2char;
            sloc2char=stringlocaux;
        end
        if norm_loc3<norm_loc4
            naux = norm_loc4; norm_loc4=norm_loc3; norm_loc3 = naux;
            stringlocaux=sloc4char; sloc4char=sloc3char;
            sloc3char=stringlocaux;
        end
        if norm_loc4<norm_loc5
```

```
            naux = norm_loc5; norm_loc5=norm_loc4; norm_loc4 = naux;
            stringlocaux=sloc5char; sloc5char=sloc4char;
            sloc4char=stringlocaux;
        end
        if norm_loc5<norm_loc6
            naux = norm_loc6; norm_loc6=norm_loc5; norm_loc5 = naux;
            stringlocaux=sloc6char; sloc6char=sloc5char;
            sloc5char=stringlocaux;
        end
        if norm_loc6<norm_loc7
            naux = norm_loc7; norm_loc7=norm_loc6; norm_loc6 = naux;
            stringlocaux=sloc7char; sloc7char=sloc6char;
            sloc6char=stringlocaux;
        end
        if norm_loc7<norm_loc8
            naux = norm_loc8; norm_loc8=norm_loc7; norm_loc7 = naux;
            stringlocaux=sloc8char; sloc8char=sloc7char;
            sloc7char=stringlocaux;
        end
        if norm_loc8<norm_loc9
            naux = norm_loc9; norm_loc9=norm_loc8; norm_loc8 = naux;
            stringlocaux=sloc9char; sloc9char=sloc8char;
            sloc8char=stringlocaux;
        end
        if norm_loc9<norm_loc10
            naux = norm_loc10; norm_loc10=norm_loc9; norm_loc9 = naux;
            stringlocaux=sloc10char; sloc10char=sloc9char;
            sloc9char=stringlocaux;
        end
        if norm_loc10<norm_loc11
            naux = norm_loc11; norm_loc11=norm_loc10; norm_loc10 = naux;
            stringlocaux=sloc11char; sloc11char=sloc10char;
            sloc10char=stringlocaux;
        end
        i=i+1;
    end
%montagem dos vetores com string ordenada dos locutores
vetor_stringsort = {sloc1char};
vetor_stringsort1 = {sloc11char;sloc10char;sloc9char;...
                     sloc8char;sloc7char;sloc6char;sloc5char;sloc4char;...
                     sloc3char;sloc2char};
%envia string de elocutores ordenados para tela
set(handles.textclassificacao,'string',vetor_stringsort1);
set(handles.textclassificacao2,'string',vetor_stringsort);
%indicar locutor pretenso no visor
set(handles.textoident,'string',sloc11char);
end
```

```
%--------ROTINA PARA ORDENACAO DOS 12 LOCUTORES PELOS DE MENOR NORMA-------
if linha==12
    for i=1:11
        if norm_loc1<norm_loc2
            naux = norm_loc2; norm_loc2=norm_loc1; norm_loc1 = naux;
            stringlocaux=sloc2char; sloc2char=sloc1char;
            sloc1char=stringlocaux;
        end
        if norm_loc2<norm_loc3
            naux = norm_loc3; norm_loc3=norm_loc2; norm_loc2 = naux;
            stringlocaux=sloc3char; sloc3char=sloc2char;
            sloc2char=stringlocaux;
        end
        if norm_loc3<norm_loc4
            naux = norm_loc4; norm_loc4=norm_loc3; norm_loc3 = naux;
            stringlocaux=sloc4char; sloc4char=sloc3char;
            sloc3char=stringlocaux;
        end
        if norm_loc4<norm_loc5
            naux = norm_loc5; norm_loc5=norm_loc4; norm_loc4 = naux;
            stringlocaux=sloc5char; sloc5char=sloc4char;
            sloc4char=stringlocaux;
        end
        if norm_loc5<norm_loc6
            naux = norm_loc6; norm_loc6=norm_loc5; norm_loc5 = naux;
            stringlocaux=sloc6char; sloc6char=sloc5char;
            sloc5char=stringlocaux;
        end
        if norm_loc6<norm_loc7
            naux = norm_loc7; norm_loc7=norm_loc6; norm_loc6 = naux;
            stringlocaux=sloc7char; sloc7char=sloc6char;
            sloc6char=stringlocaux;
        end
        if norm_loc7<norm_loc8
            naux = norm_loc8; norm_loc8=norm_loc7; norm_loc7 = naux;
            stringlocaux=sloc8char; sloc8char=sloc7char;
            sloc7char=stringlocaux;
        end
        if norm_loc8<norm_loc9
            naux = norm_loc9; norm_loc9=norm_loc8; norm_loc8 = naux;
            stringlocaux=sloc9char; sloc9char=sloc8char;
            sloc8char=stringlocaux;
        end
        if norm_loc9<norm_loc10
            naux = norm_loc10; norm_loc10=norm_loc9; norm_loc9 = naux;
            stringlocaux=sloc10char; sloc10char=sloc9char;
            sloc9char=stringlocaux;
```

```
        end
        if norm_loc10<norm_loc11
            naux = norm_loc11; norm_loc11=norm_loc10; norm_loc10 = naux;
            stringlocaux=sloc11char; sloc11char=sloc10char;
            sloc10char=stringlocaux;
        end
        if norm_loc11<norm_loc12
           naux = norm_loc12; norm_loc12=norm_loc11; norm_loc11 = naux;
           stringlocaux=sloc12char; sloc12char=sloc11char;
           sloc11char=stringlocaux;
        end
        i=i+1;
    end
%montagem dos vetores com string ordenada dos locutores
vetor_stringsort = {sloc2char;sloc1char};
vetor_stringsort1 = {sloc12char;sloc11char;sloc10char;sloc9char;...
                     sloc8char;sloc7char;sloc6char;sloc5char;sloc4char;...
                     sloc3char};
%envia string de elocutores ordenados para tela
set(handles.textclassificacao,'string',vetor_stringsort1);
set(handles.textclassificacao2,'string',vetor_stringsort);
%indicar locutor pretenso no visor
set(handles.textoident,'string',sloc12char);
end
%-------ROTINA PARA ORDENACAO DOS 13 LOCUTORES PELOS DE MENOR NORMA--------
if linha==13
    for i=1:12
        if norm_loc1<norm_loc2
            naux = norm_loc2; norm_loc2=norm_loc1; norm_loc1 = naux;
            stringlocaux=sloc2char; sloc2char=sloc1char;
            sloc1char=stringlocaux;
        end
        if norm_loc2<norm_loc3
            naux = norm_loc3; norm_loc3=norm_loc2; norm_loc2 = naux;
            stringlocaux=sloc3char; sloc3char=sloc2char;
            sloc2char=stringlocaux;
        end
        if norm_loc3<norm_loc4
            naux = norm_loc4; norm_loc4=norm_loc3; norm_loc3 = naux;
            stringlocaux=sloc4char; sloc4char=sloc3char;
            sloc3char=stringlocaux;
        end
        if norm_loc4<norm_loc5
            naux = norm_loc5; norm_loc5=norm_loc4; norm_loc4 = naux;
            stringlocaux=sloc5char; sloc5char=sloc4char;
            sloc4char=stringlocaux;
        end
```

```
        if norm_loc5<norm_loc6
            naux = norm_loc6; norm_loc6=norm_loc5; norm_loc5 = naux;
            stringlocaux=sloc6char; sloc6char=sloc5char;
            sloc5char=stringlocaux;
        end
        if norm_loc6<norm_loc7
            naux = norm_loc7; norm_loc7=norm_loc6; norm_loc6 = naux;
            stringlocaux=sloc7char; sloc7char=sloc6char;
            sloc6char=stringlocaux;
        end
        if norm_loc7<norm_loc8
            naux = norm_loc8; norm_loc8=norm_loc7; norm_loc7 = naux;
            stringlocaux=sloc8char; sloc8char=sloc7char;
            sloc7char=stringlocaux;
        end
        if norm_loc8<norm_loc9
            naux = norm_loc9; norm_loc9=norm_loc8; norm_loc8 = naux;
            stringlocaux=sloc9char; sloc9char=sloc8char;
            sloc8char=stringlocaux;
        end
        if norm_loc9<norm_loc10
            naux = norm_loc10; norm_loc10=norm_loc9; norm_loc9 = naux;
            stringlocaux=sloc10char; sloc10char=sloc9char;
            sloc9char=stringlocaux;
        end
        if norm_loc10<norm_loc11
            naux = norm_loc11; norm_loc11=norm_loc10; norm_loc10 = naux;
            stringlocaux=sloc11char; sloc11char=sloc10char;
            sloc10char=stringlocaux;
        end
        if norm_loc11<norm_loc12
           naux = norm_loc12; norm_loc12=norm_loc11; norm_loc11 = naux;
           stringlocaux=sloc12char; sloc12char=sloc11char;
           sloc11char=stringlocaux;
        end
        if norm_loc12<norm_loc13
            naux = norm_loc13; norm_loc13=norm_loc12; norm_loc12 = naux;
            stringlocaux=sloc13char; sloc13char=sloc12char;
            sloc12char=stringlocaux;
        end
        i=i+1;
    end
%montagem dos vetores com string ordenada dos locutores
vetor_stringsort = {sloc3char;sloc2char;sloc1char};
vetor_stringsort1 = {sloc13char;sloc12char;sloc11char;...
                     sloc10char;sloc9char;sloc8char;sloc7char;...
                     sloc6char;sloc5char;sloc4char};
```

```
%envia string de elocutores ordenados para tela
set(handles.textclassificacao,'string',vetor_stringsort1);
set(handles.textclassificacao2,'string',vetor_stringsort);
%indicar locutor pretenso no visor
set(handles.textoident,'string',sloc13char);
end
%------ROTINA PARA ORDENACAO DOS 14 LOCUTORES PELOS DE MENOR NORMA---------
if linha==14
    for i=1:13
        if norm_loc1<norm_loc2
            naux = norm_loc2; norm_loc2=norm_loc1; norm_loc1 = naux;
            stringlocaux=sloc2char; sloc2char=sloc1char;
            sloc1char=stringlocaux;
        end
        if norm_loc2<norm_loc3
            naux = norm_loc3; norm_loc3=norm_loc2; norm_loc2 = naux;
            stringlocaux=sloc3char; sloc3char=sloc2char;
            sloc2char=stringlocaux;
        end
        if norm_loc3<norm_loc4
            naux = norm_loc4; norm_loc4=norm_loc3; norm_loc3 = naux;
            stringlocaux=sloc4char; sloc4char=sloc3char;
            sloc3char=stringlocaux;
        end
        if norm_loc4<norm_loc5
            naux = norm_loc5; norm_loc5=norm_loc4; norm_loc4 = naux;
            stringlocaux=sloc5char; sloc5char=sloc4char;
            sloc4char=stringlocaux;
        end
        if norm_loc5<norm_loc6
            naux = norm_loc6; norm_loc6=norm_loc5; norm_loc5 = naux;
            stringlocaux=sloc6char; sloc6char=sloc5char;
            sloc5char=stringlocaux;
        end
        if norm_loc6<norm_loc7
            naux = norm_loc7; norm_loc7=norm_loc6; norm_loc6 = naux;
            stringlocaux=sloc7char; sloc7char=sloc6char;
            sloc6char=stringlocaux;
        end
        if norm_loc7<norm_loc8
            naux = norm_loc8; norm_loc8=norm_loc7; norm_loc7 = naux;
            stringlocaux=sloc8char; sloc8char=sloc7char;
            sloc7char=stringlocaux;
        end
        if norm_loc8<norm_loc9
            naux = norm_loc9; norm_loc9=norm_loc8; norm_loc8 = naux;
            stringlocaux=sloc9char; sloc9char=sloc8char;
```

