**UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE FÍSICA DE SÃO CARLOS
DEPARTAMENTO DE FÍSICA E INFORMÁTICA**

*SPIKELET: UMA NOVA TRANSFORMADA WAVELET APLICADA
AO RECONHECIMENTO DIGITAL DE PADRÕES, EM TEMPO
REAL, DE SPIKES E OVERLAPS EM SINAIS
NEUROFISIOLÓGICOS DO CAMPO VISUAL DA MOSCA*

Rodrigo Capobianco Guido

Tese apresentada ao Instituto
de Física de São Carlos da
Universidade de São Paulo, para
obtenção do título de Doutor em
Ciências: Física Aplicada - opção
Física Computacional

Orientador: Prof. Dr. Jan Frans Willem Slaets

**São Carlos – SP – Brasil
11/2003**

FOLHA DE APROVAÇÃO

Membros da Comissão Julgadora da Tese de Doutorado de Rodrigo Capobianco Guido, apresentada ao Instituto de Física de São Carlos, da Universidade de São Paulo, em 27 de novembro de 2003.

Comissão Julgadora:

Prof. Dr. Jan Frans Willem Slaets – IFSC/USP

Prof. Dr. Roland Köberle – IFSC/USP

Prof. Dr. Antônio Carlos Roque da Silva Filho – FFCLRP/USP

Prof. Dr. José Márcio Machado – IBILCE/UNESP

Profa. Dra. Eliana Xavier Linhares de Andrade – IBILCE/UNESP

" Nunca ande pelo caminho previamente traçado, pois ele conduz apenas até onde os outros já foram..."

Alexandre Graham Bell.

Dedico esta tese ao Prof. Jan,
meu orientador, pela oportunidade
concedida, a qual serei eternamente
grato.

AGRADECIMENTOS

Em primeiro lugar, ao Prof. Dr. Jan Frans Willem Slaets, orientador do presente trabalho, pela oportunidade que me foi atribuída, e que não foi fácil de ser conquistada.

Ao Tec. Lírio Onofre de Almeida, pelo apoio e pela ajuda prestada em inúmeros detalhes do desenvolvimento deste trabalho.

Aos demais integrantes do Laboratório Dipterlab, em especial ao Prof. Dr. Roland Köberle, pelas sugestões para melhoria do trabalho.

A Wladerez Ap. Gounella Caiado e a Cristiane Gomes L. Estella, do serviço de Pós-Graduação, pela dedicação em resolver os problemas relacionados com meu Doutorado.

Ao Prof. Dr. José Márcio Machado do Departamento de Computação da UNESP de São José do Rio Preto, pelo artigo sobre as *Wavelets* de Daubechies, que foi fundamental para o início do meu trabalho.

Ao Prof. Dr. Alberto Tannús, do Grupo de Ressonância Magnética do IFSC – USP e ao Prof. Dr. Adilson Gonzaga da Escola de Engenharia de São Carlos – USP, pelas sugestões dadas para complementar o trabalho.

Aos Profs. Dr. José Carmelo Interlando da Universidade de Notre Dame, em Indiana – EUA, Dr. Aledir Silveira Pereira e Dra. Eliana X. Linhares de Andrade, do Departamento de Computação da UNESP de São José do Rio Preto e ao Prof. Dr. Euclides Maréga Jr do Grupo de Semicondutores do IFSC-USP, pelo auxílio inicial prestado.

A minha família, pela colaboração e, enfim, a todos os companheiros aqui do IFSC, que sempre me receberam com muito carinho.

ÍNDICE

Introdução: Motivação.....	13
Capítulo I: Revisão Bibliográfica: Ferramentas para o Processamento Digital de Sinais Neurofisiológicos, <i>Wavelets</i> e Técnicas de Classificação de <i>Spikes</i>	17
1.1 - Sinais e Sistemas Neurofisiológicos.....	17
1.2 - Processamento Analógico X Digital de Sinais Neurofisiológicos....	19
1.3 - Filtros Digitais.....	21
1.4 – A Transformada Discreta de Fourier (DFT) e a Transformada Rápida de Fourier (FFT).....	23
1.5 - A Transformada Z.....	24
1.6 - Função de Transferência.....	25
1.7 – Definições e Teoremas.....	26
1.7.1 - Convolução Discreta.....	26
1.7.2 - Sinais Estacionários e Não-Estacionários.....	27
1.7.3 - Energia de um Sinal.....	28
1.7.4 - Teorema da Amostragem.....	28
1.7.5 - O Princípio da Incerteza.....	29
1.7.6 - Sub-Amostragem (<i>downsampling</i>).....	29
1.7.7 - Causalidade e Estabilidade.....	29
1.8 - A Transformada de Fourier de Tempo Reduzido (STFT).....	30
1.9 - A Transformada <i>Wavelet</i> Discreta e a Análise MRA.....	30
1.10 - Resumo das Principais Técnicas para Detecção de <i>Spikes</i>	36
Capítulo II: Criação da SPIKELET.....	40
2.1 - Obtenção do sinal padrão de interesse.....	40
2.2 - Técnica Proposta.....	43
2.3 - Função <i>Scaling</i>	46
2.4 - Função <i>Wavelet</i>	49
2.5 - Análise dos Momentos Nulos da SPIKELET-4.....	50
2.6 - Estabilidade e Causalidade da SPIKELET-4.....	50
2.7 – Considerações.....	52
Capítulo III: Algoritmo Proposto para Detecção e Classificação dos <i>Spikes</i> e <i>Overlaps</i>	59
3.1 – Descrição.....	60
3.2 – Considerações.....	66
Capítulo IV: Processamento <i>on the fly</i> do Sinal da Mosca com o DSP EZ-Kit Lite 21065L SHARC - Analog Devices.....	68
Capítulo V: Testes, Resultados, Conclusões e Idéias Futuras.....	72

Apêndice I: Código Fonte em Linguagem C++ para o cômputo da SPIKELET-4.....	89
Apêndice II: Código Fonte em Linguagem C++ para a Análise e Classificação dos <i>Spikes</i> e <i>Overlaps</i>	94
Apêndice III: Funcionamento Básico do Sistema Visual da Mosca.....	104
Apêndice IV: Fotos dos experimentos realizados no Laboratório Dipterolab – IFSC – USP.....	106
Apêndice V: Coeficientes dos filtros utilizados nas experiências.....	109
Apêndice VI: O formato WAV de arquivos de audio.....	110
Apêndice VII: O formato BMP de arquivos de imagem.....	111
Apêndice VIII: Código Fonte da Implementação <i>on the fly</i> do Algoritmo de Detecção e Classificação no DSP EZ-Kit Lite 21065L SHARC - Analog Devices.....	112
Apêndice IX: Trabalhos Publicados e Apresentados.....	126
Referências	127

ÍNDICE DE FIGURAS

FIG_I-1: descrição básica do experimento associado ao presente trabalho.....	14
FIG_I-2: foto do experimento com a mosca.....	14
FIG1-1: exemplo das propriedades de um filtro digital passa-baixa.....	22
FIG1-2: decomposição de um sinal discreto em vários níveis de resolução com uma transformada <i>wavelet</i>	32
FIG1-3: <i>Dyadic Grid</i>	36
FIG2-1: sinal contendo uma série de <i>spikes</i> e <i>overlaps</i>	42
FIG2-2: FFT do sinal da Figura FIG2-1.....	42
FIG2-3: 78 <i>spikes</i> utilizados na obtenção do <i>spike</i> médio padrão.....	42
FIG2-4: <i>spike</i> médio padrão.....	42
FIG2-5: <i>spike</i> e <i>overlaps</i> típicos.....	43
FIG2-6: esquema de construção da SPIKELET-4.....	44
FIG2-7: Diagrama de Bode do Filtro SPIKELET-4.....	52
FIG2-8: <i>scaling</i> – SPIKELET4.....	55
FIG2-9: <i>wavelet</i> – SPIKELET4.....	55
FIG2-10: <i>scaling</i> – DAUBECHIES4.....	55
FIG2-11: <i>wavelet</i> – DAUBECHIES4.....	55
FIG2-12: filtro Passa-Baixa-SPIKELET4.....	56
FIG2-13: filtro Passa-Alta-SPIKELET4.....	56
FIG2-14: filtro Passa-Baixa-DAUBECHIES4.....	56
FIG2-15: filtro Passa-Alta-DAUBECHIES4.....	56
FIG2-16: resposta em frequência dos filtros SPIKELET-4 em escala logarítmica.....	57
FIG2-17: reação da SPIKELET-4 a um <i>spike</i> de 32 amostras.....	58
FIG3-1: fluxograma do algoritmo proposto para detecção e classificação dos <i>spikes</i> e <i>overlaps</i> c/ uso da transformada SPIKELET-4.....	60
FIG3-2: testes de inversão de sinal entre as amostras 1 e 2, 4 e 5, 5 e 6, 6 e 7 na transformada SPIKELET4 de um possível <i>spike</i> ou <i>overlap</i>	63
FIG3-3: <i>spike</i> e filtros passa-baixa aplicados nele.....	65
FIG3-4: <i>overlap</i> e filtros passa-baixa aplicados nele.....	65
FIG3-5: decomposição do <i>spike</i>	65
FIG3-6: decomposição do <i>overlap</i>	65
FIG3-7: decomposição do sinal de 32 amostras (possível <i>spike/overlap</i>) com a Transformada SPIKELET-4 nível 3.....	66
FIG4-1: diagrama de blocos do DSP EZ-Kit Lite 21065L Sharc da Analog Devices.....	68
FIG4-2: tela do ambiente de desenvolvimento integrado do DSP.....	70
FIG5-1: exemplos de sinais detectados pelo algoritmo associado a SPIKELET-4 (1).....	75
FIG5-2: exemplos de sinais detectados pelo algoritmo associado a SPIKELET-4 (2).....	76

FIG5-3: eficiência do algoritmo associado a SPIKELET-4 para detecção de um <i>spike</i> “irregular”	77
FIG5-4: transformada de Daubechies-4 do <i>spike</i> S.....	78
FIG5-5: transformada SPIKELET-4 do <i>spike</i> S.....	78
FIG5-6: transformada de Daubechies-4 do <i>overlap</i> OS.....	78
FIG5-7: transformada SPIKELET-4 do <i>overlap</i> OS.....	78
FIG5-8: amostras 0 a 3 da Transformada Daubechies-4 do <i>spike</i> S.....	79
FIG5-9: amostras 0 a 3 da Transformada SPIKELET-4 do <i>spike</i> S.....	79
FIG5-10: amostras 0 a 3 da Transformada Daubechies-4 do <i>overlap</i> OS....	79
FIG5-11: amostras 0 a 3 da Transformada SPIKELET4 do <i>overlap</i> OS.....	79
FIG5-12: amostras 4 a 7 da Transformada Daubechies-4 do <i>spike</i> S.....	80
FIG5-13: amostras 4 a 7 da Transformada SPIKELET-4 do <i>spike</i> S.....	80
FIG5-14: amostras 4 a 7 da Transformada Daubechies-4 do <i>overlap</i> OS....	80
FIG5-15: amostras 4 a 7 da Transformada SPIKELET4 do <i>overlap</i> OS.....	80
FIG5-16: aplicação do algoritmo com outra transformada <i>wavelet</i> (1).....	81
FIG5-17: aplicação do algoritmo com outra transformada <i>wavelet</i> (2).....	82
FIG5-18: aplicação do algoritmo com outra transformada <i>wavelet</i> (3).....	83
FIG5-19: aplicação do algoritmo com outra transformada <i>wavelet</i> (4).....	84
FIG5-20: aplicação do algoritmo com outra transformada <i>wavelet</i> (5).....	85
FIG5-21: aplicação do algoritmo com outra transformada <i>wavelet</i> (6).....	86
FIG5-22: aplicação do algoritmo com outra transformada <i>wavelet</i> (7).....	87
FIG5-23: aplicação do algoritmo com outra transformada <i>wavelet</i> (8).....	88
FIG-AP3-1: cérebro da mosca, com destaque (acima à direita) para o percurso da informação a partir do olho.....	104
FIG-AP3-2: vista ampliada da placa lobular.....	105
FIG-AP3-3: padrão dos traços verticais que sensibilizam o neurônio H1 da mosca.....	105
FIG-AP4-1: vista ampliada da mosca e do eletrodo durante um experimento.....	106
FIG-AP4-2: osciloscópio com um <i>spike</i> capturado em tempo-real.....	107
FIG-AP4-3: equipamentos utilizados no Laboratório Dipterlab preparado para um experimento.....	108

ÍNDICE DE TABELAS

Tabela CP2-T1: dados do <i>spike</i> médio padrão.....	41
Tabela CP3-T1: coeficientes do filtro SPIKELET-4.....	46
Tabela AP6-T1: formato WAV de audio.....	110
Tabela AP7-T1: cabeçalho do formato BMP.....	111

GLOSSÁRIO DE ABREVIATURAS, SÍMBOLOS E OUTROS TERMOS UTILIZADOS

*	convolução
CODEC	Codificador e Decodificador de audio
DFT	<i>Discrete Fourier Transform</i> (Transformada Discreta de Fourier)
DSP	<i>Digital Signal Processor</i> (Processador Digital de Sinais)
DWT	<i>Discrete Wavelet Transform</i> (Transformada Discreta <i>Wavelet</i>)
FFT	<i>Fast Fourier Transform</i> (Transformada Rápida de Fourier)
FIR	<i>Finite Impulse Response</i> (tipo de filtro digital não recursivo)
H1	Neurônio do cérebro da mosca de ordem <i>diptera</i> , sensível aos estímulos visuais verticais
IFSC-USP	Instituto de Física de São Carlos da Universidade de São Paulo
IIR	<i>Infinite Impulse Response</i> (tipo de filtro digital recursivo)
I/O	<i>Input / Output</i> (Entrada / Saída)
Khz	Kilo hertz = 1×10^3 hertz (unidade de medida de frequência)
LTI	<i>Linear Time Invariant</i> (Linear e Invariante no Tempo)
MIPS	milhões de instruções por segundo (unidade de medida de processamento)
MFLOPS	Mega Flops = milhões de operações de ponto flutuante por segundo (unidade de medida de processamento)
Mhz	Mega hertz = 1×10^6 hertz (unidade de medida de frequência)
MRA	<i>Multiresolution Analysis</i> (Análise Multi Resolutiva)
ns	nano segundo = 1×10^{-9} segundos
overlap	<i>spike</i> sobreposto a outro <i>spike</i>
QMF	<i>Quadrature Mirror Filter</i> (Filtro de Quadratura Espelho)
<i>spike</i>	"pico" (sinal típico proveniente do neurônio H1 da mosca de ordem <i>diptera</i>)
SISD	arquitetura tipo <i>Single Instruction Stream/Single Data Stream</i>
SMD	<i>Surface Mounted Device</i> (componente eletrônico para montagem em superfície)
STFT	<i>Short Time Fourier Transform</i> (Transformada de Fourier de Tempo Reduzido)
threshold	limiar
X(z)	Transformada Z de x
μs	micro segundo = 1×10^{-6} segundos
Z	corpo dos números inteiros positivos
$\mathbb{R}$	corpo dos números reais
$\vec{v}$	vetor v
$\vec{v} \cdot \vec{u}$	produto interno (no caso, escalar) dos vetores v e u

RESUMO

A presente tese descreve a construção de uma nova transformada *wavelet*, aqui chamada de SPIKELET, que, combinada com um algoritmo proposto, é aplicada no reconhecimento computacional de padrões em *spikes* (picos) e *spikes* sobrepostos (*overlaps*) encontrados no sinal digitalizado correspondente às reações do neurônio H1 do cérebro de uma mosca de ordem *Diptera*, que é sensível aos estímulos visuais do meio externo. O algoritmo fornece, além do formato do sinal encontrado, o "instante" em que ele ocorreu, sendo que a implementação é feita, inclusive, em tempo-real, com o uso de um DSP.

Palavras-chave: reconhecimento de padrões em *spikes* e *overlaps*, *wavelets*, filtros digitais, processamento digital de sinais neurofisiológicos, algoritmos computacionais

ABSTRACT

This thesis describes the construction of a new wavelet transform, that is called SPIKELET, which is used together with a proposed algorithm, for spikes and overlaps pattern recognition, in a digitalized signal corresponding to the H1 visual neuron action potential from a *Diptera's* fly brain. The algorithm provides both the shape of the identified signal and the "instant" of time it happened. The implementation is also done in real time, using a DSP.