```
            sloc8char=stringlocaux;
        end
        if norm_loc9<norm_loc10
            naux = norm_loc10; norm_loc10=norm_loc9; norm_loc9 = naux;
            stringlocaux=sloc10char; sloc10char=sloc9char;
            sloc9char=stringlocaux;
        end
        if norm_loc10<norm_loc11
            naux = norm_loc11; norm_loc11=norm_loc10; norm_loc10 = naux;
            stringlocaux=sloc11char; sloc11char=sloc10char;
            sloc10char=stringlocaux;
        end
        if norm_loc11<norm_loc12
           naux = norm_loc12; norm_loc12=norm_loc11; norm_loc11 = naux;
           stringlocaux=sloc12char; sloc12char=sloc11char;
           sloc11char=stringlocaux;
        end
        if norm_loc12<norm_loc13
            naux = norm_loc13; norm_loc13=norm_loc12; norm_loc12 = naux;
            stringlocaux=sloc13char; sloc13char=sloc12char;
            sloc12char=stringlocaux;
        end
        if norm_loc13<norm_loc14
            naux = norm_loc14; norm_loc14=norm_loc13; norm_loc13 = naux;
            stringlocaux=sloc14char; sloc14char=sloc13char;
            sloc13char=stringlocaux;
        end
        i=i+1;
    end
%montagem dos vetores com string ordenada dos locutores
vetor_stringsort = {sloc4char;sloc3char;sloc2char;sloc1char};
vetor_stringsort1 = {sloc14char;sloc13char;sloc12char;sloc11char;...
                     sloc10char;sloc9char;sloc8char;sloc7char;...
                     sloc6char;sloc5char};
%envia string de elocutores ordenados para tela
set(handles.textclassificacao,'string',vetor_stringsort1);
set(handles.textclassificacao2,'string',vetor_stringsort);
%indicar locutor pretenso no visor
set(handles.textoident,'string',sloc14char);
end
%---------ROTINA PARA ORDENACAO DOS 15 LOCUTORES PELOS DE MENOR NORMA------
if linha==15
    for i=1:14
        if norm_loc1<norm_loc2
            naux = norm_loc2; norm_loc2=norm_loc1; norm_loc1 = naux;
            stringlocaux=sloc2char; sloc2char=sloc1char;
            sloc1char=stringlocaux;
```

```
        end
        if norm_loc2<norm_loc3
            naux = norm_loc3; norm_loc3=norm_loc2; norm_loc2 = naux;
            stringlocaux=sloc3char; sloc3char=sloc2char;
            sloc2char=stringlocaux;
        end
        if norm_loc3<norm_loc4
            naux = norm_loc4; norm_loc4=norm_loc3; norm_loc3 = naux;
            stringlocaux=sloc4char; sloc4char=sloc3char;
            sloc3char=stringlocaux;
        end
        if norm_loc4<norm_loc5
            naux = norm_loc5; norm_loc5=norm_loc4; norm_loc4 = naux;
            stringlocaux=sloc5char; sloc5char=sloc4char;
            sloc4char=stringlocaux;
        end
        if norm_loc5<norm_loc6
            naux = norm_loc6; norm_loc6=norm_loc5; norm_loc5 = naux;
            stringlocaux=sloc6char; sloc6char=sloc5char;
            sloc5char=stringlocaux;
        end
        if norm_loc6<norm_loc7
            naux = norm_loc7; norm_loc7=norm_loc6; norm_loc6 = naux;
            stringlocaux=sloc7char; sloc7char=sloc6char;
            sloc6char=stringlocaux;
        end
        if norm_loc7<norm_loc8
            naux = norm_loc8; norm_loc8=norm_loc7; norm_loc7 = naux;
            stringlocaux=sloc8char; sloc8char=sloc7char;
            sloc7char=stringlocaux;
        end
        if norm_loc8<norm_loc9
            naux = norm_loc9; norm_loc9=norm_loc8; norm_loc8 = naux;
            stringlocaux=sloc9char; sloc9char=sloc8char;
            sloc8char=stringlocaux;
        end
        if norm_loc9<norm_loc10
            naux = norm_loc10; norm_loc10=norm_loc9; norm_loc9 = naux;
            stringlocaux=sloc10char; sloc10char=sloc9char;
            sloc9char=stringlocaux;
        end
        if norm_loc10<norm_loc11
            naux = norm_loc11; norm_loc11=norm_loc10; norm_loc10 = naux;
            stringlocaux=sloc11char; sloc11char=sloc10char;
            sloc10char=stringlocaux;
        end
        if norm_loc11<norm_loc12
```

```
            naux = norm_loc12; norm_loc12=norm_loc11; norm_loc11 = naux;
            stringlocaux=sloc12char; sloc12char=sloc11char;
            sloc11char=stringlocaux;
        end
        if norm_loc12<norm_loc13
            naux = norm_loc13; norm_loc13=norm_loc12; norm_loc12 = naux;
            stringlocaux=sloc13char; sloc13char=sloc12char;
            sloc12char=stringlocaux;
        end
        if norm_loc13<norm_loc14
            naux = norm_loc14; norm_loc14=norm_loc13; norm_loc13 = naux;
            stringlocaux=sloc14char; sloc14char=sloc13char;
            sloc13char=stringlocaux;
        end
        if norm_loc14<norm_loc15
            naux = norm_loc15; norm_loc15=norm_loc14; norm_loc14 = naux;
            stringlocaux=sloc15char; sloc15char=sloc14char;
            sloc14char=stringlocaux;
        end
        i=i+1;
    end
%montagem dos vetores com string ordenada dos locutores
vetor_stringsort = {sloc5char;sloc4char;sloc3char;...
                    sloc2char;sloc1char};
vetor_stringsort1 = {sloc15char;sloc14char;...
                     sloc13char;sloc12char;sloc11char;sloc10char;...
                     sloc9char;sloc8char;sloc7char;sloc6char};
%envia string de elocutores ordenados para tela
set(handles.textclassificacao,'string',vetor_stringsort1);
set(handles.textclassificacao2,'string',vetor_stringsort);
%indicar locutor pretenso no visor
set(handles.textoident,'string',sloc15char);
end
%---------ROTINA PARA ORDENACAO DOS 16 LOCUTORES PELOS DE MENOR NORMA------
if linha==16
    for i=1:15
        if norm_loc1<norm_loc2
            naux = norm_loc2; norm_loc2=norm_loc1; norm_loc1 = naux;
            stringlocaux=sloc2char; sloc2char=sloc1char;
            sloc1char=stringlocaux;
        end
        if norm_loc2<norm_loc3
            naux = norm_loc3; norm_loc3=norm_loc2; norm_loc2 = naux;
            stringlocaux=sloc3char; sloc3char=sloc2char;
            sloc2char=stringlocaux;
        end
        if norm_loc3<norm_loc4
```

```
        naux = norm_loc4; norm_loc4=norm_loc3; norm_loc3 = naux;
        stringlocaux=sloc4char; sloc4char=sloc3char;
        sloc3char=stringlocaux;
    end
    if norm_loc4<norm_loc5
        naux = norm_loc5; norm_loc5=norm_loc4; norm_loc4 = naux;
        stringlocaux=sloc5char; sloc5char=sloc4char;
        sloc4char=stringlocaux;
    end
    if norm_loc5<norm_loc6
        naux = norm_loc6; norm_loc6=norm_loc5; norm_loc5 = naux;
        stringlocaux=sloc6char; sloc6char=sloc5char;
        sloc5char=stringlocaux;
    end
    if norm_loc6<norm_loc7
        naux = norm_loc7; norm_loc7=norm_loc6; norm_loc6 = naux;
        stringlocaux=sloc7char; sloc7char=sloc6char;
        sloc6char=stringlocaux;
    end
    if norm_loc7<norm_loc8
        naux = norm_loc8; norm_loc8=norm_loc7; norm_loc7 = naux;
        stringlocaux=sloc8char; sloc8char=sloc7char;
        sloc7char=stringlocaux;
    end
    if norm_loc8<norm_loc9
        naux = norm_loc9; norm_loc9=norm_loc8; norm_loc8 = naux;
        stringlocaux=sloc9char; sloc9char=sloc8char;
        sloc8char=stringlocaux;
    end
    if norm_loc9<norm_loc10
        naux = norm_loc10; norm_loc10=norm_loc9; norm_loc9 = naux;
        stringlocaux=sloc10char; sloc10char=sloc9char;
        sloc9char=stringlocaux;
    end
    if norm_loc10<norm_loc11
        naux = norm_loc11; norm_loc11=norm_loc10; norm_loc10 = naux;
        stringlocaux=sloc11char; sloc11char=sloc10char;
        sloc10char=stringlocaux;
    end
    if norm_loc11<norm_loc12
       naux = norm_loc12; norm_loc12=norm_loc11; norm_loc11 = naux;
       stringlocaux=sloc12char; sloc12char=sloc11char;
       sloc11char=stringlocaux;
    end
    if norm_loc12<norm_loc13
        naux = norm_loc13; norm_loc13=norm_loc12; norm_loc12 = naux;
        stringlocaux=sloc13char; sloc13char=sloc12char;
```

```
            sloc12char=stringlocaux;
        end
        if norm_loc13<norm_loc14
            naux = norm_loc14; norm_loc14=norm_loc13; norm_loc13 = naux;
            stringlocaux=sloc14char; sloc14char=sloc13char;
            sloc13char=stringlocaux;
        end
        if norm_loc14<norm_loc15
            naux = norm_loc15; norm_loc15=norm_loc14; norm_loc14 = naux;
            stringlocaux=sloc15char; sloc15char=sloc14char;
            sloc14char=stringlocaux;
        end
        if norm_loc15<norm_loc16
            naux = norm_loc16; norm_loc16=norm_loc15; norm_loc15 = naux;
            stringlocaux=sloc16char; sloc16char=sloc15char;
            sloc15char=stringlocaux;
        end
        i=i+1;
    end
%montagem dos vetores com string ordenada dos locutores
vetor_stringsort = {sloc6char;sloc5char;sloc4char;sloc3char;...
                    sloc2char;sloc1char};
vetor_stringsort1 = {sloc16char;sloc15char;sloc14char;...
                     sloc13char;sloc12char;sloc11char;sloc10char;...
                     sloc9char;sloc8char;sloc7char};
%envia string de elocutores ordenados para tela
set(handles.textclassificacao,'string',vetor_stringsort1);
set(handles.textclassificacao2,'string',vetor_stringsort);
%indicar locutor pretenso no visor
set(handles.textoident,'string',sloc16char);
end
%---------ROTINA PARA ORDENACAO DOS 17 LOCUTORES PELOS DE MENOR NORMA------
if linha==17
    for i=1:16
        if norm_loc1<norm_loc2
            naux = norm_loc2; norm_loc2=norm_loc1; norm_loc1 = naux;
            stringlocaux=sloc2char; sloc2char=sloc1char;
            sloc1char=stringlocaux;
        end
        if norm_loc2<norm_loc3
            naux = norm_loc3; norm_loc3=norm_loc2; norm_loc2 = naux;
            stringlocaux=sloc3char; sloc3char=sloc2char;
            sloc2char=stringlocaux;
        end
        if norm_loc3<norm_loc4
            naux = norm_loc4; norm_loc4=norm_loc3; norm_loc3 = naux;
            stringlocaux=sloc4char; sloc4char=sloc3char;
```

```
        sloc3char=stringlocaux;
    end
    if norm_loc4<norm_loc5
        naux = norm_loc5; norm_loc5=norm_loc4; norm_loc4 = naux;
        stringlocaux=sloc5char; sloc5char=sloc4char;
        sloc4char=stringlocaux;
    end
    if norm_loc5<norm_loc6
        naux = norm_loc6; norm_loc6=norm_loc5; norm_loc5 = naux;
        stringlocaux=sloc6char; sloc6char=sloc5char;
        sloc5char=stringlocaux;
    end
    if norm_loc6<norm_loc7
        naux = norm_loc7; norm_loc7=norm_loc6; norm_loc6 = naux;
        stringlocaux=sloc7char; sloc7char=sloc6char;
        sloc6char=stringlocaux;
    end
    if norm_loc7<norm_loc8
        naux = norm_loc8; norm_loc8=norm_loc7; norm_loc7 = naux;
        stringlocaux=sloc8char; sloc8char=sloc7char;
        sloc7char=stringlocaux;
    end
    if norm_loc8<norm_loc9
        naux = norm_loc9; norm_loc9=norm_loc8; norm_loc8 = naux;
        stringlocaux=sloc9char; sloc9char=sloc8char;
        sloc8char=stringlocaux;
    end
    if norm_loc9<norm_loc10
        naux = norm_loc10; norm_loc10=norm_loc9; norm_loc9 = naux;
        stringlocaux=sloc10char; sloc10char=sloc9char;
        sloc9char=stringlocaux;
    end
    if norm_loc10<norm_loc11
        naux = norm_loc11; norm_loc11=norm_loc10; norm_loc10 = naux;
        stringlocaux=sloc11char; sloc11char=sloc10char;
        sloc10char=stringlocaux;
    end
    if norm_loc11<norm_loc12
       naux = norm_loc12; norm_loc12=norm_loc11; norm_loc11 = naux;
       stringlocaux=sloc12char; sloc12char=sloc11char;
       sloc11char=stringlocaux;
    end
    if norm_loc12<norm_loc13
        naux = norm_loc13; norm_loc13=norm_loc12; norm_loc12 = naux;
        stringlocaux=sloc13char; sloc13char=sloc12char;
        sloc12char=stringlocaux;
    end
```

```
        if norm_loc13<norm_loc14
            naux = norm_loc14; norm_loc14=norm_loc13; norm_loc13 = naux;
            stringlocaux=sloc14char; sloc14char=sloc13char;
            sloc13char=stringlocaux;
        end
        if norm_loc14<norm_loc15
            naux = norm_loc15; norm_loc15=norm_loc14; norm_loc14 = naux;
            stringlocaux=sloc15char; sloc15char=sloc14char;
            sloc14char=stringlocaux;
        end
        if norm_loc15<norm_loc16
            naux = norm_loc16; norm_loc16=norm_loc15; norm_loc15 = naux;
            stringlocaux=sloc16char; sloc16char=sloc15char;
            sloc15char=stringlocaux;
        end
        if norm_loc16<norm_loc17
            naux = norm_loc17; norm_loc17=norm_loc16; norm_loc16 = naux;
            stringlocaux=sloc17char; sloc17char=sloc16char;
            sloc16char=stringlocaux;
        end
        i=i+1;
    end
%montagem dos vetores com string ordenada dos locutores
vetor_stringsort = {sloc7char;sloc6char;...
                    sloc5char;sloc4char;sloc3char;sloc2char;sloc1char};
vetor_stringsort1 = {sloc17char;sloc16char;sloc15char;sloc14char;...
                     sloc13char;sloc12char;sloc11char;sloc10char;...
                     sloc9char;sloc8char};
%envia string de elocutores ordenados para tela
set(handles.textclassificacao,'string',vetor_stringsort1);
set(handles.textclassificacao2,'string',vetor_stringsort);
%indicar locutor pretenso no visor
set(handles.textoident,'string',sloc17char);
end
%---------ROTINA PARA ORDENACAO DOS 18 LOCUTORES PELOS DE MENOR NORMA------
if linha==18
    for i=1:17
        if norm_loc1<norm_loc2
            naux = norm_loc2; norm_loc2=norm_loc1; norm_loc1 = naux;
            stringlocaux=sloc2char; sloc2char=sloc1char;
            sloc1char=stringlocaux;
        end
        if norm_loc2<norm_loc3
            naux = norm_loc3; norm_loc3=norm_loc2; norm_loc2 = naux;
            stringlocaux=sloc3char; sloc3char=sloc2char;
            sloc2char=stringlocaux;
        end
```

```
        if norm_loc3<norm_loc4
            naux = norm_loc4; norm_loc4=norm_loc3; norm_loc3 = naux;
            stringlocaux=sloc4char; sloc4char=sloc3char;
            sloc3char=stringlocaux;
        end
        if norm_loc4<norm_loc5
            naux = norm_loc5; norm_loc5=norm_loc4; norm_loc4 = naux;
            stringlocaux=sloc5char; sloc5char=sloc4char;
            sloc4char=stringlocaux;
        end
        if norm_loc5<norm_loc6
            naux = norm_loc6; norm_loc6=norm_loc5; norm_loc5 = naux;
            stringlocaux=sloc6char; sloc6char=sloc5char;
            sloc5char=stringlocaux;
        end
        if norm_loc6<norm_loc7
            naux = norm_loc7; norm_loc7=norm_loc6; norm_loc6 = naux;
            stringlocaux=sloc7char; sloc7char=sloc6char;
            sloc6char=stringlocaux;
        end
        if norm_loc7<norm_loc8
            naux = norm_loc8; norm_loc8=norm_loc7; norm_loc7 = naux;
            stringlocaux=sloc8char; sloc8char=sloc7char;
            sloc7char=stringlocaux;
        end
        if norm_loc8<norm_loc9
            naux = norm_loc9; norm_loc9=norm_loc8; norm_loc8 = naux;
            stringlocaux=sloc9char; sloc9char=sloc8char;
            sloc8char=stringlocaux;
        end
        if norm_loc9<norm_loc10
            naux = norm_loc10; norm_loc10=norm_loc9; norm_loc9 = naux;
            stringlocaux=sloc10char; sloc10char=sloc9char;
            sloc9char=stringlocaux;
        end
        if norm_loc10<norm_loc11
            naux = norm_loc11; norm_loc11=norm_loc10; norm_loc10 = naux;
            stringlocaux=sloc11char; sloc11char=sloc10char;
            sloc10char=stringlocaux;
        end
        if norm_loc11<norm_loc12
           naux = norm_loc12; norm_loc12=norm_loc11; norm_loc11 = naux;
           stringlocaux=sloc12char; sloc12char=sloc11char;
           sloc11char=stringlocaux;
        end
        if norm_loc12<norm_loc13
            naux = norm_loc13; norm_loc13=norm_loc12; norm_loc12 = naux;
```

```
            stringlocaux=sloc13char; sloc13char=sloc12char;
            sloc12char=stringlocaux;
        end
        if norm_loc13<norm_loc14
            naux = norm_loc14; norm_loc14=norm_loc13; norm_loc13 = naux;
            stringlocaux=sloc14char; sloc14char=sloc13char;
            sloc13char=stringlocaux;
        end
        if norm_loc14<norm_loc15
            naux = norm_loc15; norm_loc15=norm_loc14; norm_loc14 = naux;
            stringlocaux=sloc15char; sloc15char=sloc14char;
            sloc14char=stringlocaux;
        end
        if norm_loc15<norm_loc16
            naux = norm_loc16; norm_loc16=norm_loc15; norm_loc15 = naux;
            stringlocaux=sloc16char; sloc16char=sloc15char;
            sloc15char=stringlocaux;
        end
        if norm_loc16<norm_loc17
            naux = norm_loc17; norm_loc17=norm_loc16; norm_loc16 = naux;
            stringlocaux=sloc17char; sloc17char=sloc16char;
            sloc16char=stringlocaux;
        end
        if norm_loc17<norm_loc18
            naux = norm_loc18; norm_loc18=norm_loc17; norm_loc17 = naux;
            stringlocaux=sloc18char; sloc18char=sloc17char;
            sloc17char=stringlocaux;
        end
        i=i+1;
    end
%montagem dos vetores com string ordenada dos locutores
vetor_stringsort = {sloc8char;sloc7char;sloc6char;...
                    sloc5char;sloc4char;sloc3char;sloc2char;sloc1char};
vetor_stringsort1 = {sloc18char;sloc17char;sloc16char;...
                     sloc15char;sloc14char;sloc13char;sloc12char;...
                     sloc11char;sloc10char;sloc9char};
%envia string de elocutores ordenados para tela
set(handles.textclassificacao,'string',vetor_stringsort1);
set(handles.textclassificacao2,'string',vetor_stringsort);
%indicar locutor pretenso no visor
set(handles.textoident,'string',sloc18char);
end
%---------ROTINA PARA ORDENACAO DOS 19 LOCUTORES PELOS DE MENOR NORMA------
if linha==19
    for i=1:18
        if norm_loc1<norm_loc2
            naux = norm_loc2; norm_loc2=norm_loc1; norm_loc1 = naux;
```

```
        stringlocaux=sloc2char; sloc2char=sloc1char;
        sloc1char=stringlocaux;
    end
    if norm_loc2<norm_loc3
        naux = norm_loc3; norm_loc3=norm_loc2; norm_loc2 = naux;
        stringlocaux=sloc3char; sloc3char=sloc2char;
        sloc2char=stringlocaux;
    end
    if norm_loc3<norm_loc4
        naux = norm_loc4; norm_loc4=norm_loc3; norm_loc3 = naux;
        stringlocaux=sloc4char; sloc4char=sloc3char;
        sloc3char=stringlocaux;
    end
    if norm_loc4<norm_loc5
        naux = norm_loc5; norm_loc5=norm_loc4; norm_loc4 = naux;
        stringlocaux=sloc5char; sloc5char=sloc4char;
        sloc4char=stringlocaux;
    end
    if norm_loc5<norm_loc6
        naux = norm_loc6; norm_loc6=norm_loc5; norm_loc5 = naux;
        stringlocaux=sloc6char; sloc6char=sloc5char;
        sloc5char=stringlocaux;
    end
    if norm_loc6<norm_loc7
        naux = norm_loc7; norm_loc7=norm_loc6; norm_loc6 = naux;
        stringlocaux=sloc7char; sloc7char=sloc6char;
        sloc6char=stringlocaux;
    end
    if norm_loc7<norm_loc8
        naux = norm_loc8; norm_loc8=norm_loc7; norm_loc7 = naux;
        stringlocaux=sloc8char; sloc8char=sloc7char;
        sloc7char=stringlocaux;
    end
    if norm_loc8<norm_loc9
        naux = norm_loc9; norm_loc9=norm_loc8; norm_loc8 = naux;
        stringlocaux=sloc9char; sloc9char=sloc8char;
        sloc8char=stringlocaux;
    end
    if norm_loc9<norm_loc10
        naux = norm_loc10; norm_loc10=norm_loc9; norm_loc9 = naux;
        stringlocaux=sloc10char; sloc10char=sloc9char;
        sloc9char=stringlocaux;
    end
    if norm_loc10<norm_loc11
        naux = norm_loc11; norm_loc11=norm_loc10; norm_loc10 = naux;
        stringlocaux=sloc11char; sloc11char=sloc10char;
        sloc10char=stringlocaux;
```

```
        end
        if norm_loc11<norm_loc12
           naux = norm_loc12; norm_loc12=norm_loc11; norm_loc11 = naux;
           stringlocaux=sloc12char; sloc12char=sloc11char;
           sloc11char=stringlocaux;
        end
        if norm_loc12<norm_loc13
            naux = norm_loc13; norm_loc13=norm_loc12; norm_loc12 = naux;
            stringlocaux=sloc13char; sloc13char=sloc12char;
            sloc12char=stringlocaux;
        end
        if norm_loc13<norm_loc14
            naux = norm_loc14; norm_loc14=norm_loc13; norm_loc13 = naux;
            stringlocaux=sloc14char; sloc14char=sloc13char;
            sloc13char=stringlocaux;
        end
        if norm_loc14<norm_loc15
            naux = norm_loc15; norm_loc15=norm_loc14; norm_loc14 = naux;
            stringlocaux=sloc15char; sloc15char=sloc14char;
            sloc14char=stringlocaux;
        end
        if norm_loc15<norm_loc16
            naux = norm_loc16; norm_loc16=norm_loc15; norm_loc15 = naux;
            stringlocaux=sloc16char; sloc16char=sloc15char;
            sloc15char=stringlocaux;
        end
        if norm_loc16<norm_loc17
            naux = norm_loc17; norm_loc17=norm_loc16; norm_loc16 = naux;
            stringlocaux=sloc17char; sloc17char=sloc16char;
            sloc16char=stringlocaux;
        end
        if norm_loc17<norm_loc18
            naux = norm_loc18; norm_loc18=norm_loc17; norm_loc17 = naux;
            stringlocaux=sloc18char; sloc18char=sloc17char;
            sloc17char=stringlocaux;
        end
        if norm_loc18<norm_loc19
            naux = norm_loc19; norm_loc19=norm_loc18; norm_loc18 = naux;
            stringlocaux=sloc19char; sloc19char=sloc18char;
            sloc18char=stringlocaux;
        end
        i=i+1;
    end
%montagem dos vetores com string ordenada dos locutores
vetor_stringsort = {sloc9char;sloc8char;sloc7char;sloc6char;...
                    sloc5char;sloc4char;sloc3char;sloc2char;sloc1char};
vetor_stringsort1 = {sloc19char;sloc18char;sloc17char;sloc16char;...
```

```
                    sloc15char;sloc14char;sloc13char;sloc12char;...
                    sloc11char;sloc10char};
%envia string de elocutores ordenados para tela
set(handles.textclassificacao,'string',vetor_stringsort1);
set(handles.textclassificacao2,'string',vetor_stringsort);
% indicar locutor pretenso do visor
set(handles.textoident,'string',sloc19char);
end
%---------ROTINA PARA ORDENACAO DOS 20 LOCUTORES PELOS DE MENOR NORMA------
if linha==20
%bloco de ordenacao dos vetores strings
    for i=1:19
        if norm_loc1<norm_loc2
            naux = norm_loc2; norm_loc2=norm_loc1; norm_loc1 = naux;
            stringlocaux=sloc2char; sloc2char=sloc1char;
            sloc1char=stringlocaux;
        end
        if norm_loc2<norm_loc3
            naux = norm_loc3; norm_loc3=norm_loc2; norm_loc2 = naux;
            stringlocaux=sloc3char; sloc3char=sloc2char;
            sloc2char=stringlocaux;
        end
        if norm_loc3<norm_loc4
            naux = norm_loc4; norm_loc4=norm_loc3; norm_loc3 = naux;
            stringlocaux=sloc4char; sloc4char=sloc3char;
            sloc3char=stringlocaux;
        end
        if norm_loc4<norm_loc5
            naux = norm_loc5; norm_loc5=norm_loc4; norm_loc4 = naux;
            stringlocaux=sloc5char; sloc5char=sloc4char;
            sloc4char=stringlocaux;
        end
        if norm_loc5<norm_loc6
            naux = norm_loc6; norm_loc6=norm_loc5; norm_loc5 = naux;
            stringlocaux=sloc6char; sloc6char=sloc5char;
            sloc5char=stringlocaux;
        end
        if norm_loc6<norm_loc7
            naux = norm_loc7; norm_loc7=norm_loc6; norm_loc6 = naux;
            stringlocaux=sloc7char; sloc7char=sloc6char;
            sloc6char=stringlocaux;
        end
        if norm_loc7<norm_loc8
            naux = norm_loc8; norm_loc8=norm_loc7; norm_loc7 = naux;
            stringlocaux=sloc8char; sloc8char=sloc7char;
            sloc7char=stringlocaux;
        end
```