Key-Words: spike and overlap pattern recognition, wavelets, digital filters, neurophysiological digital signal processing, computational algorithms

Introdução

MOTIVAÇÃO

O presente trabalho propõe a construção de uma transformada *wavelet*, chamada SPIKELET, que quando combinada com um algoritmo desenvolvido para atuar em conjunto com ela, permite identificar e classificar, em tempo-real, *spikes* (picos) e *overlaps* (picos sobrepostos) presentes num sinal digitalizado, não estacionário, correspondente às reações do neurônio H1 de uma mosca de ordem *diptera*, aos estímulos visuais verticais recebidos do meio externo. A implementação do algoritmo se deu com sucesso, inclusive em tempo real, usando um DSP.

O experimento científico associado ao presente trabalho, descrito com o auxílio da figura FIG_I-1 abaixo, e tema de pesquisa do **Laboratório Dipteralab do IFSC – USP** [DIP1], consiste num monitor de vídeo que exibe um padrão de traços verticais que se movimentam horizontalmente, sensibilizando o neurônio da mosca, denominado H1. Com o auxílio de um microscópio, a mosca é aprisionada na frente do monitor de vídeo e um único eletrodo é afixado, de maneira extra celular, próximo ao neurônio H1 de seu cérebro, que é o responsável pela geração dos sinais a serem analisados. Os sinais são então recolhidos e processados pelo algoritmo. A figura FIG_I-2 mostra o experimento com a mosca e o eletrodo. O artigo [REP1] traz um estudo interessante relacionado ao tema.

Após uma vasta revisão da literatura sobre algoritmos de processamento de sinais, muitas idéias foram se conectando. A opção por processar e classificar os sinais com o uso de uma transformada *wavelet*, se deu pela necessidade do uso de uma técnica do tipo tempo-freqüência, que fosse capaz de identificar um *spike* ou *overlap*, precisar a sua localização e ainda discriminar formatos. A partir daí, foi criada então a transformada SPIKELET-4, que possibilitou uma análise do tipo

Figura FIG_1-1: Descrição básica do experimento associado ao presente trabalho

Figura FIG_1-2: Foto do experimento com a mosca

tempo-freqüência-formato.

Atualmente, é crescente a utilização de DSP para implementação, em tempo real, dos algoritmos de processamento digital de sinais [PDS1]. No presente trabalho, isso implicou numa melhora substancial no sistema de discriminador analógico utilizado anteriormente para identificar os *spikes* e *overlaps*.

Conforme será caracterizado no decorrer desta tese, os resultados obtidos são considerados muito satisfatórios e não se encontram relatados anteriormente, com o uso de um algoritmo de baixa ordem de complexidade computacional e determinístico, como o aqui obtido. Os detalhes estão organizados da seguinte forma:

- O Capítulo I faz uma breve revisão dos conceitos e ferramentas para o Processamento Digital de Sinais Neurofisiológicos, incluindo a transformada *wavelet*, bem como, das técnicas de classificação de *spikes* descritas na literatura.

- No Capítulo II encontra-se a descrição completa de como foi criada a transformada *wavelet* SPIKELET-4 e diversos resultados associados.

- Já o Capítulo III descreve o algoritmo que aplica a SPIKELET-4 para detecção e classificação dos *spikes* e *overlaps* utilizando a linguagem de programação C++, em um computador pessoal comum, em ambiente LINUX. Neste caso, a implementação não se faz em tempo real, sendo que o sinal processado foi recolhido e digitalizado para pós-processamento.

O Capítulo IV trata da implementação do algoritmo de detecção e classificação, em tempo real, usando um DSP SHARC 21065L da *Analog Devices Inc.*

- Finalmente, o Capítulo V descreve com mais detalhes, os resultados alcançados na classificação dos sinais quando a SPIKELET-4 se associa ao algoritmo proposto, implementado em tempo real ou não. Além disso, uma série de testes da aplicação do algoritmo com outras *wavelets* são apresentados. Diversas comparações e conclusões são fornecidas e idéias futuras são propostas.

- Nos Apêndices encontram-se os *softwares* que implementam a técnica proposta, dados complementares e fotos dos experimentos.

Capítulo I

REVISÃO BIBLIOGRÁFICA: FERRAMENTAS PARA O PROCESSAMENTO DIGITAL DE SINAIS NEUROFISIOLÓGICOS, *WAVELETS* E TÉCNICAS DE CLASSIFICAÇÃO DE *SPIKES*

Este capítulo faz uma revisão básica dos conceitos e das técnicas para o Processamento Digital de Sinais Neurofisiológicos, que são comentadas e/ou utilizadas neste trabalho, mais especificamente a DFT/FFT, a transformada wavelet, bem como uma breve abordagem dos modelos de classificação de spikes.

1.1 - Sinais e Sistemas Neurofisiológicos

Quer possamos nos dar conta do fato ou não, os sinais estão presentes em todos os momentos de nossas vidas, principalmente os neurofisiológicos. As imagens que nos chegam ao cérebro através de nossos olhos e nervos ópticos, os sons que ouvimos, os sinais de um eletrocardiograma (ECG) ou eletroencefalograma (EEG) de um ser humano são alguns exemplos desses sinais.

Na prática, enumerar tudo aquilo que constitui um sinal é tarefa impossível. De acordo com [SIN1], um sinal pode ser definido como uma função de uma ou mais variáveis que vincula informações sobre um fenômeno físico. O presente trabalho descreve a análise de um sinal unidimensional, ou seja, um sinal que é função de uma única variável, no caso o tempo, e que é gerado pelo neurônio H1 da mosca de ordem *diptera*, o qual é responsável pelas respostas aos estímulos visuais verticais externos detectados.

Em todos os exemplos comentados acima existe sempre, de maneira implícita, um sistema associado. Por exemplo, o sistema associado à interpretação de uma imagem por um inseto é o conjunto formado pelo olho, neurônios associados, nervo óptico e cérebro. Da mesma forma, o som faz uso do sistema formado pelo ouvido, cérebro e demais órgãos para sua interpretação pelo ser humano.

Obviamente esses sinais podem fazer uso de um sistema biológico e/ou artificial - no caso computadorizado - para serem interpretados e processados, como é o caso, por exemplo, dos sinais que processamos no presente trabalho.

Já um sistema é definido como uma entidade que manipula um ou mais sinais, processando-os e produzindo, assim, outros sinais que representam, para nós, determinada informação.

De acordo com o que podemos vivenciar facilmente na prática, não existe um sistema de propósito geral. Os sistemas neurofisiológicos recebem entradas, em nosso caso do meio externo, e produzem saídas correspondentes a esses estímulos, sendo que a relação entrada-saída é característica de cada sistema neurofisiológico. O sistema biológico da mosca utilizado na presente tese recebe, como entrada, estímulos visuais gerados por um monitor de vídeo e apresenta um sinal elétrico como resposta a tais estímulos. O sistema computadorizado aqui associado recebe, como entrada, o sinal elétrico em questão, obtido da saída de um outro sistema de apoio e captura, e extrai dele as informações procuradas, que dizem respeito às respostas de determinado neurônio aos estímulos gerados, que estão no formato de *spikes / overlaps*.

Um conceito muito importante quando falamos em sistemas é o de um Sistema Linear e Invariante no Tempo (LTI), que é aquele que satisfaz às condições:

- deslocar a entrada por uma determinada constante de

tempo implica no mesmo deslocamento na saída.

- a multiplicação da entrada do sistema por uma constante implica na saída multiplicada pela mesma constante.
- a soma de dois ou mais sinais na entrada de um sistema implica na soma das saídas individuais.

No presente trabalho, esses conceitos são fundamentais para determinação, com precisão, do “instante” de tempo onde o *spike / overlap* ocorreu.

1.2 - Processamento Analógico X Digital de Sinais Neurofisiológicos

Durante muitos anos, todo o processamento de sinais era realizado unicamente de modo analógico, ou seja, era realizada uma abordagem baseada em tempo contínuo. Até hoje, ainda é viável realizar o processamento analógico de alguns sinais. Como diz o próprio nome, o sistema de processamento analógico de sinais faz uso de circuitos eletrônicos analógicos. Em contrapartida, os sistemas de processamento digital de sinais recorrem basicamente ao uso de um processador capaz de realizar operações aritméticas [SIN1].

A principal vantagem do processamento analógico sobre o digital é a resposta precisa, e em tempo real, aos estímulos recebidos, através dos circuitos eletrônicos analógicos, devido à ausência de quantização de amplitude e discretização no tempo. Já a abordagem digital, como requer computações aritméticas, não pode garantir operação em tempo real.

Em contrapartida, a abordagem digital possui três vantagens principais:

- Imunidade: não existe a influência de agentes que distorcem os valores de componentes passivos nos circuitos eletrônicos,

tais como resistores e capacitores, que com o tempo ou dependendo de fatores externos (umidade / temperatura / envelhecimento) podem ter seus valores alterados modificando, assim, o comportamento global dos sistemas de processamento. Tal fato era muito comum nas centrais telefônicas antigas que utilizavam filtros para multiplexar a banda de comunicação que constantemente sofria interferências de outros canais adjacentes. Os únicos erros resultantes do processamento digital são devidos à precisão na representação numérica.

- Flexibilidade: obviamente, quando falamos em processadores e memórias, ou seja, computadores, temos em mente que todo o processamento pode ser controlado via *software* e que, portanto, qualquer modificação que se queira fazer no processamento não implica em utilizar ferramentas para manipular valores de componentes nos circuitos eletrônicos.

- Repetitividade: uma operação de processamento digital de sinais pode ser repetida diversas vezes de maneira exata enquanto que nos sistemas analógicos, podem existir variações devido à falta de imunidade comentada anteriormente.

Uma questão a observar no processamento de sinais biológicos é a interferência causada por artefatos. Um artefato é na verdade um sinal espúrio que não pertence ao sinal biológico original. Pode-se exemplificar facilmente:

- artefatos instrumentais: perturbações de origem eletromagnética ou do próprio instrumental utilizado para obtenção dos sinais. Um exemplo clássico é a interferência de 60Hz da rede elétrica, captada por componentes demasiadamente sensíveis. No experimento em questão, existe

um filtro eletrônico analógico (passa-banda 300 Hz a 7Khz – 36dB - *Butterworth* [DIG1] de 3º ordem) no sistema de captura dos sinais da mosca para diminuir tal interferência.

- artefatos de análise: são devidos aos erros de representação numérica, que acabam por introduzir eventos estranhos e incompatíveis ao sinal original.

- artefatos biológicos: nos casos em que um sinal biológico contamina outro sinal biológico. Por exemplo, no caso da mosca, freqüentemente os sinais de dois neurônios se misturam, provocando *overlaps* (sobreposições) e muitas vezes dificultam a análise de um em separado.

Uma forma de reduzir artefatos indesejados, principalmente os de origem biológica e instrumental, consiste na utilização de filtros além da boa localização do eletrodo de captação do sinal no caso do presente trabalho.

1.3 - Filtros Digitais

Um filtro digital [DSP1] é nada mais do que uma série de coeficientes com os quais se faz uma combinação linear de um sinal de entrada para obter um sinal de saída com determinadas características de frequência selecionadas. Os parâmetros mais relevantes de um filtro digital, ilustrados com a ajuda da figura FIG1-1, são:

- freqüência de corte: é a freqüência para a qual o filtro já passa a ter uma atenuação maior ou igual a -3dB (aproximadamente 70,7 %), que é o ponto onde termina a banda de passagem e inicia a banda de transição.
- freqüência de rejeição: definida neste trabalho como a freqüência para a qual o filtro já passa a ter uma atenuação maior ou igual a 95 % da atenuação máxima, que é o ponto onde termina a banda de transição e se inicia a banda de rejeição.

- banda de passagem: é a faixa de frequências anterior a frequência de corte.
- banda de transição: é a faixa de frequências que inicia no final da banda de passagem e termina no início da banda de rejeição.
- banda de rejeição: faixa de frequências posterior à frequência de rejeição.
- máxima variação de ganho na banda de passagem
- mínima atenuação na banda de rejeição
- tipo: FIR ou IIR, conforme descrito na próxima seção.
- função: passa-baixa, passa-alta, passa-faixa e rejeita-faixa.
- ordem: número de polos da função de transferência do filtro. Um filtro digital com $N+1$ coeficientes possui ordem N . À medida que a ordem do filtro aumenta, sua resposta em frequência fica mais próxima da ideal (a banda de transição é mais estreita).
- fase: linear (atraso constante da saída para toda a faixa de frequência) ou não linear.

Figura FIG1-1: exemplo das propriedades de um filtro digital passa-baixa.

A unidade de medida *dB* utilizada acima, chamada *decibel*, corresponde a uma unidade logarítmica utilizada para facilitar a mensuração de alguns parâmetros dos filtros digitais: $dB = 20 \log(\text{ganho_do_filtro_em_amplitude})$.

Caso o sinal presente na saída do filtro possua, para determinadas frequências, amplitude menor do que o sinal de entrada, o ganho do filtro

será menor do que 1 e, portanto, a unidade *dB* assumirá valor negativo representando uma perda. Caso os sinais na saída e na entrada sejam idênticos para alguma frequência, o ganho do filtro será 1, correspondendo a 0 *dB* e, caso o sinal da saída seja superior ao da entrada, a unidade *dB* assumirá um valor positivo.

No que se refere à presente obra, foram utilizados filtros digitais com o apoio do *software* Orign v.6.1 [ORI1] e Octave [OCT1] para analisar diversos comportamentos dos sinais. Mais especificamente, foram utilizados filtros digitais FFT passa-faixa, passa baixa e passa-alta [DIG1] para analisar os espectros de frequências que compõem os *spikes* e os *overlaps*.

Além disso, como a transformada *wavelet* constitui uma classe de filtros digitais, pode-se dizer que o trabalho é inteiramente baseado em tais filtros. O processo de convolução, que é na verdade o processo de filtragem digital utilizado aqui, é revisto a seguir.

Um outro conceito importante no escopo dos filtros digitais é o dos filtros QMF, que correspondem a um par de filtros, um passa-alta (*h*) e um passa-baixa (*g*), que tem seus coeficientes definidos da forma

$$g_i = (-1)^i h_{(n-i-1)} \quad , \quad (1-1)$$

onde *n* é o comprimento do filtro e *i* é o *i*-ésimo coeficiente. Portanto, as suas respostas em frequência são exatamente o “espelho vertical” da outra. A SPIKELET-4 possui os filtros da forma QMF.

1.4 - A Transformada Discreta de Fourier (DFT) e a Transformada Rápida de Fourier (FFT)

A DFT é designada para o tratamento de sinais de tempo discreto, sendo uma ferramenta já muito difundida [SIN1], [OPE1], [EQU1] e que é dada

por:

$$F[k] = \sum_{n=0}^{N-1} f[n] e^{-j \frac{2\pi n k}{N}} \quad (1.2)$$

onde $j = \sqrt{-1}$, n é a n -ésima amostra do sinal f , que está no domínio do tempo, k é a k -ésima amostra do sinal F , que está no domínio da frequência, e N é o número de pontos do sinal f . Sua inversa é dada por:

$$f[n] = \frac{1}{N} \sum_{k=0}^{N-1} F[k] e^{j \frac{2\pi n k}{N}} \quad (1.3)$$

Uma observação fundamental aqui é a seguinte: a DFT diz exatamente quais são as frequências presentes no sinal. Entretanto, não contém nenhuma informação sobre onde essas frequências estão localizadas dentro do sinal. Sendo assim, para um sinal não estacionário, como é o caso do sinal processado pelo algoritmo descrito neste trabalho, a DFT não relaciona faixas de frequências com suas localizações e, portanto, terá uso restrito.

Hoje em dia, praticamente todos os algoritmos computacionais utilizam versões rápidas da DFT, conhecidas como Transformada Rápida de Fourier (FFT) [PRO1]. Existe uma série de algoritmos FFT que reduzem a complexidade computacional da DFT, de ordem quadrática, para ordem logarítmica. Os algoritmos FFT, por serem amplamente descritos na literatura e terem uso restrito neste trabalho, não serão descritos aqui.

1.5 - A Transformada Z

No presente trabalho, a Transformada Z [SIN1], expressa através da equação (1-4) abaixo, será utilizada para o cálculo da Função de Transferência do filtro criado, sendo utilizada como uma forma de representar um sinal discreto no domínio da frequência.

$$X(z) = \sum_{n=0}^{N-1} x_n z^{-n} \quad (1-4)$$

onde $X(z)$ é a transformada Z do sinal discreto x , n é a n -ésima amostra de x , N é a quantidade de pontos de x e z é uma variável complexa. Duas propriedades da Transformada Z que serão utilizadas no presente trabalho são a linearidade:

$$y_n = k x_n, k \in \mathbb{R} \Rightarrow Y(z) = \sum_{n=0}^{N-1} k x_n z^{-n} = k \sum_{n=0}^{N-1} x_n z^{-n} = k X(z) \quad (1-5)$$

$$y_n = x_n + t_n \Rightarrow Y(z) = \sum_{n=0}^{N-1} (x_n + t_n) z^{-n} = \sum_{n=0}^{N-1} x_n z^{-n} + \sum_{n=0}^{N-1} t_n z^{-n} = X(z) + T(z) \quad (1-6)$$

e, também, o deslocamento no tempo:

$$X_{esq}(z) = \sum_{n=0}^{N-1} x_{n+k} z^{-n} = \sum_{n=0}^{N-1} x_{n+k} z^{-n+k-k} = \sum_{n=0}^{N-1} x_{n+k} z^{-(n+k)} z^k = z^k X(z) \quad (1-7)$$

$$X_{dir}(z) = \sum_{n=0}^{N-1} x_{n-k} z^{-n} = \sum_{n=0}^{N-1} x_{n-k} z^{-n+k-k} = \sum_{n=0}^{N-1} x_{n-k} z^{-(n-k)} z^{-k} = z^{-k} X(z) \quad (1-8)$$

1.6 - Função de Transferência

Se h_n é a resposta ao impulso de um sistema LTI, de tal forma que $y_n = h_n * x_n$, então $H(z)$ é conhecida como função de transferência do sistema.

$$y_n = h_n * x_n \rightarrow Y(z) = H(z) X(z) \rightarrow \frac{Y(z)}{X(z)} = H(z) \quad (1.9)$$

As raízes de $Y(z)$ são chamadas zeros da função de transferência

(valores onde ela se torna zero) e as raízes de $X(z)$ são chamadas pólos da função de transferência (valores onde ela não existe ou tende para o infinito). Quando os pólos da função têm módulo no máximo 1, o sistema é estável e causal e quando os zeros da função têm módulo no máximo 1, o sistema inverso é estável e causal. O Diagrama de Bode, utilizado para a análise de estabilidade e causalidade de um sistema, é o gráfico circular de raio unitário onde se traçam os pólos e os zeros da função de transferência. No presente trabalho, a Função de Transferência e o Diagrama de Bode são utilizados no capítulo II para análise do filtro criado.

1.7– Definições e Teoremas

1.7.1 - Convolução Discreta

A convolução discreta não recursiva [FIL1] é o tipo mais simples de sistema de filtro digital que temos. Estes filtros são chamados de filtros digitais não-recursivos, também conhecidos como filtros FIR (*finite-impulse-response*), ou seja, são aqueles sistemas em que uma determinada amostra da saída não se define recursivamente com base em amostras anteriores da saída.

Um sistema para realizar a convolução discreta não recursiva de sinais corresponde a um sistema que realiza uma combinação linear do sinal de entrada x com determinados coeficientes h , como está expresso abaixo na equação (1-10):

$$x[n] * h[n] = \sum_{k=0}^{N-1} h[k] x[n-k] , \quad (1-10)$$

onde n e k representam amostras dos sinais h e x . Existe também a possibilidade de realizar a convolução discreta recursiva, obtendo assim uma classe um pouco mais específica de filtros chamados filtros digitais recursivos, também conhecidos como filtros IIR (*infinite-impulse-response*), onde uma determinada amostra na saída se define em função das amostras anteriores ali presentes. Tais filtros, entretanto, não serão utilizados aqui.

O Teorema da Convolução, importante resultado da área de Sinais e Sistemas, pode ser assim expresso:

"A Transformada de Fourier da Convolução de duas funções no domínio do tempo é igual ao produto das Transformadas de Fourier de cada uma das funções e vice-versa."

Tal fato pode ser expresso pela equação (1-11), onde o símbolo * denota a convolução:

$$F(x[n] * h[n]) = X[k] H[k], \quad (1-11)$$

onde n é a n -ésima amostra dos sinais h e x , que estão no domínio do tempo, e k representa a k -ésima amostra dos sinais H e X que estão no domínio da frequência. Este resultado é muito importante [ANA1] pois permite operar com as amostras de um filtro e de um sinal, no domínio do tempo, através da convolução, que basicamente é o que se faz na transformada *wavelet*.

Qualquer sistema LTI pode ser completamente descrito por sua resposta ao impulso. Para um sistema LTI discreto, a saída corresponde à convolução entre a resposta ao impulso, $h[n]$ e o sinal de entrada $x[n]$, como descrito na equação (1-11). O sistema proposto neste trabalho vai apresentar um determinado sinal na saída, que resulta de algumas convoluções seguidas de sub-amostragem por 2, entre o sinal de entrada com sua resposta ao impulso.

1.7.2 - Sinais Estacionários e Não-Estacionários

Um conceito importante a ser lembrado é o de sinal estacionário [WAV1] [SIN1], que corresponde ao sinal que tem os mesmos componentes de frequência distribuídos ao longo de todo o seu período de definição. O sinal da mosca que está sob experiência neste trabalho não é considerado

estacionário.

1.7.3 - Energia de um Sinal

Um outro conceito importante mencionado no presente trabalho é o de Energia do Sinal [PRI1], que se define como a seguir:

$$E = \sum_k x_k^2 \quad , \quad (1-12)$$

onde x_k representa cada amostra do sinal discreto em questão.

1.7.4 - O Teorema da Amostragem

Obviamente, realizar processamento digital de sinais requer que seja feito uma amostragem do sinal analógico que se quer processar, para que possamos trabalhar com um número finito de pontos ou amostras. Por isso, um conceito fundamental aqui é o Teorema da Amostragem, ou Teorema de Nyquist [SIN1]:

"Um sinal contínuo deve ser amostrado pelo menos o dobro de vezes da maior frequência contida nele, para que as amostras discretas possam representá-lo sem *alias*".

Em outras palavras, e já tomando por base a situação envolvida com o presente trabalho, isso significa que, por exemplo, um sinal, amostrado a uma taxa de 44100 amostras/segundo, poderá representar frequências de, no máximo, 22050 Hz sem *alias*.

Este conceito de *alias*, diretamente ligado ao Teorema da Amostragem, corresponde ao aparecimento de frequências fantasmas, devido à amostragem realizada num determinado evento ou o sinal estar numa taxa insuficiente.

1.7.5 - O Princípio da Incerteza

Um resultado fundamental da área de sinais é conhecido como *Princípio da Incerteza* [WAV1] e diz que não é possível saber exatamente qual é a frequência que existe num determinado instante de tempo, mas sim, num intervalo de tempo. Esse fato se conecta com a transformada *wavelet*, e, conseqüentemente, com a SPIKELET, que descreve faixas de frequências em intervalos de tempo e não em instantes de tempo.

1.7.6 - Sub-Amostragem (*downsampling*)

A sub amostragem [SIN1] de um sinal s_1 por n , $n > 0$, implica num outro sinal s_2 que possui n vezes menos amostras que s_1 , sendo as amostras de s_2 selecionadas da seguinte maneira: conserva-se uma amostra de s_1 excluindo as n amostras seguintes e, assim por diante, até o fim do sinal. O processo de sub amostragem em um sinal está diretamente relacionado com o Teorema da Amostragem e com *wavelets*.

O símbolo:

denota a sub-amostragem de um sinal por n .

1.7.7 - Causalidade e Estabilidade

Um sistema LTI é classificado como **causal** quando a saída depende somente dos valores passados ou presentes da entrada, ou seja, não pode gerar uma saída antes que a entrada seja aplicada. Já um sistema é classificado como **estável** quando a saída for limitada para uma entrada limitada. Estes conceitos serão atribuídos ao filtro criado no capítulo II.

1.8 - A Transformada de Fourier de Tempo Reduzido (STFT)

Uma modificação na FFT de um sinal corresponde a STFT [WAV1], que supõe que um dado sinal não estacionário, quando repartido em partes pequenas, pode ter cada uma dessas partes consideradas isoladamente como estacionárias. A equação (1-10) ilustra o fato para o caso contínuo.

$$STFT(x(t)) = \int_t (x(t)W(t-t'))e^{-j\omega t} dt \quad (1-13)$$

onde $x(t)$ é o sinal no domínio do tempo, W é a função que "janela" o sinal $x(t)$, $t-t'$ é a largura da janela, ω é a frequência, t é o tempo e $j = \sqrt{-1}$.

Existe, entretanto, um problema com o uso da STFT: a largura da função que "janela" o sinal. Janelas estreitas resultam em boa resolução no tempo, mas uma resolução mais pobre em frequência, enquanto que janelas largas resultam numa resolução melhor em frequência e pior no tempo, além de violarem a suposição de estacionariedade do trecho do sinal envolvido.

Uma possibilidade para contornar este fato consiste no uso da transformada *wavelet*, que irá trazer diversos níveis de resolução de tempo em diversas faixas de frequência com diferentes resoluções. A STFT não será utilizada no presente trabalho e foi mencionada apenas para fazer a ligação entre a FFT e a DWT.

1.9 - A Transformada *Wavelet* Discreta (DWT) e a Análise MRA

A transformada *wavelet* discreta [PRI1] [WAV1] [RIP1] consiste numa alternativa mais eficiente do que a STFT para realizar a análise tempo-frequência de um sinal. A DWT, que é como se classifica a SPIKELET-4, é na verdade de um par de filtros, sendo um deles passa-baixa (h) e outro passa-alta (g), em geral, com frequência de corte ($-3dB$) em $\pi/2$ ($\pi = 0.5$ é a

máxima frequência angular) e do tipo QMF sendo que, dado um sinal discreto, ele é submetido a ambos os filtros via convolução. Cada vez que esse processo é aplicado, diz-se que se tem um nível de decomposição e obtém-se dois novos sinais, sendo que um deles contém as frequências abaixo da metade da máxima frequência original do sinal e, o outro, contém as frequências acima deste limiar.

Após aplicar um nível de decomposição no sinal, apenas o novo sinal obtido pela aplicação do filtro passa-baixa é usado para continuar o processo. Existem casos em que ambos os 2 sinais são submetidos novamente ao processo de filtragem. Para tal particularidade se dá o nome de transformada *wavelet packet* [PAC1], que não será estudada aqui, por não ser utilizada no trabalho.

Um detalhe fundamental a ser notado é que cada vez que um nível da transformação é realizado, os dois novos sinais obtidos são sub amostrados por 2, pois eles contém apenas metade da faixa de frequências do sinal original (Teorema da Amostragem).

Para um sinal original de n amostras, a sua transformada *wavelet* associada tem a mesma quantidade de amostras e é composta por uma seqüência de coeficientes, iniciando-se com os coeficientes provenientes da aplicação do filtro passa-baixa no último nível, seguidos pelos coeficientes resultantes da aplicação dos filtros passa-alta nos níveis intermediários e terminando com os coeficientes resultantes da aplicação do filtro passa-alta do primeiro nível de decomposição. Todo este processo se encontra explicado na figura FIG1-2.

Para realizar a decomposição até o último nível possível, é necessário que o sinal discreto tenha comprimento equivalente a uma potência de 2, sendo possível realizar $\log(n)/\log(2)$ decomposições para um sinal de comprimento n . Um fator muito importante para que um filtro digital seja considerado uma *wavelet* é que a resposta em frequência do filtro passa-

baixa associado seja 1 em $\pi = 0$ e, obviamente, no caso de filtros QMF, 0 em $\pi = 0$ para o filtro passa-alta.

A sub amostragem por dois, realizada nos sinais da transformada *wavelet* em cada nível, pode ser representada por uma convolução modificada da seguinte forma:

$$y[n] = x[n] * t[n] = \sum_{k=0}^{n-1} t[k] x[2n-k] \quad (1-14)$$

Sendo mais especificamente:

$$y_{passabaixa}[n] = x[n] * h[n] = \sum_{k=0}^{n-1} h[k] x[2n-k] \quad (1-15)$$

$$y_{passaalta}[n] = x[n] * g[n] = \sum_{k=0}^{n-1} g[k] x[2n-k] \quad (1-16)$$

FIG1-2: Decomposição de um sinal discreto em vários níveis de resolução com uma transformada *wavelet* = (A,B,C,D,...,X).

A DWT está diretamente relacionada com a Análise Multi Resolutiva (MRA) proposta por Mallat, Meyer, Stromberg e outros [WAV2], que consiste

em decompor um vetor (sinal sob análise) $\vec{f}$ em uma soma de outros vetores pertencentes a uma seqüência de subespaços vetoriais [ALG2]. Em outras palavras, isso significa representar um sinal em vários níveis de resolução.

Então, de acordo com a MRA, para um vetor $\vec{f}$ de n pontos temos:

$$\vec{f} = \vec{A} + \vec{D} \quad \text{onde} \quad \vec{A} = \sum_{k=0}^{\frac{n}{2}-1} \langle \vec{f}, \vec{v}_k \rangle \vec{v}_k \quad \text{e} \quad \vec{D} = \sum_{k=0}^{\frac{n}{2}-1} \langle \vec{f}, \vec{w}_k \rangle \vec{w}_k \quad (1-17)$$

ou seja,

$\vec{A}$ é a projeção de $\vec{f}$ num subespaço V que possui uma base $\vec{v}$ com $n/2$ vetores e $\vec{D}$ é a projeção de $\vec{f}$ num subespaço W que possui uma base $\vec{w}$ com $n/2$ vetores, sendo:

- $V \perp W$ e, portanto, $\vec{A} \perp \vec{D}$;
- $\vec{v}_j \perp \vec{w}_j$ e, portanto, $\langle \vec{v}_j, \vec{w}_j \rangle = 0$;
- as bases $\vec{v}$ e $\vec{w}$ possuem $n/2$ vetores cada.

O processo acima consiste na decomposição em nível 1. Numa transformada *wavelet* de nível 2, o vetor $\vec{A}$ é novamente decomposto na soma de dois outros vetores ortogonais. Este processo pode ser repetido, como já foi mencionado, $\log(n)/\log(2)$ vezes.

Dessa forma, generalizando, para uma decomposição de nível j , temos:

$$\vec{f} = \vec{A}_j + \vec{D}_j + \vec{D}_{j-1} + \dots + \vec{D}_2 + \vec{D}_1 = \vec{A}_j + \sum_{t=1}^j \vec{D}_t \quad (1-18)$$

onde,

$\vec{A}_j$ é a projeção de $\vec{f}$ num subespaço V_j , que possui uma base $\vec{v}$ com $\frac{n}{2^j}$ vetores.

$\vec{D}_j$ é a projeção de $\vec{f}$ num subespaço W_j , que possui uma base $\vec{w}$ com $\frac{n}{2^j}$ vetores.

Em termos das funções $\Phi(n)$ e $\Psi(n)$ ao invés de $\vec{A}$ e $\vec{D}$ tem-se:

$$f(n) = \sum_{k=0}^{n/2^j-1} H_{j,k}(n) \Phi_{j,k}(n) + \sum_{t=1}^j \sum_{k=0}^{n/2^j-1} G_{t,k}(n) \Psi_{t,k}(n) \quad (1-19)$$

onde

$$\Phi(n) = \sum_k h_k \Phi(2n - k) \quad (1-20)$$

é conhecida como função *scaling* [WAV6], e

$$\Psi(n) = \sum_k g_k \Phi(2n - k) \quad (1-21)$$

é conhecida como função *wavelet*, ambas definidas recursivamente. Assim:

- os coeficientes h_k , que são o suporte (elementos diferentes de 0) dos vetores da base $\vec{v}_i$, formam um filtro digital passa-baixa.
- os coeficientes g_k , que são os vetores da base $\vec{w}_i$, formam um filtro digital passa-alta.
- K é o k -ésimo componente dos filtros e j é o j -ésimo nível de decomposição.
- $H_{j,k}(n) = \langle f, \Phi_{j,k}(n) \rangle$
- $G_{t,k}(n) = \langle f, \Psi_{t,k}(n) \rangle$
- os subespaços V_j são tais que $\dots V_0 \subset V_1 \subset V_2 \dots$
- V_{j+1} é soma direta de V_j com W_j , onde W_j é o complemento ortogonal de V_j em V_{j+1} .