```
        if norm_loc8<norm_loc9
            naux = norm_loc9; norm_loc9=norm_loc8; norm_loc8 = naux;
            stringlocaux=sloc9char; sloc9char=sloc8char;
            sloc8char=stringlocaux;
        end
        if norm_loc9<norm_loc10
            naux = norm_loc10; norm_loc10=norm_loc9; norm_loc9 = naux;
            stringlocaux=sloc10char; sloc10char=sloc9char;
            sloc9char=stringlocaux;
        end
        if norm_loc10<norm_loc11
            naux = norm_loc11; norm_loc11=norm_loc10; norm_loc10 = naux;
            stringlocaux=sloc11char; sloc11char=sloc10char;
            sloc10char=stringlocaux;
        end
        if norm_loc11<norm_loc12
           naux = norm_loc12; norm_loc12=norm_loc11; norm_loc11 = naux;
           stringlocaux=sloc12char; sloc12char=sloc11char;
           sloc11char=stringlocaux;
        end
        if norm_loc12<norm_loc13
            naux = norm_loc13; norm_loc13=norm_loc12; norm_loc12 = naux;
            stringlocaux=sloc13char; sloc13char=sloc12char;
            sloc12char=stringlocaux;
        end
        if norm_loc13<norm_loc14
            naux = norm_loc14; norm_loc14=norm_loc13; norm_loc13 = naux;
            stringlocaux=sloc14char; sloc14char=sloc13char;
            sloc13char=stringlocaux;
        end
        if norm_loc14<norm_loc15
            naux = norm_loc15; norm_loc15=norm_loc14; norm_loc14 = naux;
            stringlocaux=sloc15char; sloc15char=sloc14char;
            sloc14char=stringlocaux;
        end
        if norm_loc15<norm_loc16
            naux = norm_loc16; norm_loc16=norm_loc15; norm_loc15 = naux;
            stringlocaux=sloc16char; sloc16char=sloc15char;
            sloc15char=stringlocaux;
        end
        if norm_loc16<norm_loc17
            naux = norm_loc17; norm_loc17=norm_loc16; norm_loc16 = naux;
            stringlocaux=sloc17char; sloc17char=sloc16char;
            sloc16char=stringlocaux;
        end
        if norm_loc17<norm_loc18
            naux = norm_loc18; norm_loc18=norm_loc17; norm_loc17 = naux;
```

```
            stringlocaux=sloc18char; sloc18char=sloc17char;
            sloc17char=stringlocaux;
        end
        if norm_loc18<norm_loc19
            naux = norm_loc19; norm_loc19=norm_loc18; norm_loc18 = naux;
            stringlocaux=sloc19char; sloc19char=sloc18char;
            sloc18char=stringlocaux;
        end
        if norm_loc19<norm_loc20
            naux = norm_loc20; norm_loc20=norm_loc19; norm_loc19 = naux;
            stringlocaux=sloc20char; sloc20char=sloc19char;
            sloc19char=stringlocaux;
        end
        i=i+1;
    end
%montagem dos vetores com string ordenada dos locutores
vetor_stringsort = {sloc10char;sloc9char;sloc8char;sloc7char;sloc6char;...
                    sloc5char;sloc4char;sloc3char;sloc2char;sloc1char};
vetor_stringsort1 = {sloc20char;sloc19char;sloc18char;sloc17char;...
                     sloc16char;sloc15char;sloc14char;sloc13char;...
                     sloc12char;sloc11char};
%envia string de elocutores ordenados para tela
set(handles.textclassificacao,'string',vetor_stringsort1);
set(handles.textclassificacao2,'string',vetor_stringsort);
%indicacao do locutor pretenso
set(handles.textoident,'string',sloc20char);
end
```

PadraoLoc20.m

```
function varargout = PadraoLoc20(varargin)
% PADRAOLOC20   Software de geracao do padrao de voz de 20 usuarios atraves
%               de 5 elocucoes de treinamento como entrada. Produz na saida
%               um arquito txt contendo os padroes de voz de todos os usua-
%               usuarios cadastrados.
%
%Autores:       Roberto F. B. Sotero Filho e Helio Magalhaes de Oliveira
%E-mail:        rsotero@hotmail.com e hmo@ufpe.br
%Universidade:  Universidade Federal de Pernambuco
%Data:          15/05/2017

% Codigo de inicializacao do programa - nao editavel
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @PadraoLoc20_OpeningFcn, ...
                   'gui_OutputFcn',  @PadraoLoc20_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% Chamada dos logos (CTG e DES) na tela.
function des_CreateFcn(hObject, eventdata, handles)
A = imread('figuras\logospadrao.png');
imshow(A);
% Chamada do logo UFPE na tela.
function ufpe_CreateFcn(hObject, eventdata, handles)
A = imread('figuras\ufpepadrao.png');
imshow(A);
% ------------ Rotinas e funcoes internas do programa --------------------
function PadraoLoc20_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;
guidata(hObject, handles);
function varargout = PadraoLoc20_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;
%campos de texto das elocucoes
function textoeloc1_1_Callback(hObject, eventdata, handles)
function textoeloc1_1_CreateFcn(hObject, eventdata, handles)
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),...
                   get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc1_2_Callback(hObject, eventdata, handles)
function textoeloc1_2_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
                   get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc1_3_Callback(hObject, eventdata, handles)
function textoeloc1_3_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
                   get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc1_4_Callback(hObject, eventdata, handles)
function textoeloc1_4_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
                   get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc1_5_Callback(hObject, eventdata, handles)
function textoeloc1_5_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
                   get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc3_1_Callback(hObject, eventdata, handles)
function textoeloc3_1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc3_2_Callback(hObject, eventdata, handles)
function textoeloc3_2_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),...
                   get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
function textoeloc3_3_Callback(hObject, eventdata, handles)
function textoeloc3_3_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),...
                   get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc3_4_Callback(hObject, eventdata, handles)
function textoeloc3_4_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),...
                   get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc3_5_Callback(hObject, eventdata, handles)
function textoeloc3_5_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),...
                   get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
%Acoes executadas ao pressionar botao selecionar (elocucao1)
function botaoeloc1_1_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
    'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc1_1,'string',arquivovoz11);
end
%Acoes executadas ao pressionar botao selecionar (elocucao2)
function botaoeloc1_2_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
    'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc1_2,'string',arquivovoz11);
end

function botaoeloc1_3_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
```

```
    'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc1_3,'string',arquivovoz11);
end

function botaoeloc1_4_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
    'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc1_4,'string',arquivovoz11);
end

function botaoeloc1_5_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
    'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc1_5,'string',arquivovoz11);
end

function botaoeloc3_1_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
    'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc3_1,'string',arquivovoz11);
end

function botaoeloc3_2_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
    'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc3_2,'string',arquivovoz11);
end
```

```
function botaoeloc3_3_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
    'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc3_3,'string',arquivovoz11);
end

function botaoeloc3_4_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
    'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc3_4,'string',arquivovoz11);
end

function textoeloc4_1_Callback(hObject, eventdata, handles)
function textoeloc4_1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc4_2_Callback(hObject, eventdata, handles)
function textoeloc4_2_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
                   get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc4_3_Callback(hObject, eventdata, handles)
function textoeloc4_3_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
                   get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc4_4_Callback(hObject, eventdata, handles)
function textoeloc4_4_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),...
                   get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor','white');
end

function textoeloc4_5_Callback(hObject, eventdata, handles)
function textoeloc4_5_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
                   get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function botaoeloc4_1_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
                  'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc4_1,'string',arquivovoz11);
end

function botaoeloc4_2_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
                  'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc4_2,'string',arquivovoz11);
end

function botaoeloc4_3_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
                  'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc4_3,'string',arquivovoz11);
end

function botaoeloc4_4_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
                  'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
```

```
    set(handles.textoeloc4_4,'string',arquivovoz11);
end

function botaoeloc4_5_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
                   'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc4_5,'string',arquivovoz11);
end

function textoeloc5_1_Callback(hObject, eventdata, handles)
function textoeloc5_1_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),...
                   get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc5_2_Callback(hObject, eventdata, handles)
function textoeloc5_2_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),...
                   get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc5_3_Callback(hObject, eventdata, handles)
function textoeloc5_3_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
                   get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc5_4_Callback(hObject, eventdata, handles)
function textoeloc5_4_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
                   get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc5_5_Callback(hObject, eventdata, handles)
function textoeloc5_5_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
                   get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor','white');
end

function botaoeloc5_1_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
                  'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc5_1,'string',arquivovoz11);
end

function botaoeloc5_2_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
                  'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc5_2,'string',arquivovoz11);
end

function botaoeloc5_3_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
                  'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc5_3,'string',arquivovoz11);
end

function botaoeloc5_4_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
                  'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc5_4,'string',arquivovoz11);
end

function botaoeloc5_5_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
                   'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
```

```
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc5_5,'string',arquivovoz11);
end

function nome3_Callback(hObject, eventdata, handles)
function nome3_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
                   get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function nome14_Callback(hObject, eventdata, handles)
function nome14_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
                   get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function nome4_Callback(hObject, eventdata, handles)
function nome4_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
                   get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc6_1_Callback(hObject, eventdata, handles)
function textoeloc6_1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
                   get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc6_2_Callback(hObject, eventdata, handles)
function textoeloc6_2_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
                   get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc6_3_Callback(hObject, eventdata, handles)
function textoeloc6_3_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
                   get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
function textoeloc6_4_Callback(hObject, eventdata, handles)
function textoeloc6_4_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
                   get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc6_5_Callback(hObject, eventdata, handles)
function textoeloc6_5_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
                   get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function botaoeloc6_1_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
                   'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc6_1,'string',arquivovoz11);
end

function botaoeloc6_2_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
    'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc6_2,'string',arquivovoz11);
end

function botaoeloc6_3_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
                   'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc6_3,'string',arquivovoz11);
end

function botaoeloc6_4_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
```

```
                    'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc6_4,'string',arquivovoz11);
end

function botaoeloc6_5_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
                    'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc6_5,'string',arquivovoz11);
end

function nome6_Callback(hObject, eventdata, handles)
function nome6_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),...
                    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc7_1_Callback(hObject, eventdata, handles)
function textoeloc7_1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
                    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc7_2_Callback(hObject, eventdata, handles)
function textoeloc7_2_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
                    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc7_3_Callback(hObject, eventdata, handles)
function textoeloc7_3_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
                    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
function textoeloc7_4_Callback(hObject, eventdata, handles)
function textoeloc7_4_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
                   get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc7_5_Callback(hObject, eventdata, handles)
function textoeloc7_5_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
                   get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function botaoeloc7_1_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
                  'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc7_1,'string',arquivovoz11);
end

function botaoeloc7_2_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
                  'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc7_2,'string',arquivovoz11);
end

function botaoeloc7_3_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
                   'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc7_3,'string',arquivovoz11);
end

function botaoeloc7_4_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
                  'Selecione o arquivo .wav para processamento');
```

```
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc7_4,'string',arquivovoz11);
end

function nome7_Callback(hObject, eventdata, handles)
function nome7_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
                   get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc9_1_Callback(hObject, eventdata, handles)
function textoeloc9_1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
                   get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc9_2_Callback(hObject, eventdata, handles)
function textoeloc9_2_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
                   get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc9_3_Callback(hObject, eventdata, handles)
function textoeloc9_3_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
                   get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc9_4_Callback(hObject, eventdata, handles)
function textoeloc9_4_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
                   get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc9_5_Callback(hObject, eventdata, handles)
function textoeloc9_5_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
                   get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
```