Uma propriedade de interesse na DWT é a quantidade de momentos nulos, que pode ser calculada através da equação (1-22).

$$\sum_{k=0}^{n-1} t_k^m \Psi(t_k) \quad (1-22)$$

sendo n a quantidade de pontos da função *wavelet*, m o momento desejado e t cada ponto onde a função pode possuir valor diferente de 0 ($t = 0.5s$, onde s é um escalar inteiro maior ou igual a 0).

A equação (1-23), que faz parte do sistema utilizado para construção da Transformada de Daubechies, faz com que os coeficientes h_i obtidos correspondam a uma *wavelet* com $n/2$ momentos nulos, sendo n o tamanho do suporte.

$$\sum_{k=0}^{n-1} (-1)^k h_k k^b = 0, \quad b = 0, 1, \dots, n/2 - 1. \quad (1-23)$$

A quantidade de momentos nulos é uma propriedade interessante pois, para um sinal que pode ser (aproximadamente) descrito por um polinômio de grau menor que M e uma *wavelet* que possui M momentos nulos, os coeficientes de detalhamento serão (aproximadamente) zero. Embora este fato seja primordialmente importante nos esquemas de compressão de dados, ele será analisado na construção da SPIKELET-4.

Em particular, os termos chamados coeficientes de detalhamento designam o sinal obtido quando da passagem do sinal original pelo filtro passa-alta e os termos chamados coeficientes de aproximação designam o sinal obtido quando da passagem do sinal original pelo filtro passa-baixa.

A figura FIG1-3 apresenta o que é conhecido como *dyadic grid*, que é uma representação que relaciona o refinamento tempo-freqüência da transformada *wavelet*.

Dentre as *wavelets* mais conhecidas estão as transformadas de Haar, Daubechies, Coiflets, Symmlets, entre outras. No presente trabalho, vamos

nos fixar com mais detalhes na transformada de Daubechies, já que a SPIKELET-4 consiste numa modificação aplicada sobre ela.

FIG1-3: Dyadic Grid

1.10 - Resumo das Principais Técnicas para Detecção de *Spikes*

Para os sinais discretizados e armazenados de maneira digital, que possivelmente contenham *spikes* e *overlaps* e que foram colhidos de um único eletrodo extra ou intracelular, existem referências na literatura sobre alguns esquemas de detecção. A detecção de sinais gravados por mais de um eletrodo simultaneamente, como abordam [TES1] e [CRO1], foge ao escopo do presente trabalho. A seguir, estão relacionadas algumas destas técnicas.

- Detecção por *threshold*: corresponde ao método mais básico para realizar a classificação dos *spikes*, sendo que uma amostra do sinal é classificado como *spike* caso ele seja superior, em valor absoluto, a um determinado valor. Algumas pequenas variações existem [MIC1], mas, em geral, com resultados similares.
 - Vantagens: possui uma simplicidade muito grande para implementação e é totalmente determinístico.
 - Desvantagens: não é capaz de reconhecer um determinado formato e nem de detectar *overlaps*.

- Detecção por threshold com aferição intervalar entre spikes: consiste de uma pequena melhoria no método anterior, que usa como base um *threshold* e um histograma para verificar a isolação entre *spikes*.

- Vantagens: possui uma simplicidade relativa para implementação.

- Desvantagens: não é capaz de reconhecer um determinado formato. Baseia-se numa abordagem não totalmente determinística e são necessários muitos *spikes* para uma mensuração mais precisa.

- Detecção por correspondência de formatos: Consiste na tentativa de reconhecimento de um determinado formato de *spike*, onde as principais características avaliadas são a largura e a altura do *spike*, bem como a distância entre dois *spikes*.

- Vantagens: ainda possui simplicidade para implementação e é determinístico.

- Desvantagens: a altura e largura do *spike* não são suficientes para caracterizar o seu formato, sendo que *overlaps* não podem ser detectados com essa técnica.

- Detecção baseada na análise do componente principal: propõe descobrir as direções que representem as maiores variações no sinal.

- Vantagens: segundo [REV1] e [JAV1], é uma das técnicas de melhor desempenho com um custo computacional baixo.

- Desvantagens: ainda não é um método propício para a detecção e posicionamento de *overlaps*.

- Detecção baseada em filtros: Diversos filtros digitais são construídos para dar máxima resposta a um determinado formato de *spike* e mínima resposta aos outros formatos, sendo que o sinal sob análise é convoluído com os coeficientes dos filtros e os sinais resultantes são comparados em amplitude.

- Vantagens: o método tem um custo computacional não

muito alto e faz uma análise das frequências envolvidas no sinal.

- Desvantagens: segundo [REV1], esses métodos baseados em filtros não classificam *spikes* e *overlaps* de uma maneira tão precisa quanto à análise do componente principal ou mesmo à correspondência de formato. Ainda segundo [REV1], métodos baseados em filtros não apresentam bons resultados quando comparados com outros métodos e, além disso, a identificação de *overlaps* com formas não estacionárias, via filtragem, é um problema de difícil solução. Outros autores, tais como [SIG1] [TIM1], apresentam métodos baseados na transformada *wavelet*, com resultados interessantes, principalmente para detecção e classificação de sinais médicos de ECG e EEG [FIL3].

- Detecção baseada no algoritmo de Lewicki: realiza uma decomposição do sinal em diversos outros sinais com formatos de *spikes* / *overlaps*. Após encontrar um pico num desses sinais e selecionar a região em torno desse pico, as amostras do provável *spike* / *overlap* são classificadas com o uso de uma estrutura de busca que retorna uma lista dos mais prováveis *spikes* que compõe o *spike* principal e sua probabilidade.

- Vantagens: possui custo computacional de ordem logarítmica, permite a identificação de *overlaps* e uma simplicidade relativa para implementação.

- Desvantagens: usa uma abordagem não determinística e não possui 100 % de acertos.

- Detecção Bayseana: tenta descobrir, através de cálculo probabilístico condicional, a probabilidade de um certo conjunto de pontos, relativos ao possível *spike*, pertencer a uma certa classe pré-estabelecida de pontos que melhor descreve um *spike* de determinado formato.

- Vantagens: realiza uma quantificação da certeza com a qual os *spikes* são classificados e possui um custo computacional não muito elevado.
- Desvantagens: usa uma abordagem não determinística. Uma variante, encontrada em [SPK1], possui uma abordagem interessante para detecção de *overlaps*.

- Detecção baseada em redes neurais: faz a detecção com o uso de algoritmos “inteligentes” e baseados em aprendizagem, além de cálculos estatísticos .
 - Vantagens: possui, em geral, um grau maior de acerto nas classificações do que diversos outros métodos e algumas de suas variantes podem detectar *overlaps*.
 - Desvantagens: em geral, usa uma abordagem não determinística, além de possuir um custo computacional mais elevado.

- Detecção baseada em correlação: existem diversas variações desta proposta, mas todas elas se centram em correlacionar o sinal original com um *spike* pré-definido, obtendo um coeficiente que expresse o grau de semelhança entre os dois sinais.
 - Vantagem: possui custo computacional baixo.
 - Desvantagens: não se presta bem a detecção de formatos nos *overlaps*.

Muito embora a descrição apresentada acima tenha um caráter resumido, pode-se dizer de maneira geral que não existe um método perfeito, mas sim que, cada um tem suas características, vantagens e desvantagens.

O próximo capítulo descreve, com detalhes, a construção da SPIKELET-4.

Capítulo II

CONSTRUÇÃO DA SPIKELET

Este capítulo descreve, de maneira completa, como foi criada a SPIKELET-4, aproveitando para revisar o modo como foram criadas as wavelets de Daubechies.

Para criar um novo filtro *wavelet* que respondesse bem aos *spikes* e *overlaps* de interesse, várias técnicas foram pesquisadas: [ALG1], [FIL2], [INT1], [WAV2], [WAV3], [WAV7], entre outras. Em especial, o artigo [WAV4] foi muito útil, pois possui explicações detalhadas da construção das *wavelets* de Daubechies. Além disso, o artigo [EST1] propõe um método interessante para criação de *wavelets* para se ajustar a um determinado sinal.

Analisando mais profundamente este último artigo, foi possível perceber que ele não adotava critérios rigorosos quando propunha fazer uma maximização da energia do sinal resultante da convolução do sinal original sob análise com os coeficientes do filtro procurado (n incógnitas). Em particular, a proposta para resolver o sistema $n \times n$ linear homogêneo resultante era dispensar uma das equações (sem critério para a escolha) e então resolver o sistema restante com um grau de liberdade. Além disso, o filtro resultante possuía frequência de corte arbitrária e dependente da equação dispensada, sendo difícil mensurar o grau de dependência, devido ao fato de que coeficientes completamente arbitrários podem aparecer no sistema.

Esse último artigo propiciou uma idéia que, utilizada em conjunto com a forma como Daubechies construiu seus filtros, levou à construção da SPIKELET-4. A seguir, encontram-se descritos todos os passos da criação.

2.1 – Obtenção do sinal padrão de interesse

A partir do arquivo digital, amostrado a 44100 amostras/segundo, que contém diversos *spikes* e *overlaps*, cujo um pequeno trecho está exibido a seguir na figura FIG2-1, foram selecionados, por inspeção visual, 78 *spikes*, com 32 amostras cada, também plotados a seguir na figura FIG2-3. A partir deles, e com o suporte de um programa de computador em linguagem C++, foi determinado um *spike médio padrão*, também com 32 amostras, como sendo o sinal de interesse.

O *spike médio padrão*, exibido na figura FIG2-4, corresponde ao sinal definido experimentalmente com base em 78 *spikes*, sendo que, dentre as suas 32 amostras ($32 \text{ amostras} \approx 725.62 \mu s$), a décima primeira amostra correspondente à amplitude de maior declividade. Este ponto (amostra, amplitude de maior declividade) fica doravante definido como o “instante” onde um *spike* ou *overlap* ocorre dentro do sinal. A utilização de 32 amostras para compor o *spike médio padrão* deve-se a dois fatores:

- este número de pontos descreve perfeitamente o formato dos *spikes* e *overlaps* de interesse;
- 32 é potência de 2, que é propício para realizar uma transformada *wavelet* do sinal.

O *spike médio padrão* possui os valores a seguir (amostra, amplitude):

0, 4533	4, 7858	8, -6648	12, -29400	16, -20572	20, -2644	24, 7318	28, 8421
1, 5562	5, 7278	9, -14330	13, -29865	17, -15930	21, 807	25, 8302	29, 7822
2, 6517	6, 4903	10, -21331	14, -28071	18, -11207	22, 3627	26, 8771	30, 7044
3, 7358	7, 187	11, -26532	15, -24728	19, -6682	23, 5771	27, 8772	31, 6222

Tabela CP2-T1: Dados do spike médio padrão

A figura FIG2-2 mostra a FFT do sinal da figura FIG2-1, confirmando a forte presença de frequências na faixa abaixo de 5512.50 Hz. Tal fato será utilizado durante o algoritmo de detecção, detalhado no próximo capítulo. A figura FIG2-5 mostra um *spike* e 5 formatos de *overlaps* comumente encontrados nos sinais de interesse neste trabalho e que serão facilmente classificados com o uso da SPIKELET-4 combinada com o algoritmo proposto, detalhado no próximo capítulo.

Figura FIG2-3: os 78 spikes utilizados para a obtenção do spike médio padrão

Figura FIG2-1: sinal contendo uma série de spikes e overlaps

Figura FIG2-4: spike médio padrão

Figura FIG2-2: FFT do sinal da figura FIG2-1

Spike e 5 tipos de Overlaps

Figura FIG2-5: Spike e overlaps típicos

2.2 - Técnica Proposta

Depois de definido o sinal de interesse (*spike* médio padrão), três conjuntos de etapas foram seguidas: as etapas do grupo A (A1, A2 e A3), as etapas do grupo B (B1, B2 e B3) que ocorrem em paralelo com as etapas do grupo A e, finalmente, as etapas do grupo C (C1, C2 e C3) que são posteriores às etapas dos grupos A e B. A figura FIG2-6 exhibe o processo completo, descrito detalhadamente abaixo em cada uma das suas etapas:

- Etapas A1: Os três grupos de equações, quando juntos, correspondem à definição da Transformada de Daubechies. O primeiro é o responsável por fazer com que a função *scaling* possa representar um polinômio de grau no máximo $n/2$, sendo n o tamanho do suporte do filtro. Isso garante que, em termos de resposta em frequência, tenhamos um filtro passa-baixa com frequência de corte na metade da máxima frequência angular, ou

Construção da SPIKELET-4

Figura FIG2-6: esquema de construção da SPIKELET-4

seja, em $\pi/2$. Isso implica que a *wavelet* possui $n/2$ momentos nulos. O segundo grupo, constituído por uma única equação, cuida da normalização dos coeficientes e vem do fato de que, para a definição das funções *scaling* de um dado formato, a área correspondente debaixo de cada curva deve ser unitária [WAV4]:

$\int \Phi(x) dx = 1$. O terceiro grupo cuida da ortogonalidade entre cada par de vetores da função *scaling*.

- Etapa A2: Nesta etapa, retira-se a condição de ortogonalidade da Transformada de Daubechies, já que esta condição não terá uso na Transformada SPIKELET-4. Não é necessário, para a aplicação em questão, reconstruir o sinal original a partir de sua transformada *wavelet*.
- Etapa A3: As equações da etapa A2 são especificadas para um suporte $n = 4$. O sistema resultante desta etapa é linear e possui $n/2 + 1 = 3$ equações com n incógnitas.
- Etapa B1: Realiza-se a convolução, seguida de sub-amostragem por 2, do *spike* médio padrão (vetor $\vec{U}$ da figura FIG2-6) com um filtro de suporte 4 com coeficientes genéricos (incógnitas) h_0, h_1, h_2, h_3 . O sinal resultante é a projeção de $\vec{U}$ no subespaço W .
- Etapa B2: a energia do sinal resultante da etapa B1 anterior (projeção de $\vec{U}$ em W) é maximizada, objetivando transferir o máximo de energia do sinal original $\vec{U}$ para o sinal resultante. Para isso, são calculadas as derivadas parciais do sinal-energia resultante em relação a cada um dos coeficientes h_0, h_1, h_2, h_3 .

- Etapa B3: cada uma das 4 derivadas parciais obtidas na etapa B2 é igualada a 0, formando um sistema linear homogêneo de 4 equações com 4 incógnitas.
- Etapa C1: reúnem-se as equações obtidas nas etapas A3 e B3. O novo sistema é linear e possui $(n/2 + 1) + n = 7$ equações e $n = 4$ incógnitas.
- Etapa C2: devido à inexistência de solução exata para o sistema da etapa C1, optou-se pela solução através do Método dos Mínimos Quadrados [CAL1]. O sistema é então expresso na forma matricial e cada lado da igualdade é multiplicado pela matriz transposta dos pesos das incógnitas.
- Etapa C3: a solução obtida implica nos coeficientes h_0, h_1, h_2, h_3 encontrados, que representam o filtro passa-baixa. Como o sistema definido é do tipo QMF, os coeficientes do filtro passa-alta respectivo obedecem à equação (1-1), ficando portanto a SPIKELET-4 definida de acordo com a Tabela CP3-T1:

Filtro Passa-Baixa SPIKELET-4	Filtro Passa-Alta SPIKELET-4
$h_0 = 0.26964482896235847376$	$g_0 = 0.23524044702452745481$
$h_1 = 0.76237548312490721614$	$g_1 = -0.73270322306815560687$
$h_2 = 0.73270322306815560687$	$g_2 = 0.76237548312490721614$
$h_3 = 0.23524044702452745481$	$g_3 = -0.26964482896235847376$
<i>Tabela CP3-T1: coeficientes do Filtro SPIKELET-4</i>	

Agora, de posse dos coeficientes dos filtros, podemos construir as funções *scaling* e *wavelet*.

2.3 - Função *Scaling*

De acordo com a teoria das *wavelets*, a função *scaling* é definida recursivamente pela equação (1-20). Se temos $n = 4$ coeficientes, os valores de $\Phi(x)$ para $x < 0$ são nulos. Da mesma forma, para $x > n-1$, também a função é nula. Sendo assim, existirão valores, possivelmente diferentes de 0, para $\Phi(0)$, $\Phi(1)$, $\Phi(2)$ e $\Phi(3)$ e também nos pontos intermediários $\Phi(1/2)$, $\Phi(3/2)$ e $\Phi(5/2)$.

De acordo com a definição recursiva dada acima, temos então:

$$\Phi(0) = h_0 \Phi(0) ,$$

$$\Phi(1) = h_0 \Phi(2) + h_1 \Phi(1) + h_2 \Phi(0) ,$$

$$\Phi(2) = h_1 \Phi(3) + h_2 \Phi(2) + h_3 \Phi(1) ,$$

$$\Phi(3) = h_3 \Phi(3) .$$

Na forma matricial, $MT = T$, onde

$$M = \begin{bmatrix} h_0 & 0 & 0 & 0 \\ h_2 & h_1 & h_0 & 0 \\ 0 & h_3 & h_2 & h_1 \\ 0 & 0 & 0 & h_3 \end{bmatrix} ,$$

e

$$T = \begin{bmatrix} \Phi(0) \\ \Phi(1) \\ \Phi(2) \\ \Phi(3) \end{bmatrix} .$$

Portanto, a matriz T com os valores da função *scaling* procurada é o autovetor de M correspondente ao autovalor 1. Como a solução do problema $(M - I)T = 0$ não é única, uma condição de normalização é necessária para determinar um único autovetor. Esta condição é

$$\sum_k \Phi(k) = 1$$

Sendo assim, temos:

$$\begin{aligned}(h_0 - 1)\Phi(0) &= 0 \quad , \\ h_2 \Phi(0) + (h_1 - 1)\Phi(1) + h_0 \Phi(2) &= 0 \quad , \\ h_3 \Phi(1) + (h_2 - 1)\Phi(2) + h_1 \Phi(3) &= 0 \quad , \\ (h_3 - 1)\Phi(3) &= 0 \quad , \\ \Phi(0) + \Phi(1) + \Phi(2) + \Phi(3) &= 1 \quad .\end{aligned}$$

Como já encontramos os valores de h_0, h_1, h_2 e h_3 , e eles são todos diferentes de zero, o sistema acima implica em:

$$\begin{aligned}\Phi(0) &= 0 \quad , \\ \Phi(3) &= 0 \quad , \\ (h_1 - 1)\Phi(1) + h_0 \Phi(2) &= 0 \quad , \\ h_3 \Phi(1) + (h_2 - 1)\Phi(2) &= 0 \quad , \\ \Phi(0) + \Phi(1) + \Phi(2) + \Phi(3) &= 1 \quad .\end{aligned}$$

e restam as equações:

$$\begin{aligned}(h_1 - 1)\Phi(1) + h_0 \Phi(2) &= 0 \quad , \\ h_3 \Phi(1) + (h_2 - 1)\Phi(2) &= 0 \quad , \\ \Phi(1) + \Phi(2) &= 1 \quad .\end{aligned}$$

que são:

$$\begin{aligned}-0.23762451687509278386 \Phi(1) + 0.26964482896235847376 \Phi(2) &= 0 \quad , \\ 0.23524044702452745481 \Phi(1) - 0.26729677693184439313 \Phi(2) &= 0 \quad , \\ \Phi(1) + \Phi(2) &= 1 \quad .\end{aligned}$$

Devido ao fato de que as equações da etapa A3 da figura FIG2-6

foram perturbadas pelo ajuste via Mínimos Quadrados realizado para encontrar os coeficientes $h_0, \dots, h_3$, as duas primeiras equações que formam o sistema logo acima não são exatamente múltiplas uma da outra, como ocorre na criação da função *scaling* do filtro de Daubechies. Caso tal propriedade estivesse presente, o sistema em questão possuiria solução exata e diferente de 0. Optou-se aqui pela solução utilizando novamente o Método dos Mínimos Quadrados, obtendo-se:

$$\Phi(1) = 0.53172639832546342298 \quad ,$$

$$\Phi(2) = 0.46827358753833858707 \quad .$$

e, para os pontos intermediários,

$$\Phi(x/2) = \sum_k h_k \Phi(x - k) \quad .$$

Portanto:

$$\Phi(1/2) = h_0 \Phi(1) = 0.14337727373124048436 \quad ,$$

$$\Phi(3/2) = h_1 \Phi(2) + h_2 \Phi(1) = 0.74659794837766346731 \quad ,$$

$$\Phi(5/2) = h_3 \Phi(2) = 0.11015688806229795293 \quad .$$

2.4 - Função *Wavelet*

Utilizando os resultados obtidos no cálculo da função *scaling* e a definição recursiva da função *wavelet* (1-21), obtemos:

$$\Psi(0) = g_0 \Phi(0) \quad ,$$

$$\Psi(1) = g_0 \Phi(2) + g_1 \Phi(1) + g_2 \Phi(0) \quad ,$$

$$\Psi(2) = g_1 \Phi(3) + g_2 \Phi(2) + g_3 \Phi(1) \quad ,$$

$$\Psi(3) = g_3 \Phi(3) \quad .$$

o que leva a

$$\Psi(0) = 0 \quad ,$$

$$\Psi(3) = 0 \quad ,$$

$$\Psi(1) = g_0 \Phi(2) + g_1 \Phi(1) = -0.27944075778119104037 \quad ,$$

$$\Psi(2) = g_2 \Phi(2) + g_3 \Phi(1) = 0.21362302880293393414 \quad .$$

Para os pontos intermediários,

$$\Psi(x/2) = \sum_k g_k \Phi(x-k)$$

e, portanto,

$$\Psi(1/2) = g_0 \Phi(1) = 0.12508355563682396761 \quad ,$$

$$\Psi(3/2) = g_1 \Phi(2) + g_2 \Phi(1) = 0.06226960294661321171 \quad ,$$

$$\Psi(5/2) = g_3 \Phi(2) = -0.12626755141936529814 \quad .$$

2.5 - Análise dos Momentos Nulos da SPIKELET-4

A Transformada de Daubechies de suporte 4 possui 2 momentos nulos devidos à equação (1-23). Embora a Transformada SPIKELET-4 conserve esta equação, o ajuste por Mínimos Quadrados utilizado para encontrar os coeficientes $h_0, \dots, h_3$ causa perturbação. O cálculo dos momentos da SPIKELET-4 através da equação (1-22), revela, entretanto, que esses dois momentos não estão demasiadamente distantes de 0:

$$\text{- momento de ordem 0} = -0.0047321 \quad ,$$

$$\text{- momento de ordem 1} = -0.0119173 \quad .$$

2.6 - Estabilidade e Causalidade da SPIKELET-4

Tomando os coeficientes $h_0, \dots, h_3$, é possível escrever a equação de diferenças:

$$\begin{aligned} y_n &= h_0 x_{2n} + h_1 x_{2n-1} + h_2 x_{2n-2} + h_3 x_{2n-3} \\ &= h_0 x_{n+n} + h_1 x_{n+n-1} + h_2 x_{n+n-2} + h_3 x_{n+n-3} \end{aligned}$$

Tomando a Transformada Z de ambos os lados, obtemos a função de transferência $Y(z)/X(z)$:

$$\begin{aligned} Y(z) &= h_0 X(z) z^n + h_1 X(z) z^{n-1} + h_2 X(z) z^{n-2} + h_3 X(z) z^{n-3} \\ &= X(z) (h_0 z^n + h_1 z^{n-1} + h_2 z^{n-2} + h_3 z^{n-3}) \\ \frac{Y(z)}{X(z)} &= h_0 z^n + h_1 z^{n-1} + h_2 z^{n-2} + h_3 z^{n-3} \\ &= z^n (h_0 + h_1 z^{-1} + h_2 z^{-2} + h_3 z^{-3}) \\ &= \frac{h_0 + h_1 z^{-1} + h_2 z^{-2} + h_3 z^{-3}}{z^{-n}} \end{aligned}$$

Substituindo os valores de $h_0, \dots, h_3$ tem-se,

$$\frac{Y(z)}{X(z)} = \frac{0.269644828962358 + 0.762375483124907 z^{-1} + 0.732703223068155 z^{-2} + 0.235240447024527 z^{-3}}{z^{-n}}$$

A figura FIG2.7 exibe o Diagrama de Bode correspondente. O fato de todos os pólos estarem dentro do círculo unitário implica que o sistema é estável e causal. Como existem dois zeros fora do círculo unitário, o sistema inverso não é estável e causal. Entretanto, como ele não é necessário para a aplicação do algoritmo proposto neste trabalho, e nem mesmo foi calculado aqui, este fato não implica em nenhum problema.

Figura FIG2-7: Diagrama de Bode do filtro SPIKELET-4: os círculos em azul correspondem aos zeros e a cruz em vermelho corresponde aos pólos da função de transferência.

2.7 - Considerações

Para todos os testes foi utilizada *double-precision* para o cálculo dos coeficientes do filtro SPIKELET-4.

A seguir, nas figuras FIG2-8 e FIG2-9, encontram-se os gráficos das funções *scaling* e *wavelet* para os filtros SPIKELET-4 e, nas figuras FIG2-10 e FIG2-11, encontram-se as mesmas funções para os filtros DAUBECHIES-4. Pode-se notar que essas funções referentes ao filtro SPIKELET-4 são relativamente diferentes das funções do filtro DAUBECHIES-4, devido à otimização realizada para o sinal do *spike* médio padrão e à perturbação causada pelo ajuste por Mínimos Quadrados.

As figuras FIG2-12 e FIG2-13 apresentam as respostas em frequência normalizadas dos filtros SPIKELET-4: $H_{passabaixa}(0) = 1$, $H_{passabaixa}(\pi) \approx 0$ e

$H_{passa\text{alta}}(0) \approx 0$, $H_{passa\text{alta}}(\pi) = 1$, enquanto que as figuras FIG2-14 e FIG2-15 apresentam as respostas em frequência normalizadas dos filtros DAUBECHIES-4. Já a figura FIG2-16 exibe a resposta em frequência dos filtros passa-baixa e passa-alta SPIKELET-4 em escala logarítmica (dB).

Finalmente, a figura FIG2-17 mostra que a parte do *spike* cuja transformada SPIKELET-4 possui maior amplitude é a borda de subida. Este resultado será fundamental para o desenvolvimento do algoritmo de classificação proposto no próximo capítulo.

Um fato interessante que ocorre é que, para suportes $n > 4$, a frequência de corte dos filtros SPIKELET passa-baixa e passa-alta se afasta muito de $\pi/2$, ao contrário dos filtros da família DAUBECHIES. Isso ocorre devido ao fato das $n/2$ equações que correspondem à interpolação polinomial mencionada anteriormente na etapa A1 da figura FIG2-6 serem muito perturbadas pelo ajuste por Mínimos Quadrados. Entretanto, como os filtros com suporte 4 já se mostraram suficientes para aplicação do algoritmo de detecção desenvolvido, como consta no próximo capítulo, este fato não representou problema.

Outro fato a ser mencionado é que os filtros aqui construídos não são classificados como ortogonais, pois essa característica foi retirada das equações de construção para dar lugar às equações de correspondência com o *spike* médio padrão. Portanto, de acordo com o estudo precedente e as figuras FIG2-12, 13 e 16, o filtro SPIKELET-4 possui as seguintes características :

- *wavelet* não simétrica;
- filtro tipo FIR (*Finite Impulse Response filter*);
- filtro estável e causal;
- filtro com suporte compacto (4) de 3º ordem;
- a energia do filtro, que é dada pela equação (1-12), é:

$$h_0^2 + h_1^2 + h_2^2 + h_3^2 = 1.24611679206683811927 \text{ , que é diferente de}$$

1 e, portanto, a transformada SPIKELET-4 não conserva a energia do sinal original a qual ela é aplicada;

- a resposta em fase é não linear;
- a atenuação da banda de rejeição é de cerca de 49.8 dB;
- a máxima variação de ganho na banda de passagem é de 3 dB;
- a banda de transição vai de 3 a 49.8 dB, ou seja a largura de banda é de 46.8 dB e a frequência de corte está em $\pi/5$. Embora a frequência de corte esteja relativamente afastada de $\pi/2$ e a largura de banda seja consideravelmente alta, o filtro apresentou desempenho ótimo para classificar os *spikes* e *overlaps* e ainda conseguiu separar relativamente bem as faixas de frequências de interesse, após uma decomposição *wavelet* nível 3, como será visto mais adiante;
- cada vetor V_j é ortogonal apenas ao vetor W_j , onde V é a base constituída pelos vetores de suporte 4 relacionada aos coeficientes do filtro passa-baixa (coeficientes h) e W é a base constituída dos vetores de suporte 4 relacionada aos coeficientes do filtro passa-alta (coeficientes g):

$$\begin{aligned}
 V_0 &= (h_0, h_1, h_2, h_3, 0, 0, 0, \dots, 0), & W_0 &= (g_0, g_1, g_2, g_3, 0, 0, 0, \dots, 0), \\
 V_1 &= (0, 0, h_0, h_1, h_2, h_3, 0, \dots, 0), & W_1 &= (0, 0, g_0, g_1, g_2, g_3, 0, \dots, 0), \\
 &\dots & &\dots \\
 V_n &= (h_2, h_3, 0, 0, \dots, 0, 0, h_0, h_1), & W_n &= (g_2, g_3, 0, 0, \dots, 0, 0, g_0, g_1).
 \end{aligned}$$

Em termos de *software*, foi implementado na forma de um programa de computador em linguagem C++ [EST2], em ambiente RED HAT LINUX 8.0 – *kernel* 2.4.18-14 [LIN1], num computador PC com Processador Intel Pentium IV – 1.7 Ghz e com 256Mb de memória RAM, um programa que parte dos dados do *spike* médio padrão e calcula os coeficientes do filtro SPIKELET-4. Ele consta do Apêndice I.

O próximo capítulo descreve o algoritmo proposto para classificação dos *spikes* e *overlaps*, que faz uso da SPIKELET-4.

Figura FIG2-10: Scaling – DAUBECHIES-4

Figura FIG2-8: Scaling – SPIKELET4

Figura FIG2-11: Wavelet – DAUBECHIES-4

Figura FIG2-9: Wavelet – SPIKELET4

Figura FIG2-14: Filtro Passa-Baixa-DAUBECHIES-4

Figura FIG2-12: Filtro Passa-Baixa-SPIKELET4

Figura FIG2-15: Filtro Passa-Alta- DAUBECHIES-4

Figura FIG2-13: Filtro Passa-Alta-SPIKELET4

Figura FIG2-16: Resposta em frequência dos filtros SPIKELET-4 em escala logarítmica. F_c = frequência de corte. F_s = frequência de rejeição.

Figura FIG2-17: Reação da SPIKELET-4 a um spike de 32 amostras

Capítulo III

ALGORITMO PROPOSTO PARA DETECÇÃO E CLASSIFICAÇÃO DOS *SPIKES* E *OVERLAPS*

Este Capítulo tratará de descrever o Algoritmo proposto para a classificação dos spikes e dos spikes sobrepostos (overlaps) fazendo uso da SPIKELET-4. Para compreender o algoritmo, é realizado um estudo, no domínio da frequência, do spike médio padrão e de um overlap.

De posse dos coeficientes SPIKELET-4 já construídos, basta agora formalizar o algoritmo que realiza as detecções correspondentes. A Transformada SPIKELET-4 respondeu muito bem a identificação dos *spikes* e *overlaps*, sendo possível realizar a identificação dos seguintes tipos de formatos, que se encontram exemplificados ao final do Capítulo:

- *spike*: definido como formato muito similar ao *spike* médio padrão apresentado anteriormente.
- *overlap acentuado à esquerda*: *spike* com uma sobreposição de outro *spike* bem à esquerda, próximo ao início.
- *overlap à esquerda*: *spike* com uma sobreposição de outro *spike* entre o início e o pico negativo.
- *overlap à direita*: *spike* com uma sobreposição de outro *spike* entre o pico negativo e o fim.
- *overlap acentuado à direita*: *spike* com uma sobreposição de outro *spike* bem à direita, próximo ao fim.

- *overlap à esquerda e à direita: spike* com sobreposição de outros dois *spikes*, um à esquerda e outro à direita.