```
end

function botaoeloc9_1_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
                   'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc9_1,'string',arquivovoz11);
end

function botaoeloc9_2_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
                   'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc9_2,'string',arquivovoz11);
end

function botaoeloc9_3_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
                   'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc9_3,'string',arquivovoz11);
end

function botaoeloc9_4_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
                   'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc9_4,'string',arquivovoz11);
end

function botaoeloc9_5_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
                   'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
```

```
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc9_5,'string',arquivovoz11);
end

function textoeloc10_1_Callback(hObject, eventdata, handles)
function textoeloc10_1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
                   get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc10_2_Callback(hObject, eventdata, handles)
function textoeloc10_2_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
                   get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc10_3_Callback(hObject, eventdata, handles)
function textoeloc10_3_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
                   get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc10_4_Callback(hObject, eventdata, handles)
function textoeloc10_4_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
                   get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc10_5_Callback(hObject, eventdata, handles)
function textoeloc10_5_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
                   get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function botaoeloc10_1_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
                  'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
```

```
    set(handles.textoeloc10_1,'string',arquivovoz11);
end

function botaoeloc10_2_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
                   'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc10_2,'string',arquivovoz11);
end

function botaoeloc10_3_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
                   'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc10_3,'string',arquivovoz11);
end

function botaoeloc10_4_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
                   'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc10_4,'string',arquivovoz11);
end

function botaoeloc10_5_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
    'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc10_5,'string',arquivovoz11);
end

function nome10_Callback(hObject, eventdata, handles)
function nome10_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
                   get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor','white');
end

function textoeloc2_1_Callback(hObject, eventdata, handles)
function textoeloc2_1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
                   get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc2_2_Callback(hObject, eventdata, handles)
function textoeloc2_2_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc2_3_Callback(hObject, eventdata, handles)
function textoeloc2_3_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc2_4_Callback(hObject, eventdata, handles)
function textoeloc2_4_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
                   get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc2_5_Callback(hObject, eventdata, handles)
function textoeloc2_5_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),...
                   get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function botaoeloc2_1_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
                  'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc2_1,'string',arquivovoz11);
```

```
end

function botaoeloc2_2_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
                   'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc2_2,'string',arquivovoz11);
end

function botaoeloc2_3_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
                  'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc2_3,'string',arquivovoz11);
end

function botaoeloc2_4_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
                  'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc2_4,'string',arquivovoz11);
end

function botaoeloc2_5_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
                  'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc2_5,'string',arquivovoz11);
end

function nome2_Callback(hObject, eventdata, handles)
function nome2_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),...
                   get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor','white');
end

function textoeloc8_1_Callback(hObject, eventdata, handles)
function textoeloc8_1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
                   get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc8_2_Callback(hObject, eventdata, handles)
function textoeloc8_2_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
                   get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc8_3_Callback(hObject, eventdata, handles)
function textoeloc8_3_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
                   get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc8_4_Callback(hObject, eventdata, handles)
function textoeloc8_4_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
                    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc8_5_Callback(hObject, eventdata, handles)
function textoeloc8_5_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
                    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function botaoeloc8_2_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
                  'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc8_2,'string',arquivovoz11);
end
```

```
function botaoeloc8_3_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
                  'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc8_3,'string',arquivovoz11);
end

function botaoeloc8_4_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
                  'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc8_4,'string',arquivovoz11);
end

function botaoeloc8_5_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
                  'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc8_5,'string',arquivovoz11);
end

function nome8_Callback(hObject, eventdata, handles)
function nome8_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
                   get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function botaoeloc8_1_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
                  'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc8_1,'string',arquivovoz11);
end
```

```
function pushbutton49_Callback(hObject, eventdata, handles)

function tabA_Callback(hObject, eventdata, handles)
set(handles.painelA, 'Visible', 'on');
set(handles.painelB, 'Visible', 'off');

function tabB_Callback(hObject, eventdata, handles)
set(handles.painelA, 'Visible', 'off');
set(handles.painelB, 'Visible', 'on');

function nome1_Callback(hObject, eventdata, handles)
function nome1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
                   get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function botaoeloc3_5_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
                  'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc3_5,'string',arquivovoz11);
end

function botaoeloc7_5_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
                  'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc7_5,'string',arquivovoz11);
end

function nome5_Callback(hObject, eventdata, handles)
function nome5_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
                   get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function nome9_Callback(hObject, eventdata, handles)
function nome9_CreateFcn(hObject, eventdata, handles)
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc11_1_Callback(hObject, eventdata, handles)
function textoeloc11_1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
                   get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc11_2_Callback(hObject, eventdata, handles)
function textoeloc11_2_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
                    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc11_3_Callback(hObject, eventdata, handles)
function textoeloc11_3_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
                   get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc11_4_Callback(hObject, eventdata, handles)
function textoeloc11_4_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
                   get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc11_5_Callback(hObject, eventdata, handles)
function textoeloc11_5_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
                   get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc13_1_Callback(hObject, eventdata, handles)
function textoeloc13_1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
function textoeloc13_2_Callback(hObject, eventdata, handles)
function textoeloc13_2_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
                   get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc13_3_Callback(hObject, eventdata, handles)
function textoeloc13_3_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
                    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc13_4_Callback(hObject, eventdata, handles)
function textoeloc13_4_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
                    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc13_5_Callback(hObject, eventdata, handles)
function textoeloc13_5_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
                    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function botaoeloc11_1_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
                'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc11_1,'string',arquivovoz11);
end

function botaoeloc11_2_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
                   'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc11_2,'string',arquivovoz11);
end
```

```
function botaoeloc11_3_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
                  'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc11_3,'string',arquivovoz11);
end

function botaoeloc11_4_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
                  'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc11_4,'string',arquivovoz11);
end

function botaoeloc11_5_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
                  'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc11_5,'string',arquivovoz11);
end

function botaoeloc13_1_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
                  'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc13_1,'string',arquivovoz11);
end

function botaoeloc13_2_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
                  'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
```

```
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc13_2,'string',arquivovoz11);
end

function botaoeloc13_3_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
                    'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc13_3,'string',arquivovoz11);
end

function botaoeloc13_4_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
                'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc13_4,'string',arquivovoz11);
end

function textoeloc14_1_Callback(hObject, eventdata, handles)
function textoeloc14_1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
                   get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc14_2_Callback(hObject, eventdata, handles)
function textoeloc14_2_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc14_3_Callback(hObject, eventdata, handles)
function textoeloc14_3_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc14_4_Callback(hObject, eventdata, handles)
```

```
function textoeloc14_4_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
                get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc14_5_Callback(hObject, eventdata, handles)
function textoeloc14_5_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function botaoeloc14_1_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
    'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc14_1,'string',arquivovoz11);
end

function botaoeloc14_2_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
    'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc14_2,'string',arquivovoz11);
end

function botaoeloc14_3_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
    'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc14_3,'string',arquivovoz11);
end

function botaoeloc14_4_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
    'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
```

```
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc14_4,'string',arquivovoz11);
end

function botaoeloc14_5_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
    'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc14_5,'string',arquivovoz11);
end

function textoeloc15_1_Callback(hObject, eventdata, handles)
function textoeloc15_1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc15_2_Callback(hObject, eventdata, handles)
function textoeloc15_2_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc15_3_Callback(hObject, eventdata, handles)
function textoeloc15_3_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc15_4_Callback(hObject, eventdata, handles)
function textoeloc15_4_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc15_5_Callback(hObject, eventdata, handles)
function textoeloc15_5_CreateFcn(hObject, eventdata, handles)
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function botaoeloc15_1_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
    'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc15_1,'string',arquivovoz11);
end

function botaoeloc15_2_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
    'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc15_2,'string',arquivovoz11);
end

function botaoeloc15_3_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
    'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc15_3,'string',arquivovoz11);
end

function botaoeloc15_4_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
    'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc15_4,'string',arquivovoz11);
end

function botaoeloc15_5_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
```

```
    'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc15_5,'string',arquivovoz11);
end

function nome13_Callback(hObject, eventdata, handles)
function nome13_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc16_1_Callback(hObject, eventdata, handles)
function textoeloc16_1_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc16_2_Callback(hObject, eventdata, handles)
function textoeloc16_2_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc16_3_Callback(hObject, eventdata, handles)
function textoeloc16_3_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc16_4_Callback(hObject, eventdata, handles)
function textoeloc16_4_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc16_5_Callback(hObject, eventdata, handles)
function textoeloc16_5_CreateFcn(hObject, eventdata, handles)
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function botaoeloc16_1_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
    'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc16_1,'string',arquivovoz11);
end

function botaoeloc16_2_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
    'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc16_2,'string',arquivovoz11);
end

function botaoeloc16_3_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
    'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc16_3,'string',arquivovoz11);
end

function botaoeloc16_4_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
    'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc16_4,'string',arquivovoz11);
end

function botaoeloc16_5_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
```

```
    'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc16_5,'string',arquivovoz11);
end

function nome16_Callback(hObject, eventdata, handles)
function nome16_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc17_1_Callback(hObject, eventdata, handles)
function textoeloc17_1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc17_2_Callback(hObject, eventdata, handles)
function textoeloc17_2_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc17_3_Callback(hObject, eventdata, handles)
function textoeloc17_3_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc17_4_Callback(hObject, eventdata, handles)
function textoeloc17_4_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc17_5_Callback(hObject, eventdata, handles)
function textoeloc17_5_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor','white');
end

function botaoeloc17_1_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
    'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc17_1,'string',arquivovoz11);
end

function botaoeloc17_2_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
    'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc17_2,'string',arquivovoz11);
end

function botaoeloc17_3_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
    'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc17_3,'string',arquivovoz11);
end

function botaoeloc17_4_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
    'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc17_4,'string',arquivovoz11);
end

function nome17_Callback(hObject, eventdata, handles)
function nome17_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor','white');
end

function textoeloc19_1_Callback(hObject, eventdata, handles)
function textoeloc19_1_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc19_2_Callback(hObject, eventdata, handles)
function textoeloc19_2_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc19_3_Callback(hObject, eventdata, handles)
function textoeloc19_3_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc19_4_Callback(hObject, eventdata, handles)
function textoeloc19_4_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc19_5_Callback(hObject, eventdata, handles)
function textoeloc19_5_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function botaoeloc19_1_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
    'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc19_1,'string',arquivovoz11);
```

```
end

function botaoeloc19_2_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
    'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc19_2,'string',arquivovoz11);
end

function botaoeloc19_3_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
    'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc19_3,'string',arquivovoz11);
end

function botaoeloc19_4_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
    'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc19_4,'string',arquivovoz11);
end

function botaoeloc19_5_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
    'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc19_5,'string',arquivovoz11);
end

function textoeloc20_1_Callback(hObject, eventdata, handles)
function textoeloc20_1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
```

```
end

function textoeloc20_2_Callback(hObject, eventdata, handles)
function textoeloc20_2_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc20_3_Callback(hObject, eventdata, handles)
function textoeloc20_3_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc20_4_Callback(hObject, eventdata, handles)
function textoeloc20_4_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc20_5_Callback(hObject, eventdata, handles)
function textoeloc20_5_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function botaoeloc20_1_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
    'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc20_1,'string',arquivovoz11);
end

function botaoeloc20_2_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
    'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
```

```
    set(handles.textoeloc20_2,'string',arquivovoz11);
end

function botaoeloc20_3_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
    'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc20_3,'string',arquivovoz11);
end

function botaoeloc20_4_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
    'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc20_4,'string',arquivovoz11);
end

function botaoeloc20_5_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
    'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc20_5,'string',arquivovoz11);
end

function nome20_Callback(hObject, eventdata, handles)
function nome20_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc12_1_Callback(hObject, eventdata, handles)
function textoeloc12_1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
function textoeloc12_2_Callback(hObject, eventdata, handles)
function textoeloc12_2_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc12_3_Callback(hObject, eventdata, handles)
function textoeloc12_3_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc12_4_Callback(hObject, eventdata, handles)
function textoeloc12_4_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc12_5_Callback(hObject, eventdata, handles)
function textoeloc12_5_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function botaoeloc12_1_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
    'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc12_1,'string',arquivovoz11);
end

function botaoeloc12_2_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
    'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc12_2,'string',arquivovoz11);
```

```
end

function botaoeloc12_3_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
    'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc12_3,'string',arquivovoz11);
end

function botaoeloc12_4_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
    'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc12_4,'string',arquivovoz11);
end

function botaoeloc12_5_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
    'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc12_5,'string',arquivovoz11);
end

function nome12_Callback(hObject, eventdata, handles)
function nome12_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc18_1_Callback(hObject, eventdata, handles)
function textoeloc18_1_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
function textoeloc18_2_Callback(hObject, eventdata, handles)
function textoeloc18_2_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc18_3_Callback(hObject, eventdata, handles)
function textoeloc18_3_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc18_4_Callback(hObject, eventdata, handles)
function textoeloc18_4_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function textoeloc18_5_Callback(hObject, eventdata, handles)
function textoeloc18_5_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function botaoeloc18_2_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
    'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc18_2,'string',arquivovoz11);
end

function botaoeloc18_3_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
    'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
```