- *spike irregular / deformado*: *spike* fora do formato, possivelmente por interferência de sinais de outros neurônios, ficando o *spike* resultante relativamente desfigurado.

3.1 - Descrição

O algoritmo de identificação, exibido na figura FIG3-1 a seguir, consiste basicamente de um série de testes e rotinas que perduram enquanto as amostras do sinal não terminam e que são realizados após o arquivo que contém os pontos (formato de audio wav [WAV5] - mono canal – sem compressão - 16 bits - 44100 Khz) ser desmontado:

Figura FIG3-1: Fluxograma do algoritmo proposto para identificação dos spikes e overlaps com o uso da transformada SPIKELET4

- Teste AMP: A passagem por tal teste indica que a amostra em questão possui um limiar (*threshold*), em modulo, maior ou igual a uma amplitude pré-estabelecida. Para os testes realizados, foi considerado um candidato válido, todo *spike* com amplitude igual ou superior a 20 % da máxima amplitude possível (32768 para um dado de 16 bits com sinal) que é, aproximadamente, 6600.

- Teste FORM: A passagem por este teste indica que a amostra sob análise é maior, em módulo, que as amostras imediatamente à esquerda e imediatamente à direita. Isso indica um pico no sinal. A implementação deste teste garante que, numa vizinhança de 13 amostras anteriores ao pico e 18 posteriores a ele, não existe amostra com maior amplitude (em módulo), pois isso é fundamental para a correta identificação dos formatos. Em suma, a amostra classificada como pico é, de fato, o pico do *spike*, ou, o pico principal (de maior amplitude) de um *overlap*.

- FORM SIGNAL: Apenas no caso de aprovação nos dois testes anteriores, forma-se um sinal composto por 32 amostras, sendo 13 anteriores ao pico e 18 posteriores à ele. Em seguida, realiza-se a transformada SPIKELET-4 nível 3 desse sinal.

- Teste SPK: Consiste em verificar se a Transformada SPIKELET-4 nível 3 do sinal, possui uma inversão entre as amostras 1 e 2 (segunda e terceira amostras). Conforme será explicado mais à frente, este teste confirma a presença de frequências no intervalo de 0 a 2756.25 Hz entre as amostras 16 e 23 do sinal original formado em *FORM SIGNAL*. Tal intervalo contém a borda de subida do *spike*. Caso o sinal não seja aprovado por esse teste, o *spike* é considerado irregular, pois contém a porção mais forte de frequências procuradas fora do intervalo esperado, o que indica uma deformação no *spike*. Já no caso de aprovação, alguns outros testes são realizados para verificar em qual das 6 possibilidades descritas acima o *spike* se enquadra melhor.

- Teste MRA-1: confirma a presença de 3 picos na faixa de frequências de 2756.25 até 5512.5 Hz, indicando um *spike* regular. Tais picos são confirmados pela inversão de sinal entre as amostras 4 e 5, 5 e 6 e, também, 6 e 7 da transformada SPIKELET-4 do sinal.

- Teste MRA-2: confirma a presença de 2 picos na faixa de frequências de 2756.25 até 5512.5 Hz, indicando um *overlap* à esquerda. Tais picos são confirmados pela inversão de sinal entre as amostras 4 e 5 e, também, 5 e 6 da transformada SPIKELET-4 do sinal.

- Teste MRA-3: confirma a presença de 1 pico na faixa de frequências de 2756.25 até 5512.5 Hz, indicando um *overlap* acentuado à esquerda. Tal pico é confirmado pela inversão de sinal entre as amostras 4 e 5 da transformada SPIKELET-4 do sinal.

- Teste MRA-4: confirma a presença de 2 picos na faixa de frequências de 2756.25 até 5512.5 Hz, indicando um *overlap* à direita. Tais picos são confirmados pela inversão de sinal entre as amostras 5 e 6 e também 6 e 7 da transformada SPIKELET-4 do sinal.

- Teste MRA-5: confirma a presença de 3 picos na faixa de frequências de 2756.25 até 5512.5 Hz, indicando um *overlap* acentuado à direita. Tal pico é confirmado pela inversão de sinal entre as amostras 6 e 7 da transformada SPIKELET-4 do sinal.

- Teste MRA-6: confirma a presença de 2 picos na faixa de frequências de 2756.25 até 5512.5 Hz, indicando um *overlap* à esquerda e à direita. Tais picos são confirmados pela inversão de sinal entre as amostras 4 e 5 e também 6 e 7 da transformada SPIKELET-4 do sinal.

O esquema de testes MRA está esboçado na figura FIG3-2 a seguir. Todos os testes MRA-1, ... , MRA-6 emitem como resposta o formato do sinal

detectado, bem como o “instante” em que o *spike* / *overlap* ocorre, que corresponde à amostra que tem a reta tangente à ela com maior declividade em relação ao eixo do tempo.

Deve ficar claro que a classificação dos formatos realizada, em especial o posicionamento dos overlaps, é feita com base no pico do sinal que foi encontrado em Teste FORM. Como o pico do spike médio padrão encontra-se na décima quarta amostra (amostra número 13), ficam então justificados os valores 13 e 18 mencionados em Teste FORM e em FORM SIGNAL.

Cada vez que um *spike* / *overlap* é encontrado, o algoritmo pode avançar 32 amostras para frente do pico correspondente a ele, pois esta é a menor distância pico-a-pico entre 2 *spikes* / *overlaps*. No caso desta distância ser menor, o *spike* seguinte é considerado um *overlap* do anterior.

Figura FIG3-2: Testes de inversão de sinal entre as amostras 1 e 2, 4 e 5, 5 e 6, 6 e 7 na transformada SPIKELET4 de um possível spike ou overlap

Cada um dos testes MRA acima leva em conta quantas bordas de subida existem em uma determinada faixa de frequências e também em qual (ou quais) intervalo(s) elas estão localizadas. O fato de existir uma

inversão de sinal entre amostras vizinhas de uma transformada *wavelet* (SPIKELET-4 no caso), significa que o gráfico de tal transformada tem uma passagem por zero (*zero-crossing*) entre estas duas amostras. Para uma *wavelet* não simétrica, como é o caso da SPIKELET-4, os *zero-crossings* da transformada indicam os picos dominantes do sinal original para os quais o filtro é mais sensível [HAN1 – pp. 232], que no caso da SPIKELET-4, é a borda de subida, como já foi observado no capítulo anterior.

A figura FIG3-3 a seguir mostra um *spike* comum sobreposto ao mesmo sinal composto apenas por frequências de 0 a 2756.25 Hz. Este valor é o valor expresso pelas 4 primeiras amostras de uma transformada *wavelet* (SPIKELET-4 no caso) de um sinal de 32 amostras amostrado a 44100 am/seg e decomposto em nível 3. É fácil notar que não existe grande diferença e, portanto, para qualquer *spike* sem *overlap*, pode-se representá-lo apenas com esta faixa de frequências. Isso já não acontece com um *overlap*, como exibido na figura FIG3-4, que precisa dos componentes de 2756.25 até 5512.50 Hz para não perder o formato.

As figuras FIG3-5 e FIG3-6 mostram a decomposição nas sub-bandas *wavelet* para um *spike* e um *overlap*, respectivamente, onde pode-se notar as bordas de subida, que são essenciais para o esquema de identificação. Já a figura FIG3-7 exhibe o esquema de decomposição *wavelet* nível 3 para a análise dos sinais em questão. Note que na decomposição nível 3, existem 4 amostras representando a faixa de frequência que vai de 0 a 2756.25 Hz, designadas por A e 4 amostras representando a faixa de frequência que vai de 2756.25 até 5512.50 Hz, designadas por B na figura. Como o sinal possui 32 pontos, tem-se que $32 / 4 = 8$ e, portanto, cada coeficiente da parte A representa 8 pontos no sinal. O mesmo ocorre com a parte B.

Como esses dois conjuntos A e B são os únicos utilizados na análise, pode-se detectar apenas 3 picos em cada intervalo de frequência correspondente, pois como existem 4 amostras, só podem haver, no máximo, 3 inversões entre amostras vizinhas.

Figura FIG3-5: Decomposição do spike

Figura FIG3-3: Spike e filtros passa-baixa aplicados nele: observe que mantendo as frequências abaixo de 2756.25 Hz o spike praticamente não se altera (curva cor verde)

Figura FIG3-6 : Decomposição do overlap

Figura FIG3-4: Overlap e filtros passa-baixa aplicados nele: manter só as frequências abaixo de 2756.25 Hz (curva cor vermelha) causa a perda do formato. É necessário manter as frequências abaixo de 5512.50 Hz (curva cor verde)

TRANSFORMADA SPIKELET4 NÍVEL 3 DE UM SPIKE DE 32 AMOSTRAS

Figura FIG3-7: Decomposição do sinal de 32 amostras (possível spike/overlap) com a transformada SPIKELET4 nível 3.

3.2 - Considerações

O algoritmo de detecção dos *spikes* e *overlaps* aqui apresentado foi implementado na forma de um programa de computador em linguagem C++, nas mesmas condições de *hardware* descritas ao final do capítulo anterior, sendo trabalho do algoritmo desmontar os cabeçalhos e retirar as informações úteis do arquivo que contém o sinal sob análise, gravado no formato de audio wav - 16 bits - mono canal - sem compressão - 44100 Khz. Além disso, o programa cria um arquivo de imagem no formato bmp [BMP1] [Apêndice VII] de 160 x 160 pixels para cada um dos *spikes* / *overlaps* detectados, de forma que o nome do arquivo contém o tipo de sinal detectado e o "instante" em que ele ocorreu. Este programa consta no Apêndice II do presente trabalho.

A parte principal do algoritmo possui ordem de complexidade linear, o que o torna relativamente eficaz, e a função construída para o cômputo da SPIKELET-4 funciona de maneira recursiva, realizando a convolução discreta do sinal original com os coeficientes do filtro, seguido da sub-

amostragem por 2.

Deve ter ficado claro pelo algoritmo exibido que a transformada SPIKELET-4 só é calculada para uma janela de 32 pontos que possivelmente contém um *spike* ou *overlap*. Isso é possível devido aos testes *TESTE_AMP* e *TESTE_FORM* e reduz drasticamente a necessidade de processamento. Além disso, o filtro digital utilizado possui comprimento 4 e, sendo assim, o cálculo da transformada SPIKELET-4 é relativamente rápido, sem problemas para ser realizado mesmo em um processador comum em um computador pessoal.

Outro ponto a observar é que a abordagem proposta pelo algoritmo em questão é totalmente determinística, ao contrário de outros métodos, como comentado anteriormente, que realizam abordagens estatísticas e que possuem certo grau de incerteza. É importante ainda ressaltar que não é utilizado nenhum esquema de *threshold* na transformada SPIKELET-4 calculada, sendo que o único *threshold* levado em consideração é aquele embutido no *TESTE_AMP* e que possui valor pré-determinado de acordo com intervenção manual.

O próximo capítulo descreve a implementação do algoritmo descrito aqui, em tempo real, utilizando um DSP.

Capítulo IV

Processamento *on the fly* do Sinal da Mosca com o DSP EZ-Kit Lite 21065L SHARC - Analog Devices

O processamento *on the fly* (em tempo real) do sinal da mosca, foi realizado com o uso de um DSP EZ-Kit Lite 21065L Sharc, da Analog Devices [DSP2], que está disponível no Laboratório Dipterlab. A seguir, na figura FIG-4 -1, encontra-se o seu diagrama de blocos:

Figura FIG-4-1: Diagrama de blocos do DSP EZ-Kit Lite 21065L Sharc da Analog Devices (figura obtida no datasheet do produto)

O DSP possui as seguintes características básicas:

- capacidade de processamento: 66 MIPS e 198 MFLOPS.
- conversão analógico-digital com precisão de 32 bits.
- clock: 30 Mhz
- arquitetura SISD [COM1] e encapsulamento do processador com

196 pinos, padrão SMD [SMD1].

- memória Interna de 544 Kbits.
- codec integrado AD1819A, 16-bit estéreo AC'97 SoundMAX® CODEC
- conectores para as entradas de *mic-in/line-in* e *line-out* do CODEC
- sete botões de pressionamento para interação com usuário
- seis LED's programáveis (FLAG4, ..., FLAG9)
- interface serial RS-232 para conexão com computador
- compilador VisualDSP++ versão 3.0 para ambiente Windows [WIN1].

As respostas do DSP a cada *spike / overlap* encontrado são:

- cada um dos 3, dentre os 6 leds disponíveis *on-board* no DSP, nomeados de FLAG5, FLAG6 e FLAG7, acendem ou apagam em resposta a cada tipo de sinal encontrado, da seguinte forma (0 = apagado, 1 = acesso):

FLAG5	FLAG6	FLAG7	sinal encontrado
0	0	0	ausência de <i>spike/overlap</i>
0	0	1	<i>overlap</i> acentuado à esquerda
0	1	0	<i>overlap</i> à esquerda
0	1	1	<i>spike</i>
1	0	0	<i>overlap</i> à direita
1	0	1	<i>overlap</i> acentuado à direita
1	1	0	<i>overlap</i> à esquerda e a direita
1	1	1	<i>spike</i> irregular

- no canal *left* de saída, o sinal de entrada é reproduzido, na sua totalidade, sem modificações;
- cada um dos *spikes / overlaps* encontrados (32 amostras) é enviado ao canal *right* de saída, com atraso **fixo** de 32 amostras (aproximadamente 725.62 μ s);
- um pino de I/O digital do DSP (FLAG4) é utilizado para emitir um pulso digital (discriminador) com duração de 1 amostra (aproximadamente 22.68 μ s) e com atraso **fixo** de 32 amostras (aproximadamente 725.62 μ s) em relação ao "instante" que o *spike / overlap* foi detectado (ponto de maior declividade).

Figura FIG-4-2: Tela do ambiente de desenvolvimento integrado do DSP

A figura FIG-4-2 exibe uma tela capturada com o ambiente de desenvolvimento integrado do DSP, executando o algoritmo implementado, que é exatamente o mesmo descrito no Capítulo III e no Apêndice II, com poucas modificações em relação ao código fonte da implementação, devido às regras da linguagem de programação do DSP, que é muito semelhante à linguagem C++.

Os ajustes necessários no ambiente integrado do DSP para executar o *software* são:

- *sampling rate*: 44100 KHz;
- *source*: LINE;
- *gain*: 12.0.

A necessidade de memória no DSP foi de 2400 bytes (módulo *seg_pmco*). O código fonte do algoritmo implementado no DSP consta do Apêndice VIII da presente tese.

O próximo capítulo mostra uma série de resultados relativos às classificações de *spikes* e *overlaps* realizadas pelo programa que implementa o algoritmo, bem como faz uma série de comparações do mesmo algoritmo aplicado a outros filtros.

Capítulo V

TESTES, RESULTADOS, CONCLUSÕES E IDÉIAS FUTURAS

Este capítulo trata de descrever vários testes realizados com a SPIKELET-4 associada ao algoritmo proposto no Capítulo III no que se refere à classificação dos spikes e overlaps. São realizadas também comparações da aplicação do algoritmo com outros filtros, onde diversas conclusões são tiradas e idéias futuras são propostas. Os testes descritos não se referem à abordagem em tempo real, muito embora possam ser estendidos à ela.

Vários testes foram realizados com o algoritmo proposto associado a SPIKELET-4. As figuras FIG5-1 e FIG5-2 mostram exemplos de detecção de diversos *spikes* e *overlaps* e a figura FIG5-3 mostra a detecção de um *spike* irregular, comprovando a eficiência do algoritmo.

As figuras FIG5-4 e FIG5-5 exibem, respectivamente, para efeitos de comparação, as Transformadas Daubechies-4 e SPIKELET-4 de um mesmo *spike* de 32 amostras (*SPIKE S*). Já as figuras FIG5-6 e FIG5-7 exibem as mesmas transformadas de um mesmo *overlap* (*OVERLAPED SPIKE OS*).

Para facilitar as comparações, as figuras FIG5-8 e FIG5-9 mostram as amostras 0 a 3 (frequências de 0 a 2756.25 Hz) das transformadas Daubechies-4 e SPIKELET-4 do *spike* em questão e as figuras FIG5-10 e FIG5-11 mostram as mesmas transformadas do *OVERLAPED SPIKE OS*. Já, as figuras FIG5-12 a FIG5-15, mostram as amostras 4 a 7 (frequências de 2576.25 a 5512.5 Hz) dos mesmos sinais, respectivamente.