```
    set(handles.textoeloc18_3,'string',arquivovoz11);
end

function botaoeloc18_4_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
    'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc18_4,'string',arquivovoz11);
end

function botaoeloc18_5_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
    'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc18_5,'string',arquivovoz11);
end

function nome18_Callback(hObject, eventdata, handles)
function nome18_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function botaoeloc18_1_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav','Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc18_1,'string',arquivovoz11);
end

function nome11_Callback(hObject, eventdata, handles)
function nome11_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function nome15_Callback(hObject, eventdata, handles)
```

```
function nome15_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function nome19_Callback(hObject, eventdata, handles)
function nome19_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function botaoeloc13_5_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
    'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc13_5,'string',arquivovoz11);
end

function botaoeloc17_5_Callback(hObject, eventdata, handles)
[nome,diretorio] = uigetfile('*.wav',...
    'Selecione o arquivo .wav para processamento');
if isequal(nome,0) || isequal(diretorio,0)
    disp('Usuario pressionou Cancelar')
else
    arquivovoz11= strcat(diretorio,nome);
    set(handles.textoeloc17_5,'string',arquivovoz11);
end

function botaogerar_Callback(hObject, eventdata, handles)

h = waitbar(0,'Processando...');          %habilita barra de processamento

eloc1string = get(handles.textoeloc1_1,'string');
eloc1 = audioread(eloc1string);
eloc2string = get(handles.textoeloc1_2,'string');
eloc2 = audioread(eloc2string);
eloc3string = get(handles.textoeloc1_3,'string');
eloc3 = audioread(eloc3string);
eloc4string = get(handles.textoeloc1_4,'string');
eloc4 = audioread(eloc4string);
eloc5string = get(handles.textoeloc1_5,'string');
eloc5 = audioread(eloc5string);
```

```
%gera padrao do locutor 1                              %barra de processamento
padrao1=gerPadrao5eloc(eloc1,eloc2,eloc3,eloc4,eloc5);  waitbar(1/21,h);

eloc1string = get(handles.textoeloc2_1,'string');
eloc1 = audioread(eloc1string);
eloc2string = get(handles.textoeloc2_2,'string');
eloc2 = audioread(eloc2string);
eloc3string = get(handles.textoeloc2_3,'string');
eloc3 = audioread(eloc3string);
eloc4string = get(handles.textoeloc2_4,'string');
eloc4 = audioread(eloc4string);
eloc5string = get(handles.textoeloc2_5,'string');
eloc5 = audioread(eloc5string);
%gera padrao do locutor 2                              %barra de processamento
padrao2=gerPadrao5eloc(eloc1,eloc2,eloc3,eloc4,eloc5);  waitbar(2/21,h);

eloc1string = get(handles.textoeloc3_1,'string');
eloc1 = audioread(eloc1string);
eloc2string = get(handles.textoeloc3_2,'string');
eloc2 = audioread(eloc2string);
eloc3string = get(handles.textoeloc3_3,'string');
eloc3 = audioread(eloc3string);
eloc4string = get(handles.textoeloc3_4,'string');
eloc4 = audioread(eloc4string);
eloc5string = get(handles.textoeloc3_5,'string');
eloc5 = audioread(eloc5string);
%gera padrao do locutor 3                              %barra de processamento
padrao3=gerPadrao5eloc(eloc1,eloc2,eloc3,eloc4,eloc5);   waitbar(3/21,h);

eloc1string = get(handles.textoeloc4_1,'string');
eloc1 = audioread(eloc1string);
eloc2string = get(handles.textoeloc4_2,'string');
eloc2 = audioread(eloc2string);
eloc3string = get(handles.textoeloc4_3,'string');
eloc3 = audioread(eloc3string);
eloc4string = get(handles.textoeloc4_4,'string');
eloc4 = audioread(eloc4string);
eloc5string = get(handles.textoeloc4_5,'string');
eloc5 = audioread(eloc5string);
%gera padrao do locutor 4                              %barra de processamento
padrao4=gerPadrao5eloc(eloc1,eloc2,eloc3,eloc4,eloc5);  waitbar(4/21,h);

eloc1string = get(handles.textoeloc5_1,'string');
eloc1 = audioread(eloc1string);
eloc2string = get(handles.textoeloc5_2,'string');
eloc2 = audioread(eloc2string);
eloc3string = get(handles.textoeloc5_3,'string');
```

```
eloc3 = audioread(eloc3string);
eloc4string = get(handles.textoeloc5_4,'string');
eloc4 = audioread(eloc4string);
eloc5string = get(handles.textoeloc5_5,'string');
eloc5 = audioread(eloc5string);
%gera padrao do locutor 5                          %barra de processamento
padrao5=gerPadrao5eloc(eloc1,eloc2,eloc3,eloc4,eloc5); waitbar(5/21,h);

eloc1string = get(handles.textoeloc6_1,'string');
eloc1 = audioread(eloc1string);
eloc2string = get(handles.textoeloc6_2,'string');
eloc2 = audioread(eloc2string);
eloc3string = get(handles.textoeloc6_3,'string');
eloc3 = audioread(eloc3string);
eloc4string = get(handles.textoeloc6_4,'string');
eloc4 = audioread(eloc4string);
eloc5string = get(handles.textoeloc6_5,'string');
eloc5 = audioread(eloc5string);
%gera padrao do locutor 6                          %barra de processamento
padrao6=gerPadrao5eloc(eloc1,eloc2,eloc3,eloc4,eloc5); waitbar(6/21,h);

eloc1string = get(handles.textoeloc7_1,'string');
eloc1 = audioread(eloc1string);
eloc2string = get(handles.textoeloc7_2,'string');
eloc2 = audioread(eloc2string);
eloc3string = get(handles.textoeloc7_3,'string');
eloc3 = audioread(eloc3string);
eloc4string = get(handles.textoeloc7_4,'string');
eloc4 = audioread(eloc4string);
eloc5string = get(handles.textoeloc7_5,'string');
eloc5 = audioread(eloc5string);
%gera padrao do locutor 7                          %barra de processamento
padrao7=gerPadrao5eloc(eloc1,eloc2,eloc3,eloc4,eloc5);  waitbar(7/21,h);

eloc1string = get(handles.textoeloc8_1,'string');
eloc1 = audioread(eloc1string);
eloc2string = get(handles.textoeloc8_2,'string');
eloc2 = audioread(eloc2string);
eloc3string = get(handles.textoeloc8_3,'string');
eloc3 = audioread(eloc3string);
eloc4string = get(handles.textoeloc8_4,'string');
eloc4 = audioread(eloc4string);
eloc5string = get(handles.textoeloc8_5,'string');
eloc5 = audioread(eloc5string);
%gera padrao do locutor 8                          %barra de processamento
padrao8=gerPadrao5eloc(eloc1,eloc2,eloc3,eloc4,eloc5);  waitbar(8/21,h);
```

```
eloc1string = get(handles.textoeloc9_1,'string');
eloc1 = audioread(eloc1string);
eloc2string = get(handles.textoeloc9_2,'string');
eloc2 = audioread(eloc2string);
eloc3string = get(handles.textoeloc9_3,'string');
eloc3 = audioread(eloc3string);
eloc4string = get(handles.textoeloc9_4,'string');
eloc4 = audioread(eloc4string);
eloc5string = get(handles.textoeloc9_5,'string');
eloc5 = audioread(eloc5string);
%gera padrao do locutor 9                          %barra de processamento
padrao9=gerPadrao5eloc(eloc1,eloc2,eloc3,eloc4,eloc5);  waitbar(9/21,h);

eloc1string = get(handles.textoeloc10_1,'string');
eloc1 = audioread(eloc1string);
eloc2string = get(handles.textoeloc10_2,'string');
eloc2 = audioread(eloc2string);
eloc3string = get(handles.textoeloc10_3,'string');
eloc3 = audioread(eloc3string);
eloc4string = get(handles.textoeloc10_4,'string');
eloc4 = audioread(eloc4string);
eloc5string = get(handles.textoeloc10_5,'string');
eloc5 = audioread(eloc5string);
%gera padrao do locutor 10                          %barra de processamento
padrao10=gerPadrao5eloc(eloc1,eloc2,eloc3,eloc4,eloc5);  waitbar(10/21,h);

eloc1string = get(handles.textoeloc11_1,'string');
eloc1 = audioread(eloc1string);
eloc2string = get(handles.textoeloc11_2,'string');
eloc2 = audioread(eloc2string);
eloc3string = get(handles.textoeloc11_3,'string');
eloc3 = audioread(eloc3string);
eloc4string = get(handles.textoeloc11_4,'string');
eloc4 = audioread(eloc4string);
eloc5string = get(handles.textoeloc11_5,'string');
eloc5 = audioread(eloc5string);
%gera padrao do locutor 11                          %barra de processamento
padrao11=gerPadrao5eloc(eloc1,eloc2,eloc3,eloc4,eloc5);  waitbar(11/21,h);

eloc1string = get(handles.textoeloc12_1,'string');
eloc1 = audioread(eloc1string);
eloc2string = get(handles.textoeloc12_2,'string');
eloc2 = audioread(eloc2string);
eloc3string = get(handles.textoeloc12_3,'string');
eloc3 = audioread(eloc3string);
eloc4string = get(handles.textoeloc12_4,'string');
eloc4 = audioread(eloc4string);
```

```
eloc5string = get(handles.textoeloc12_5,'string');
eloc5 = audioread(eloc5string);
%gera padrao do locutor 12                          %barra de processamento
padrao12=gerPadrao5eloc(eloc1,eloc2,eloc3,eloc4,eloc5);  waitbar(12/21,h);

eloc1string = get(handles.textoeloc13_1,'string');
eloc1 = audioread(eloc1string);
eloc2string = get(handles.textoeloc13_2,'string');
eloc2 = audioread(eloc2string);
eloc3string = get(handles.textoeloc13_3,'string');
eloc3 = audioread(eloc3string);
eloc4string = get(handles.textoeloc13_4,'string');
eloc4 = audioread(eloc4string);
eloc5string = get(handles.textoeloc13_5,'string');
eloc5 = audioread(eloc5string);
%gera padrao do locutor 13                          %barra de processamento
padrao13=gerPadrao5eloc(eloc1,eloc2,eloc3,eloc4,eloc5);  waitbar(13/21,h);

eloc1string = get(handles.textoeloc14_1,'string');
eloc1 = audioread(eloc1string);
eloc2string = get(handles.textoeloc14_2,'string');
eloc2 = audioread(eloc2string);
eloc3string = get(handles.textoeloc14_3,'string');
eloc3 = audioread(eloc3string);
eloc4string = get(handles.textoeloc14_4,'string');
eloc4 = audioread(eloc4string);
eloc5string = get(handles.textoeloc14_5,'string');
eloc5 = audioread(eloc5string);
%gera padrao do locutor 14                          %barra de processamento
padrao14=gerPadrao5eloc(eloc1,eloc2,eloc3,eloc4,eloc5);  waitbar(14/21,h);

eloc1string = get(handles.textoeloc15_1,'string');
eloc1 = audioread(eloc1string);
eloc2string = get(handles.textoeloc15_2,'string');
eloc2 = audioread(eloc2string);
eloc3string = get(handles.textoeloc15_3,'string');
eloc3 = audioread(eloc3string);
eloc4string = get(handles.textoeloc15_4,'string');
eloc4 = audioread(eloc4string);
eloc5string = get(handles.textoeloc15_5,'string');
eloc5 = audioread(eloc5string);
%gera padrao do locutor 15                          %barra de processamento
padrao15=gerPadrao5eloc(eloc1,eloc2,eloc3,eloc4,eloc5);  waitbar(15/21,h);

eloc1string = get(handles.textoeloc16_1,'string');
eloc1 = audioread(eloc1string);
eloc2string = get(handles.textoeloc16_2,'string');
```