Note que a transformada SPIKELET-4 se faz absolutamente necessária para a aplicação do algoritmo proposto, pois as posições dos *zero-crossings* presentes nela, para um determinado sinal, diferem dos da transformada de Daubechies do mesmo sinal, em especial no que se refere às transformadas do *OVERLAPED SPIKE OS*. Isso ocorre devido a porção do *spike* sob análise

para a qual a SPIKELET-4 é mais "sensível" (apresenta maior amplitude) do que a transformada de Daubechies. As figuras FIG5-16 até FIG5-23 mostram diversos exemplos da aplicação do algoritmo com outros filtros:

- Haar 2: em geral é utilizada para detecção de descontinuidades, possui suporte compacto, é ortogonal, simétrica e descontínua [HAA1];
- Daubechies 4 (menor suporte para o qual é definido): possui suporte compacto, é ortogonal, não simétrica e contínua [DAU1];
- Daubechies 20: idem à anterior, com suporte 20;
- Coiflet 6 (menor suporte para o qual é definido): ambas as *wavelets* pai e mãe possuem 2, 4, 6, 8 ou 10 momentos nulos e são propícias para a análise de transientes rápidos em sinais [COI1];
- Symmlet 8 (menor suporte para o qual é definido): possui 4, 6, 8 ou 10 momentos nulos, é quase simétrica, contínua, ortogonal e de suporte compacto [SYM1];
- Battle-and-Lemarié 12 (menor suporte para o qual é definido): decaimento exponencial baseado em uma função *spline* [BAT1];
- Beylkin 18 (menor suporte para o qual é definido) coloca as raízes da resposta em frequência no eixo real, próximas ao limite de Nyquist [BEY1];
- Vaidyanathan 24 (menor suporte para o qual é definido): filtro otimizado para processamento de sinais de fala, que não satisfaz nenhuma condição de momentos nulos [VAI1];

Muito embora tais filtros não tenham sido otimizados exatamente para

realizar a classificação dos sinais propostos no presente trabalho, ainda assim as comparações são válidas para mostrar que, nem o filtro de suporte menor que 4 (Haar: suporte 2) nem os filtros com os maiores suportes testados (Daubechies: suporte 20 e Vaidynathan: suporte 24), conseguem detectar todos os casos de *spikes* e *overlaps*, já que nada, a não ser os testes realizados, poderia garantir que alguma dessas outras *wavelets* associadas ao algoritmo proposto conseguiriam ou não classificar todos os sinais de interesse .

Em especial, um resultado interessante obtido com essas comparações com outras *wavelets* é que o filtro mais seletivo (que separa melhor as frequências) não é o que distingue melhor os padrões.

Em um âmbito geral, o trabalho desenvolvido apresentou resultados bastante motivantes. Nenhum dos outros algoritmos revistos e nenhuma outra transformada *wavelet* foram capazes de detectar, classificar e precisar a localização dos *spikes*, dos 5 tipos de *overlaps* e dos *spikes* irregulares com um algoritmo de baixa complexidade computacional, determinístico e que pode ser implementado em tempo real.

Uma idéia futura para a continuidade do presente trabalho é a adaptação da SPIKELET-4, e do algoritmo proposto para detectar e classificar múltiplos *overlaps* em cada lado (direita e esquerda) do pico do *spike*, talvez com a utilização de uma decomposição *wavelet packet* e alguma modificação no algoritmo. Outra idéia consiste em pesquisar se existe outra maneira mais eficiente de definir o “instante” em que o *spike* (ou *overlap*) ocorre, que não seja o ponto de maior declividade.

Figura FIG5-1: exemplo de sinais detectados pelo algoritmo associado a SPIKELET-4

Figura FIG5-2: exemplo de sinais detectados pelo algoritmo associado a SPIKELET-4

Figura FIG5-3: eficiência do algoritmo associado a SPIKELET-4 para detecção de um spike "irregular"

Transformada DAUBECHIES-4 nível 3 do OVERLAPED-SPIKE OS

Figura FIG5-6: Transformada de Daubechies-4 do overlap OS

Transformada de DAUBECHIES-4 nível 3 do SPIKE S

Figura FIG5-4: Transformada de Daubechies-4 do spike S

Transformada SPIKELET-4 nível 3 do OVERLAPED-SPIKE OS

Figura FIG5-7: Transformada SPIKELET-4 do overlap OS

Transformada SPIKELET-4 nível 3 do SPIKE S

Figura FIG5-5: Transformada de SPIKELET-4 do spike S

Figura FIG5-10: Amostras 0 a 3 da Transf. Daub-4 do overlap OS

Figura FIG5-8: Amostras 0 a 3 da Transf. Daubechies-4 do spike S

Figura FIG5-11: Amostras 0 a 3 ransf. SPIKELET-4 do overlap OS

Figura FIG5-9: Amostras 0 a 3 da Transf. SPIKELET-4 do spike S

AMOSTRAS 4, 5, 6 e 7 da Transformada DAUBECHIES-4
nível 3 do OVERLAPED SPIKE OS

Figura FIG5-14:Am. 4 a 7 da Daubechies-4 do overlap OS

AMOSTRAS 4, 5, 6 e 7 da Transformada DAUBECHIES-4
nível 3 do SPIKE S

Figura FIG5-12:Amostras 4 a 7 da Daubechies-4 do spike S

AMOSTRAS 4, 5, 6 e 7 da Transformada SPIKELET-4
nível 3 do OVERLAPED SPIKE OS

Figura FIG5-15:Am. 4 a 7 da SPIKELET4 do overlap OS

AMOSTRAS 4, 5, 6, e 7 da Transformada SPIKELET-4
nível 3 do SPIKE S

Figura FIG5-13:Amostras 4 a 7 da SPIKELET4 do spike S

RESULTADOS DA IDENTIFICAÇÃO COM HAAR-2

DISTINGUE APENAS 4 CASOS EMBORA TENHA SUPORTE MENOR QUE A SPIKELET-4

Figura FIG5-16: Aplicação do algoritmo com outra transformada wavelet

RESULTADOS DA IDENTIFICAÇÃO COM DAUBECHIES-4

DISTINGUE APENAS 4 CASOS E TEM SUPORTE IGUAL A SPIKELET-4

Figura FIG5-17: Aplicação do algoritmo com outra transformada wavelet

RESULTADOS DA IDENTIFICAÇÃO COM DAUBECHIES-20 DISTINGUE APENAS 2 CASOS E TEM SUPORTE 5 VEZES MAIOR QUE A SPIKELET-4

Figura FIG5-18: Aplicação do algoritmo com outra transformada wavelet

RESULTADOS DA IDENTIFICAÇÃO COM COIFLET-6

DISTINGUE APENAS 5 CASOS E TEM SUPORTE 1,5 VEZES MAIOR QUE A SPIKELET-4

Figura FIG5-19: Aplicação do algoritmo com outra transformada wavelet

RESULTADOS DA IDENTIFICAÇÃO COM SYMMLET-8

DISTINGUE APENAS 3 CASOS E TEM SUPORTE 2 VEZES MAIOR QUE A SPIKELET-4

Figura FIG5-20: Aplicação do algoritmo com outra transformada wavelet

RESULTADOS DA IDENTIFICAÇÃO COM BATTLE-LEMARIE-12 DISTINGUE APENAS 5 CASOS E TEM SUPORTE 3 VEZES MAIOR QUE A SPIKELET-4

Figura FIG5-21: Aplicação do algoritmo com outra transformada wavelet

RESULTADOS DA IDENTIFICAÇÃO COM BEYLKIN-18 DISTINGUE APENAS 4 CASOS E TEM SUPORTE 4,5 VEZES MAIOR QUE A SPIKELET-4

Figura FIG5-22: Aplicação do algoritmo com outra transformada wavelet

RESULTADOS DA IDENTIFICAÇÃO COM VAIDYANATHAN-24 DISTINGUE APENAS 4 CASOS E TEM SUPORTE 6 VEZES MAIOR QUE A SPIKELET-4

Figura FIG5-23: Aplicação do algoritmo com outra transformada wavelet

APÊNDICE I

Código Fonte em Linguagem C++ para o Cômputo da SPIKELET-4

```
////////////////////////////////////  
///SISTEMA.CPP////////////////////////////////////  
///A PARTIR DO SPIKE MEDIO PADRAO, CONSTROI O FILTRO SPIKELET COM SUPORTE VARIÁVEL////////////////////////////////  
///DE ACORDO COM A TECNICA DEFINIDA////////////////////////////////  
///By Rodrigo C. Guido - IFSC - USP – 2003////////////////////////////////  
////////////////////////////////////  
#include<iostream.h>  
#include<stdio.h>  
#include<math.h>  
#define tamsup 4 //escolher o tamanho de suporte desejado  
  
main()  
{  
cout<<"\n\n===== \n";  
cout<<"\n\nCONSTRUCAO DOS COEFICIENTES DO FILTRO WAVELET SPIKELET\n\n";  
cout<<"\n\n===== \n";  
int tamf=32;  
cout<<"Tamanho do sinal f: "<<tamf;  
cout<<"\n----- \n ";  
cout<<"\n";  
double r;  
double* f=new double[tamf];  
f[0]=4533;  
f[1]=5562;  
f[2]=6517;  
f[3]=7358;  
f[4]=7858;  
f[5]=7278;  
f[6]=4903;  
f[7]=187;  
f[8]=-6648;  
f[9]=-14330;  
f[10]=-21331;  
f[11]=-26532;  
f[12]=-29400;  
f[13]=-29865; //pico negativo
```

```

f[14]=-28071;
f[15]=-24728;
f[16]=-20572;
f[17]=-15930;
f[18]=-11207;
f[19]=-6682;
f[20]=-2644;
f[21]=807;
f[22]=3627;
f[23]=5771;
f[24]=7318;
f[25]=8302;
f[26]=8771;
f[27]=8772;
f[28]=8421;
f[29]=7822;
f[30]=7044;
f[31]=6222;
for(int m=0;m<tamf;m++)
{
    f[m]/=29865.0;//normaliza
    cout<<"\nElemento "<<m<<" de f: ";
    printf("%.20f",f[m]);
}
double a[(int)(3*tamsup/2)+1][tamsup];
cout<<"\n\n\n";
for(int j=0;j<tamsup;j++)
{
    for(int k=0;k<tamsup;k++)
    {
        r=0;
        for(int i=0;i<(int)(tamf/2);i++)
            r+=f[(i*2+j)%(tamf-1)]*f[(i*2+k)%(tamf-1)];
        printf(" %.20f h%d",r,k);
        a[j][tamsup-k-1]=r;
    }
    cout<<"\n=====\\n";
}
for(int i=0;i<tamsup;i++)
    for(int j=1;j<tamsup;j+=2)
        a[i][j]*=-1;
for(int j=tamsup;j<(int)(3*tamsup/2);j++)
    for(int k=0;k<tamsup;k++)
        a[j][k]=pow(-1,k)*pow(k,j-tamsup);
for(int k=0;k<tamsup;k++)
    a[(int)(3*tamsup/2)][k]=1;

```

```

cout<<"\n\nMATRIZ A: \n\n";
for(int i=0;i<(int)(3*tamsup/2)+1;i++)
{
    for(int j=0;j<tamsup;j++)
        printf(" %.20f",a[i][j]);
    cout<<"\n";
}
cout<<"\n===== \n";
//=====
//=====
double b[(int)(3*tamsup/2)+1][1];

for(int i=0;i<tamsup;i++)
    b[i][0]=0;
for(int i=tamsup;i<(int)(3*tamsup/2)+1;i++)
    b[i][0]=0;
b[(int)(3*tamsup/2)][0]=2;
cout<<"\n\nMATRIZ B: \n\n";
for(int i=0;i<(int)(3*tamsup/2)+1;i++)
    printf("\n%.20f",b[i][0]);
cout<<"\n===== \n";
//=====
//=====
double at[tamsup][(int)(3*tamsup/2)+1];
for(int i=0;i<tamsup;i++)
    for(int j=0;j<(int)(3*tamsup/2)+1;j++)
        at[i][j]=a[j][i];
cout<<"\n\nMATRIZ A_TRANSPOSTA: \n\n";
for(int i=0;i<tamsup;i++)
{
    for(int j=0;j<(int)(3*tamsup/2)+1;j++)
        printf(" %.20f",at[i][j]);
    cout<<"\n";
}
cout<<"\n===== \n";
//=====
//=====
double at_a[tamsup][tamsup];
for(int i=0;i<tamsup;i++)
    for(int j=0;j<tamsup;j++)
        at_a[i][j]=0;
for(int i=0;i<tamsup;i++)
    for(int j=0;j<tamsup;j++)
        for(int k=0;k<(int)(3*tamsup/2)+1;k++)
            at_a[i][j]+=at[i][k]*a[k][j];
cout<<"\n\nMATRIZ A_TRANSPOSTA X A: \n\n";

```

```

for(int i=0;i<tamsup;i++)
{
    for(int j=0;j<tamsup;j++)
        printf("%.20f, ",at_a[i][j]);
    cout<<"\n";
}
cout<<"\n=====\\n";

double at_b[tamsup][1];
for(int i=0;i<tamsup;i++)
    at_b[i][0]=0;
for(int i=0;i<tamsup;i++)
    for(int k=0;k<(int)(3*tamsup/2)+1;k++)
        at_b[i][0]+=at_a[i][k]*b[k][0];
cout<<"\\n\\nMATRIZ A_TRANSPOSTA X B:\\n\\n";
for(int i=0;i<tamsup;i++)
    printf("\\n%.20f",at_b[i][0]);
cout<<"\\n=====\\n";
cout<<"\\n=====\\n";
cout<<"\\n=====\\n";

double maior=at_a[0][0];
for(int i=0;i<tamsup;i++)
{
    for(int j=0;j<tamsup;j++)
        if(at_a[i][j]>maior)
            maior=at_a[i][j];
    for(int j=0;j<tamsup;j++)
        at_a[i][j]/=maior;
    at_b[i][0]/=maior;
}

cout<<"\\n=====\\n";
////////////////////////////////////
//escalonando o sistema
double m;
int lcpivo=0;
do
{
    for(int k=lcpivo;k<tamsup-1;k++)
    {
        m=(-at_a[k+1][lcpivo]/at_a[lcpivo][lcpivo]);
        for(int j=lcpivo;j<tamsup;j++)
            at_a[k+1][j]+=at_a[lcpivo][j]*m;
        at_b[k+1][0]+=at_b[lcpivo][0]*m;
    }
}

```

```

        lcpivo++;
    } while(lcpivo<tamsup-1);
cout<<"\n\nSISTEMA FINAL ***ESCALONADO*** A SER RESOLVIDO: \n\n";
cout<<"\n\nMATRIZ A_TRANSPOSTA_DE_A X A ESCALONADA: \n\n";
for(int i=0;i<tamsup;i++)
{
    for(int j=0;j<tamsup;j++)
        printf("%.20f, ",at_a[i][j]);
    cout<<"\n";
}
cout<<"\n===== \n";
cout<<"\n\nMATRIZ A_TRANSPOSTA_DE_A X B ESCALONADA: \n\n";
for(int i=0;i<tamsup;i++)
    printf("\n%.20f",at_b[i][0]);
cout<<"\n===== \n";

////////////////////////////////////
//solucao do sistema escalonado
cout<<"\n\nSOLUCAO DO SISTEMA FINAL ESCALONADO:(coeficientes scaling-filtro passa-baixa)\n";
double* h=new double[tamsup];
for(int i=0;i<tamsup;i++)
    h[i]=at_b[i][0];
h[tamsup-1]=at_b[tamsup-1][0]/at_a[tamsup-1][tamsup-1];
for(int i=tamsup-2;i>=0;i--)
{
    for(int j=i;j<tamsup;j++)
        h[j]=at_a[i][j+1]*h[j+1];
    h[i]=at_a[i][i];
}
double soma_h2=0;
for(int i=0;i<tamsup;i++)
{
    printf("\n%.20f",h[i]);
    soma_h2+=h[i]*h[i];
}
cout<<"\n\nSOLUCAO DO SISTEMA FINAL ESCALONADO:(coeficientes wavelet-filtro passa-alta)\n";
for(int i=tamsup-1;i>=0;i--)
    if(i%2!=0)
        printf("\n%.20f",h[i]);
    else
        printf("\n%.20f",-h[i]);
printf("\n\nEnergia do sinal: %.20f",soma_h2);
cout<<"\n\n";
}