```
eloc2 = audioread(eloc2string);
eloc3string = get(handles.textoeloc16_3,'string');
eloc3 = audioread(eloc3string);
eloc4string = get(handles.textoeloc16_4,'string');
eloc4 = audioread(eloc4string);
eloc5string = get(handles.textoeloc16_5,'string');
eloc5 = audioread(eloc5string);
%gera padrao do locutor 16                          %barra de processamento
padrao16=gerPadrao5eloc(eloc1,eloc2,eloc3,eloc4,eloc5);  waitbar(16/21,h);

eloc1string = get(handles.textoeloc17_1,'string');
eloc1 = audioread(eloc1string);
eloc2string = get(handles.textoeloc17_2,'string');
eloc2 = audioread(eloc2string);
eloc3string = get(handles.textoeloc17_3,'string');
eloc3 = audioread(eloc3string);
eloc4string = get(handles.textoeloc17_4,'string');
eloc4 = audioread(eloc4string);
eloc5string = get(handles.textoeloc17_5,'string');
eloc5 = audioread(eloc5string);
%gera padrao do locutor 17                          %barra de processamento
padrao17=gerPadrao5eloc(eloc1,eloc2,eloc3,eloc4,eloc5);  waitbar(17/21,h);

eloc1string = get(handles.textoeloc18_1,'string');
eloc1 = audioread(eloc1string);
eloc2string = get(handles.textoeloc18_2,'string');
eloc2 = audioread(eloc2string);
eloc3string = get(handles.textoeloc18_3,'string');
eloc3 = audioread(eloc3string);
eloc4string = get(handles.textoeloc18_4,'string');
eloc4 = audioread(eloc4string);
eloc5string = get(handles.textoeloc18_5,'string');
eloc5 = audioread(eloc5string);
%gera padrao do locutor 18                          %barra de processamento
padrao18=gerPadrao5eloc(eloc1,eloc2,eloc3,eloc4,eloc5);  waitbar(18/21,h);

eloc1string = get(handles.textoeloc19_1,'string');
eloc1 = audioread(eloc1string);
eloc2string = get(handles.textoeloc19_2,'string');
eloc2 = audioread(eloc2string);
eloc3string = get(handles.textoeloc19_3,'string');
eloc3 = audioread(eloc3string);
eloc4string = get(handles.textoeloc19_4,'string');
eloc4 = audioread(eloc4string);
eloc5string = get(handles.textoeloc19_5,'string');
eloc5 = audioread(eloc5string);
%gera padrao do locutor 19                          %barra de processamento
```

```
padrao19=gerPadrao5eloc(eloc1,eloc2,eloc3,eloc4,eloc5);  waitbar(19/21,h);

eloc1string = get(handles.textoeloc20_1,'string');
eloc1 = audioread(eloc1string);
eloc2string = get(handles.textoeloc20_2,'string');
eloc2 = audioread(eloc2string);
eloc3string = get(handles.textoeloc20_3,'string');
eloc3 = audioread(eloc3string);
eloc4string = get(handles.textoeloc20_4,'string');
eloc4 = audioread(eloc4string);
eloc5string = get(handles.textoeloc20_5,'string');
eloc5 = audioread(eloc5string);
%gera padrao do locutor 20                          %barra de processamento
padrao20=gerPadrao5eloc(eloc1,eloc2,eloc3,eloc4,eloc5); waitbar(20/21,h);
%cria matriz com os padroes dos 20 locutores
matrizpadrao=[padrao1;padrao2;padrao3;padrao4;padrao5;padrao6;padrao7;...
    padrao8;padrao9;padrao10;padrao11;padrao12;padrao13;padrao14;...
    padrao15;padrao16;padrao17;padrao18;padrao19;padrao20];

locutor1=get(handles.nome1,'string');
locutor2=get(handles.nome2,'string');
locutor3=get(handles.nome3,'string');
locutor4=get(handles.nome4,'string');
locutor5=get(handles.nome5,'string');
locutor6=get(handles.nome6,'string');
locutor7=get(handles.nome7,'string');
locutor8=get(handles.nome8,'string');
locutor9=get(handles.nome9,'string');
locutor10=get(handles.nome10,'string');
locutor11=get(handles.nome11,'string');
locutor12=get(handles.nome12,'string');
locutor13=get(handles.nome13,'string');
locutor14=get(handles.nome14,'string');
locutor15=get(handles.nome15,'string');
locutor16=get(handles.nome16,'string');
locutor17=get(handles.nome17,'string');
locutor18=get(handles.nome18,'string');
locutor19=get(handles.nome19,'string');
locutor20=get(handles.nome20,'string');

waitbar(21/21,h);
close(h);
%gera vetor com nome dos 20 locutores
vetornomes={locutor1;locutor2;locutor3;locutor4;locutor5;locutor6;...
    locutor7;locutor8;locutor9;locutor10;locutor11;locutor12;...
    locutor13;locutor14;locutor15;locutor16;locutor17;locutor18;...
    locutor19;locutor20};
```

```
T1 = table([vetornomes],[matrizpadrao]);
[filename, pathname] = uiputfile('*.txt',...
    'Salvar arquivo de padroes dos locutores como:'); %abre tela salvar
arquivodospadroes= strcat(pathname,filename);
writetable(T1,arquivodospadroes);       % gera arquivo txt com padroes
```

vetorCaract.m

```
function [vetorCaracteristico]= vetorCaract(arquivoVoz)
%VETORCARACT     Programa intermediario que faz uso da amplitude  media
%                presente no modulo do espectro de frequencias de cada
%                sinal  de voz,  para gerar o vetor caracteristico de
%                treinamento de cada locutor em  teste. Este vetor
%                sera utilizado  no programa de geracao do padrao de cada
%                usuario.
%                Recebe como entrada o arquivo de voz (wav) de treinamento
%                e gera como saida seu vetor caracteristico.
%
%Autores:        Roberto F. B. Sotero Filho e Helio Magalhaes de Oliveira
%E-mail:         rsotero@hotmail.com e hmo@ufpe.br
%Universidade:   Universidade Federal de Pernambuco
%Data:           04/06/2008

fs = 8000;                              % frequencia amostragem utilizada.
c = fs*20e-3;                           % numero de amostras em 20ms
arquivoVoz = myVAD(arquivoVoz);         %processa os endpoints do sinal
N = length(arquivoVoz);                 %comprimento do arquivo de entrada

%calculo do numero de blocos no arquivo de voz
if rem(N,c) ~= 0
    n = round(N/c + 0.5);
    arquivoVoz = [arquivoVoz; 0*ones(n*c-N,1)];
else
    n = N/c;
end

HAM =  hamming(c);                      %criando janela de Hamming
%-------------------------------------------------------------------------|
% Bloco de montagem espectro de frequencia simplificado (1 amostra/oitava)|
%-------------------------------------------------------------------------|

for i=0:n-1
    %cria bloco de FFTs janeladas com apenas as amplitudes.
  spectrum_bloco(c*i+1:c*i+c)= abs(fft(HAM(1:c).*arquivoVoz(c*i+1:c*i+c)));
end

for k =0:n-1;

        spectrum_bloco(c*k+1)= 0;  % anular parte dc
        spectrum_bloco(c*k+81)= 0; % anular ponto desprezivel no espectro

%-------------------------------------------------------------------------
```

```
%                       Oitava -2 (32 - 64Hz)
      intmenos2 = c*k+2;
      a(intmenos2)= spectrum_bloco(intmenos2);
      spectrum_bloco(find(a ~= max(a)& a~=0))= 0;
      clear a;
%-------------------------------------------------------------------------
%                       Oitava -1 (64 - 128Hz)
      intmenos1 = c*k+3;
      b(intmenos1)= spectrum_bloco(intmenos1);
      spectrum_bloco(find(b ~= max(b)& b~=0))= 0;
      clear b;
%-------------------------------------------------------------------------
%                       Oitava 0 (128 - 256Hz)
      int0 = c*k+4:c*k+6;
      g(int0)= spectrum_bloco(int0);
      spectrum_bloco(find(g ~= max(g)& g~=0))= 0;
      clear g;
%-------------------------------------------------------------------------
%                       Oitava 1 (256 - 512 Hz)
      int1 = c*k+7:c*k+11;
      y(int1)= spectrum_bloco(int1);
      spectrum_bloco(find(y ~= max(y)& y~=0))= 0;
      clear y;
%-------------------------------------------------------------------------
%                       Oitava 2 (512 - 1024 Hz)
      int2 = c*k+12:c*k+21;
      z(int2)=spectrum_bloco(int2);
      spectrum_bloco(find(z ~= max(z)& z~=0))=0;
      clear z;
%-------------------------------------------------------------------------
%                       Oitava 3 (1024 - 2048 Hz)
      int3 = c*k+22:c*k+41;
      h(int3)=spectrum_bloco(int3);
      spectrum_bloco(find(h ~= max(h)& h~=0))=0;
      clear h;
%-------------------------------------------------------------------------
%                       Oitava 4 (2048 - 4096 Hz)
      int4 = c*k+42:c*k+80;
      j(int4)=spectrum_bloco(int4);
      spectrum_bloco(find(j ~= max(j)&j~=0))=0;
      clear j;
end
%-------------------------------------------------------------------------
%      MONTAGEM DAS ENERGIAS POR OITAVA E COMPARACAO DO PADRAO DE VOZ
%-------------------------------------------------------------------------
for i=0:n-1
%coloca os elementos das oitavas em linhas da matriz "block matrix"
```

```
    spc_block_matrix_oitmenos2(i+1,2)= spectrum_bloco(c*i+2);
    spc_block_matrix_oitmenos1(i+1,3)= spectrum_bloco(c*i+3);
    spc_block_matrix_oit0(i+1,4:6)= spectrum_bloco(c*i+4:c*i+6);
    spc_block_matrix_oit1(i+1,7:11)= spectrum_bloco(c*i+7:c*i+11);
    spc_block_matrix_oit2(i+1,12:21)= spectrum_bloco(c*i+12:c*i+21);
    spc_block_matrix_oit3(i+1,22:41)= spectrum_bloco(c*i+22:c*i+41);
    spc_block_matrix_oit4(i+1,42:80)= spectrum_bloco(c*i+42:c*i+80);
end
% media das amplitudes de todas as janelas para as oitava
    spc_block_matrix_oitmenos2_mean = mean(spc_block_matrix_oitmenos2);
    spc_block_matrix_oitmenos1_mean = mean(spc_block_matrix_oitmenos1);
    spc_block_matrix_oit0_mean = mean(spc_block_matrix_oit0);
    spc_block_matrix_oit1_mean = mean(spc_block_matrix_oit1);
    spc_block_matrix_oit2_mean = mean(spc_block_matrix_oit2);
    spc_block_matrix_oit3_mean = mean(spc_block_matrix_oit3);
    spc_block_matrix_oit4_mean = mean(spc_block_matrix_oit4);
% Soma das medias das amplitudes de todos os elementos das oitavas
    somaOitavamenos2 = sum(spc_block_matrix_oitmenos2_mean);
    somaOitavamenos1 = sum(spc_block_matrix_oitmenos1_mean);
    somaOitava0 = sum(spc_block_matrix_oit0_mean);
    somaOitava1 = sum(spc_block_matrix_oit1_mean);
    somaOitava2 = sum(spc_block_matrix_oit2_mean);
    somaOitava3 = sum(spc_block_matrix_oit3_mean);
    somaOitava4 = sum(spc_block_matrix_oit4_mean);
%Soma geral de todas as oitavas
somatotal = somaOitavamenos2 + somaOitavamenos1 + somaOitava0 + ...
    somaOitava1 + somaOitava2 + somaOitava3 + somaOitava4;
%Porcentagem de amplitude por oitava do arquivo de entrada a ser comparado
vetorCaracteristico = [somaOitavamenos2 somaOitavamenos1 somaOitava0 ...
                       somaOitava1 somaOitava2 somaOitava3 somaOitava4]...
                       /somatotal * 100;
```

gerPadrao5eloc.m

```
function [padrao]= gerPadrao5eloc(x1,x2,x3,x4,x5)
%GERPADRAO5ELOC  Programa que calcula o padrao (tom equivalente / oitava)
%                de cadalocutor.
%                Recebe como entrada os 5 arquivos (wav) de treinamento e
%                gera como saida o padrao desse locutor.
%
%Autores:        Roberto F. B. Sotero Filho e Helio Magalhaes de Oliveira
%E-mail:         rsotero@hotmail.com e hmo@ufpe.br
%Universidade:   Universidade Federal de Pernambuco
%Data:           04/06/2008

vetorCaract(x1);
```

```
ans1=ans;
vetorCaract(x2);
ans2=ans;
vetorCaract(x3);
ans3=ans;
vetorCaract(x4);
ans4=ans;
vetorCaract(x5);
ans5=ans;

% geracao do padrao (media dos 5 vetores caracteristicos de treinamento
padrao = (ans1+ans2+ans3+ans4+ans5)/5;


end
```

gerPadraoEdit.m

```
function [padrao]= gerPadraoEdit(x1,x2,x3,x4,x5,x6) %acrescentar entradas
%GERPADRAOEDIT   Programa, editavel pelo usuario, que calcula o padrao (tom
%                equivalente/oitava) de cada locutor de acordo com procedi-
%                mentos disponiveis no Anexo C.
%                Recebe como entrada os N arquivos (wav) de treinamento e
%                gera como saida o padrao desse locutor.
%
%Autores:        Roberto F. B. Sotero Filho e Helio Magalhaes de Oliveira
%E-mail:         rsotero@hotmail.com e hmo@ufpe.br
%Universidade:   Universidade Federal de Pernambuco
%Data:           04/06/2008

vetorCaract(x1);
ans1=ans;
vetorCaract(x2);
ans2=ans;
vetorCaract(x3);
ans3=ans;
vetorCaract(x4);
ans4=ans;
vetorCaract(x5);
ans5=ans;
vetorCaract(x6);            %exemplo para 6 elocucoes de treinamento
ans6=ans;
% vetorCaract(xN);          %acrescentar ou retirar as linhas de codigos de
% ansN=ans;                 %acordo com o numero de elocucoes de treinamento

%geracao do padrao (media dos N vetores caracteristicos de treinamento
```

```
%padrao = (ans1+ans2+ans3+ans4+ans5+ans6+ansN)/N;  %editavel pelo usuario

%exemplo para 6 elocucoes de treinamento
padrao = (ans1+ans2+ans3+ans4+ans5+ans6)/6;

 end
```

myVAD.m

```
function trimmedX = myVAD(x)
%Author:        Olutope Foluso Omogbenigun
%Email:         olutopeomogbenigun at hotmail.com
%University:    London Metropolitan University
%Date:          02/08/07
%Syntax:        trimmedSample = myVAD2(samplex);
%This function accepts an audio sample 'samplex' as input and returns a
%trimmed down version with non-speech sections trimmed off. Also known as
%voice activity detection, it utilises the algorithm due to Rabiner &
%Sambur (1975)

Ini = 0.1;          %Initial silence duration in seconds
Ts = 0.01;          %Frame width in seconds
Tsh = 0.005;        %Frame shift in seconds
Fs = 16000;          %Sampling Frequency
counter1 = 0;
counter2 = 0;
counter3 = 0;
counter4 = 0;
ZCRCountf = 0;      %Stores forward count of crossing rate > IZCT
ZCRCountb = 0;      %As above, for backward count
ZTh = 40;           %Zero crossing comparison rate for threshold
w_sam = fix(Ts*Fs);                   %No of Samples/window
o_sam = fix(Tsh*Fs);                  %No of samples/overlap
lengthX = length(x);
segs = fix((lengthX-w_sam)/o_sam)+1;  %Number of segments in speech signal
sil = fix((Ini-Ts)/Tsh)+1;            %Number of segments in silent period
win = hamming(w_sam);

Limit = o_sam*(segs-1)+1;             %Start index of last segment

FrmIndex = 1:o_sam:Limit;             %Vector containing starting index
                                      %for each segment
ZCR_Vector = zeros(1,segs);           %Vector to hold zero crossing rate
                                      %for all segments
```

```
%Below code computes and returns zero crossing rates for all segments in
%speech sample
for t = 1:segs
    ZCRCounter = 0;
    nextIndex = (t-1)*o_sam+1;
    for r = nextIndex+1:(nextIndex+w_sam-1)
        if (x(r) >= 0) && (x(r-1) >= 0)

        elseif (x(r) >= 0) && (x(r-1) < 0)
         ZCRCounter = ZCRCounter + 1;
        elseif (x(r) < 0) && (x(r-1) < 0)

        elseif (x(r) < 0) && (x(r-1) >= 0)
         ZCRCounter = ZCRCounter + 1;
        end
    end
    ZCR_Vector(t) = ZCRCounter;
end

%Below code computes and returns frame energy for all segments in speech
%sample
Erg_Vector = zeros(1,segs);
for u = 1:segs
    nextIndex = (u-1)*o_sam+1;
    Energy = x(nextIndex:nextIndex+w_sam-1).*win;
    Erg_Vector(u) = sum(abs(Energy));
end

IMN = mean(Erg_Vector(1:sil));  %Mean silence energy (noise energy)
IMX = max(Erg_Vector);          %Maximum energy for entire utterance
I1 = 0.03 * (IMX-IMN) + IMN;    %I1 & I2 are Initial thresholds
I2 = 4 * IMN;
ITL = min(I1,I2);               %Lower energy threshold
ITU = 5 * ITL;                  %Upper energy threshold
IZC = mean(ZCR_Vector(1:sil));  %mean zero crossing rate for silence region
stdev = std(ZCR_Vector(1:sil)); %standard deviation of crossing rate for
                                %silence region
IZCT = min(ZTh,IZC+2*stdev);    %Zero crossing rate threshold
indexi = zeros(1,lengthX);      %Four single-row vectors are created
indexj = indexi;                %in these lines to facilitate computation
                                %below
indexk = indexi;
indexl = indexi;

%Search forward for frame with energy greater than ITU
for i = 1:length(Erg_Vector)
    if (Erg_Vector(i) > ITU)
```

```
        counter1 = counter1 + 1;
        indexi(counter1) = i;
    end
end
ITUs = indexi(1);

%Search further forward for frame with energy greater than ITL
for j = ITUs:-1:1
    if (Erg_Vector(j) < ITL)
        counter2 = counter2 + 1;
        indexj(counter2) = j;
    end
end
start = indexj(1)+1;

Erg_Vectorf = fliplr(Erg_Vector);%Flips round the energy vector
%Search forward for frame with energy greater than ITU
%This is equivalent to searching backward from last sample for energy > ITU
for k = 1:length(Erg_Vectorf)
    if (Erg_Vectorf(k) > ITU)
        counter3 = counter3 + 1;
        indexk(counter3) = k;
    end
end
ITUf = indexk(1);

%Search further forward for frame with energy greater than ITL
for l = ITUf:-1:1
    if (Erg_Vectorf(l) < ITL)
        counter4 = counter4 + 1;
        indexl(counter4) = l;
    end
end

finish = length(Erg_Vector)-indexl(1)+1;%Tentative finish index

%Search back from start index for crossing rates higher than IZCT

BackSearch = min(start,25);
for m = start:-1:start-BackSearch+1
    rate = ZCR_Vector(m);
    if rate > IZCT
        ZCRCountb = ZCRCountb + 1;
        realstart = m;
    end
end
if ZCRCountb > 3
```

```
    start = realstart;          %If IZCT is exceeded in more than 3 frames
                                %set start to last index where IZCT is
                                %exceeded
end

%Search forward from finish index for crossing rates higher than IZCT
FwdSearch = min(length(Erg_Vector)-finish,25);
for n = finish+1:finish+FwdSearch
    rate = ZCR_Vector(n);
    if rate > IZCT
        ZCRCountf = ZCRCountf + 1;
        realfinish = n;
    end
end
if ZCRCountf > 3
    finish = realfinish;        %If IZCT is exceeded in more than 3 frames
                                %set finish to last index where IZCT is
                                %exceeded
end

x_start = FrmIndex(start);      %actual sample index for frame 'start'
x_finish = FrmIndex(finish-1);  %actual sample index for frame 'finish'
trimmedX = x(x_start:x_finish); %Trim speech sample by start and finish
                                %indices
```

ANEXO C – MANUAL PARA GERAÇÃO DO PADRÃO DE VOZ

Procedimentos que o usuário dever realizar caso necessite acrescentar mais de 5 elocuções de treinamento para geração do padrão do locutor.

- Abrir MATLAB (versões acima de 2013) e selecionar o diretório do sistema de RAL;
- no prompt de comando do MATLAB, chamar todas as elocuções selecionadas para treinamento através do comando “audioread”;
  – exemplo para elocuções gravadas no diretório “C:\”:

```
>> eloc1 = audioread('C:\elocucao1.wav');
>> eloc2 = audioread('C:\elocucao2.wav');
...
>> elocN = audioread('C:\elocucaoN.wav');
```

- editar algoritmo **gerPadraoEdit** (disponível na pasta principal do sistema de RAL proposto contido no CD anexo) da seguinte forma:

  a) acrescentar quantidade de entradas na função de acordo com o número N de elocuções de treinamento;
  – exemplo:

```
function [padrao]= gerPadraoEdit(x1,x2,x3,x4,x5,x6,...,xN)
```

  b) acrescentar linhas de códigos para a chamada do algoritmo **vetorCaract**, também de acordo com o número N de elocuções de treinamento;
  – exemplo:

```
vetorCaract(x1);
ans1=ans;
vetorCaract(x2);
ans2=ans;
vetorCaract(x3);
ans3=ans;

...

vetorCaract(xN)
ansN=ans;
```

c) acrescentar linhas de códigos para obter o padrão de acordo com o número N de elocuções de treinamento;
– exemplo:

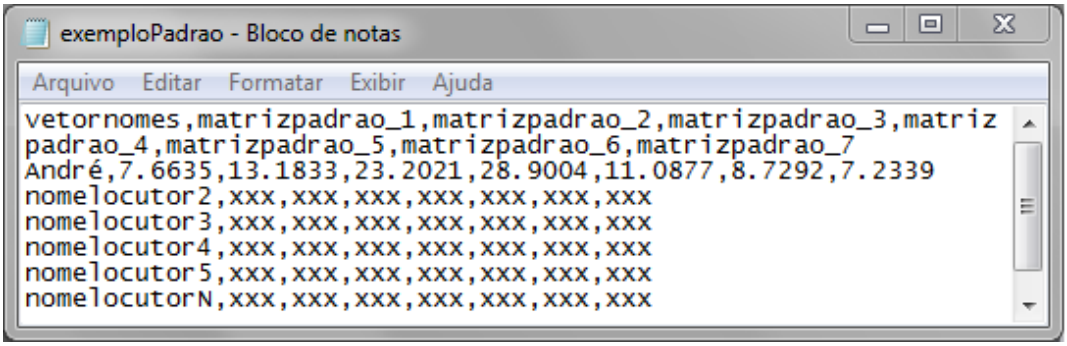
```
padrao = (ans1+ans2+ans3+ans4+ans5+...+ansN)/N
```

- salvar algoritmo **gerPadraoEdit**;
- No prompt de comando do MATLAB, chamar o algoritmo **gerPadraoEdit** da seguinte forma:

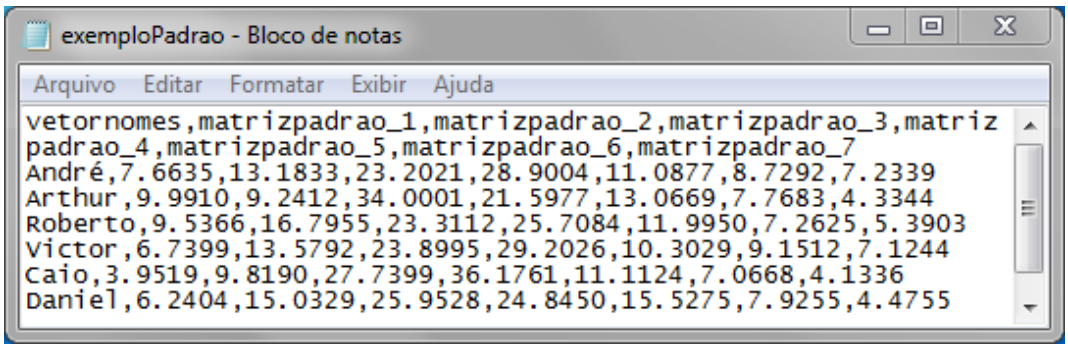
```
>> gerPadraoEdit(eloc1,eloc2,eloc3,eloc4,eloc5,...,elocN)
```

- Copiar os valores gerados pelo sistema e substituir nas linhas do arquivo *exemploPadrao.txt*, separados por vírgula, de acordo com o exemplo abaixo. Renomear, também, "nomedolocutor1" para o nome do locutor detentor do padrão;
– exemplo (locutor1 = André):

```
ans =
  7.6635   13.1833   23.2021   28.9004   11.0877   8.7292   7.2339
```



- Fazer o mesmo procedimento para os demais locutores, completando as outras linhas do arquivo *exemploPadrao.txt*;
– exemplo para 6 locutores (André, Arthur, Roberto, Victor, Caio e Daniel):



- Salvar arquivo *exemploPadrao.txt* com o nome que desejar.

Ao finalizar esses procedimentos, o arquivo conterá todos os padrões de voz dos N locutores, e estará habilitado para ser alimentado no software **RecLoc**.

ANEXO D – CD

Este anexo inclui um CD contendo:

- Versão em *pdf* desta dissertação;
- Algoritmos do *vocoder* proposto;
- Algoritmos do sistema de RAL proposto;
- Arquivos de voz utilizados nos testes.