```

APÊNDICE II

Código Fonte em Linguagem C++ para Análise e Classificação dos *Spikes* e *Overlaps*

```
/////////////////////////////////////////////////////////////////
//FINDSPIKE.CPP/////////////////////////////////////////////////////////////////
//DETECTA OS SPIKES E OS OVERLAPED SPIKES A PARTIR DO ARQUIVO DE DADOS WAV/////////////////////////////////////////////////////////////////
//By Rodrigo C. Guido - IFSC - USP – 2003/////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////
#include<iostream.h>
#include<stdio.h>
#include<math.h>
#include<string.h>

//-----
main(int i,char* n[])
{
short converte2de8para1de16(unsigned char, unsigned char);
void transformada_wavelet_nivel_k(double*, long, int);
void transformada_spikelet_nivel_k(double*, long, int);
char tipo_spike_descoberto[200];
tipo_spike_descoberto[0]='\0';
FILE* fr_wav;
if(((fr_wav=fopen(n[1],"rb"))!=NULL))
{
struct
{
unsigned char riff[4];
unsigned long len;
} riff_header;
fread(&riff_header,sizeof(riff_header),1,fr_wav);
cout<<"\nArquivo do tipo: "<<riff_header.riff[0]<<riff_header.riff[1]<<riff_header.riff[2]<<riff_header.riff[3];
cout<<"\nTamanho excluindo header: "<<riff_header.len;

/////////////////////////////////////////////////////////////////

unsigned char wave[4];
fread(&wave,sizeof(wave),1,fr_wav); /////
cout<<"\nSub-Tipo: "<<wave[0]<<wave[1]<<wave[2]<<wave[3];
```

```

////////////////////////////////////
struct
{
    unsigned char id[4];
    unsigned long len;
} riff_chunk;

fread(&riff_chunk,sizeof(riff_chunk),1,fr_wav);
cout<<"\nIdentificador: "<<riff_chunk.id[0]<<riff_chunk.id[1]<<riff_chunk.id[2]<<riff_chunk.id[3];
cout<<"\nComprimento do chunk apos header: "<<riff_chunk.len;

////////////////////////////////////
struct
{
    unsigned short formattag;
    unsigned short numberofchannels;
    unsigned long samplingrate;
    unsigned long avgbytespersecond;
    unsigned short blockalign;
} wave_chunk;

fread(&wave_chunk,sizeof(wave_chunk),1,fr_wav);
cout<<"\nCategoria do formato: "<<wave_chunk.formattag;
cout<<"\nquantidade de canais: "<<wave_chunk.numberofchannels;
cout<<"\nTaxa de amostragem: "<<wave_chunk.samplingrate;
cout<<"\nMedia do num. de bps: "<<wave_chunk.avgbytespersecond;
cout<<"\nAlinhamento do bloco em bytes: "<<wave_chunk.blockalign;

////////////////////////////////////

if(wave_chunk.formattag==1) //PCM
{
    int resolucao=(wave_chunk.avgbytespersecond * 8)/(wave_chunk.numberofchannels *
wave_chunk.samplingrate);
    cout<<"\nResolucao: "<<resolucao;

    struct
    {
        unsigned char data[4];
        unsigned long chunk_size;
    } header_data_chunk;

    fread(&header_data_chunk,sizeof(header_data_chunk),1,fr_wav);
    cout<<"\nIdentificacao:
"<<header_data_chunk.data[0]<<header_data_chunk.data[1]<<header_data_chunk.data[2]<<header_da
ta_chunk.data[3];

    printf("\nTamanho do chunk de dados: %d",header_data_chunk.chunk_size);
    printf("\nquantidade de frames para amostrar:

```

```

%d",(long)((double)header_data_chunk.chunk_size/(double)wave_chunk.blockalign));

    long tamanho_da_janela=2;
    while(tamanho_da_janela<=((long)((double)header_data_chunk.chunk_size/(double)wave_chun
k.blockalign))
        tamanho_da_janela*=2;
    tamanho_da_janela/=2;
    printf("\nTamanho da janela: %d",tamanho_da_janela);
    if((resolucao==16) && (wave_chunk.numberofchannels==1))
    {
        unsigned char waveformdata_lsb, waveformdata_msb;
        double* amostras_originais = new
double[(header_data_chunk.chunk_size/wave_chunk.blockalign)];
        for(long i=0;i<((header_data_chunk.chunk_size/wave_chunk.blockalign);i++)
        {
            fread(&waveformdata_lsb,sizeof(waveformdata_lsb),1,fr_wav);
            fread(&waveformdata_msb,sizeof(waveformdata_msb),1,fr_wav);
            amostras_originais[i]=(double)converte2de8para1de16(waveformdata_lsb,waveformat
a_msb);
        }
    }
    //////////////////////////////////////
    long i=13;
    int k=0;
    int quantidade_de_spikes=0;
    double spikelet[32];
    cout<<"\n\n" ;
    unsigned char mpb[160][160], mpg[160][160], mpr[160][160];
    long imenor;
    double menor;
    while(i<((header_data_chunk.chunk_size/wave_chunk.blockalign)-18)
    {
        if((((int)amostras_originais[i-1])>=((int)amostras_originais[i]))
        && (((int)amostras_originais[i+1])>=((int)amostras_originais[i]))
        && (((int)amostras_originais[i])<=-6600))
        {
            imenor=i;
            menor=amostras_originais[i];
            for(long b=i;b<i+32;b++)
                if((amostras_originais[b]<menor)&&(amostras_originais[i+1]>amostras_originais[i]
)&&(amostras_originais[i-1]>amostras_originais[i]))
                {
                    menor=amostras_originais[b];
                    imenor=b;
                }
            i=imenor;
            int linha, próxima_linha,y;

```



```

for(int linha=0;linha<160;linha++)
    for(int coluna=0;coluna<160;coluna++)
        {
            mpb[linha][coluna]=255;
            mpg[linha][coluna]=255;
            mpr[linha][coluna]=255;
        }
k=0;
for(int coluna=0;coluna<160;coluna+=(int)(160/32))
    {
        linha=(int)(160+((160.0/65535.0)*(amostras_originais[k+i-13]-
32767)));

        mpb[linha][coluna]=0;
        mpg[linha][coluna]=0;
        mpr[linha][coluna]=0;
        próxima_linha=(int)(160+((160.0/65535.0)*(amostras_originais[
k+1+i-13]-32767)));

        if(próxima_linha>linha)
            for(int x=linha;x<próxima_linha;x++)
                {
                    y=(int)(round(((160.0/32.0)/(double)(próxi
ma_linha-linha))*(x-linha)+coluna));

                    mpb[x][y]=0;
                    mpg[x][y]=0;
                    mpr[x][y]=0;
                }
            else
                for(int x=próxima_linha;x<linha;x++)
                    {
                        y=(int)(round(((160.0/32.0)/(double)(próxi
ma_linha-linha))*(x-linha)+coluna));

                        mpb[x][y]=0;
                        mpg[x][y]=0;
                        mpr[x][y]=0;
                    }
                k++;
            }

        long j=i-1;
        int maior=abs((amostras_originais[j-1]-amostras_originais[j]));
        int imaior=j;
        while((amostras_originais[j-1]>amostras_originais[j]) &&
(amostras_originais[j]>amostras_originais[j+1]))
            {
                if(abs(amostras_originais[j]-1)-

```

```

amostras_originais[j])>maior)

        {
            maior=abs(amostras_originais[j]-1] -

amostras_originais[j]);

            imaior=j;
        }

        j--;
    }

    //i candidato a ser o pico do spike, seja ele positivo ou
negativo

    k=0; //obter o possivel spike para anAise, centrando o pico
na amostra nr. 14

    for(int h=i-13;h<=i+18;h++)
    {
        spikelet[k]=amostras_originais[h]/32768.0;
        k++;
    }

    //fazer a transformada spikelet nivel 3 do possivel spike e
guardar no proprio vetor

    transformada_spikelet_nivel_k(&spikelet[0],32,3);
    if((spikelet[1]*spikelet[2]<0)&&(spikelet[4]*spikelet[5]<0)&&(spi
kelet[5]*spikelet[6]>0)&&(spikelet[6]*spikelet[7]>0))

    {
        tipo_spike_descoberto[0]='\0';
        strcat(tipo_spike_descoberto,"overlap_acentuado
_a_esquerda");

        printf("\nOVERLAP ACENTUADO A ESQUERDA....Nro
%d. Detectado ponto de > inclinacao no instante %f segundos com amplitude
%d",quantidade_de_spikes,(double)((double)(imaior)/(double)(wave_chunk.samplingrate)),(int)amostras_
originais[i]);

        quantidade_de_spikes++;
        i+=30;
    }

    else if

    ((spikelet[1]*spikelet[2]<0)&&(spikelet[4]*spikelet[5]<0)&&(spikelet[5]*spikelet[6]<0)&&(spikelet[6]*spikelet[7]>0
))

    {
        tipo_spike_descoberto[0]='\0';
        strcat(tipo_spike_descoberto,"overlap_a_esquerda
");

        printf("\nOVERLAP A ESQUERDA.....Nro %d.
Detectado ponto de > inclinacao no instante %f segundos com amplitude
%d",quantidade_de_spikes,(double)((double)(imaior)/(double)(wave_chunk.samplingrate)),(int)amostras_
originais[i]);

        quantidade_de_spikes++;
        i+=30;
    }

```

```

    }
    else if
((spikelet[1]*spikelet[2]<0)&&(spikelet[4]*spikelet[5]<0)&&(spikelet[5]*spikelet[6]<0)&&(spikelet[6]*spikelet[7]<0
))

    {
        tipo_spike_descoberto[0]='\0';
        strcat(tipo_spike_descoberto,"spike");
        printf("\nSPIKE.....Nro %d. Detectado
ponto de > inclinacao no instante %f segundos com amplitude
%d",quantidade_de_spikes,(double)((double)(imaior)/(double)(wave_chunk.samplingrate)),(int)amostras_
originais[i]);

        quantidade_de_spikes++;
        i+=30;
    }

    else if
((spikelet[1]*spikelet[2]<0)&&(spikelet[4]*spikelet[5]>0)&&(spikelet[5]*spikelet[6]<0)&&(spikelet[6]*spikelet[7]<0
))

    {
        tipo_spike_descoberto[0]='\0';
        strcat(tipo_spike_descoberto,"overlap_a_direita");
        printf("\nOVERLAP A DIREITA.....Nro %d.
Detectado ponto de > inclinacao no instante %f segundos com amplitude
%d",quantidade_de_spikes,(double)((double)(imaior)/(double)(wave_chunk.samplingrate)),(int)amostras_
originais[i]);

        quantidade_de_spikes++;
        i+=30;
    }

    else if
((spikelet[1]*spikelet[2]<0)&&(spikelet[4]*spikelet[5]>0)&&(spikelet[5]*spikelet[6]>0)&&(spikelet[6]*spikelet[7]<0
))

    {
        tipo_spike_descoberto[0]='\0';
        strcat(tipo_spike_descoberto,"overlap_acentuado
_a_direita");

        printf("\nOVERLAP ACENTUADO A DIREITA.....Nro
%d. Detectado ponto de > inclinacao no instante %f segundos com amplitude
%d",quantidade_de_spikes,(double)((double)(imaior)/(double)(wave_chunk.samplingrate)),(int)amostras_
originais[i]);

        quantidade_de_spikes++;
        i+=30;
    }

    else if
((spikelet[1]*spikelet[2]<0)&&(spikelet[4]*spikelet[5]<0)&&(spikelet[5]*spikelet[6]>0)&&(spikelet[6]*spikelet[7]<0
))

    {

```

```

tipo_spike_descoberto[0]='\0';
strcat(tipo_spike_descoberto,"overlap_a_esquerda
_e_a_direita");

printf("\nOVERLAP A ESQUERDA E A DIREITA...Nro
%d. Detectado ponto de > inclinacao no instante %f segundos com amplitude
%d",quantidade_de_spikes,(double)((double)(imaior)/(double)(wave_chunk.samplingrate)),(int)amostras_
originais[i]);

quantidade_de_spikes++;
i+=30;
}
else
{
tipo_spike_descoberto[0]='\0';
strcat(tipo_spike_descoberto,"spike_deformado");
printf("\nSPIKE DEFORMADO DETECTADO.....Nro
%d. Detectado ponto de > inclinacao no instante %f segundos com amplitude
%d",quantidade_de_spikes,(double)((double)(imaior)/(double)(wave_chunk.samplingrate)),(int)amostras_
originais[i]);

quantidade_de_spikes++;
i+=30;
}
FILE* fw_bmp;
char string[12];
char string_temp[12];
int pos=0;
int pos_temp=0;
int
valor_int=(int)((((double)((double)(imaior)/(double)(wave_chunk.samplingrate))))*1000000000);
for(int z=0;z<9;z++)
{
string_temp[pos_temp]=(char)((valor_int%10)+48);
valor_int=(int)(valor_int/10);
pos_temp++;
}
pos_temp--;
do
{
string[pos]=string_temp[pos_temp];
pos++;
pos_temp--;
} while(pos_temp>=0);
string[pos]='\0';
strcat(tipo_spike_descoberto,"_com_pico_principal_em_0.");
strcat(tipo_spike_descoberto,string);
strcat(tipo_spike_descoberto,"_segundos");
strcat(tipo_spike_descoberto,".bmp");

```

```

fw_bmp=fopen(tipo_spike_descoberto,"wb");
char bmp_b='B';
char bmp_m='M';
int bmp_filesize=76854;
int bmp_reserved=0;
int bmp_offset=54;
int bmp_sizeofheader=40;
int bmp_width=160;
int bmp_height=160;
short bmp_planes=1;
short bmp_depth=24;
int bmp_compressiontype=0;
int bmp_sizeofimage=76800;
int bmp_hr=2834;
int bmp_vr=2834;
int bmp_nc=0;
int bmp_nic=0;

fwrite(&bmp_b,sizeof(bmp_b),1,fw_bmp);
fwrite(&bmp_m,sizeof(bmp_m),1,fw_bmp);
fwrite(&bmp_filesize,sizeof(bmp_filesize),1,fw_bmp);
fwrite(&bmp_reserved,sizeof(bmp_reserved),1,fw_bmp);
fwrite(&bmp_offset,sizeof(bmp_offset),1,fw_bmp);
fwrite(&bmp_sizeofheader,sizeof(bmp_sizeofheader),1,fw_b

mp);

fwrite(&bmp_width,sizeof(bmp_width),1,fw_bmp);
fwrite(&bmp_height,sizeof(bmp_height),1,fw_bmp);
fwrite(&bmp_planes,sizeof(bmp_planes),1,fw_bmp);
fwrite(&bmp_depth,sizeof(bmp_depth),1,fw_bmp);
fwrite(&bmp_compressiontype,sizeof(bmp_compressiontype

),1,fw_bmp);

fwrite(&bmp_sizeofimage,sizeof(bmp_sizeofimage),1,fw_bmp);

p);

fwrite(&bmp_hr,sizeof(bmp_hr),1,fw_bmp);
fwrite(&bmp_vr,sizeof(bmp_vr),1,fw_bmp);
fwrite(&bmp_nc,sizeof(bmp_nc),1,fw_bmp);
fwrite(&bmp_nic,sizeof(bmp_nic),1,fw_bmp);

for(int linha=0;linha<160;linha++)
    for(int coluna=0;coluna<160;coluna++)
        fprintf(fw_bmp,"%c%c%c",mpb[linha][colu

na],mpg[linha][coluna],mpr[linha][coluna]);

fclose(fw_bmp);
}

i++;

```

```

        }
////////////////////////////////////
    }
    else
    {
        cout<<"Resolucao ou numero de canais invalido(s)";
        exit(0);
    }
}
else
    cout<<"FORA DO FORMATO PCM...";
fclose(fr_wav);
}
else
    cout<<"Arquivo wav nao existe ou nao pode ser aberto";
cout<<"\n\n\n";
}
//-----
void transformada_spikelet_nivel_k(double* f, long n, int nivel)
{
    double sf[] = {
        0.26964482896235847376,
        0.76237548312490721614,
        0.73270322306815560687,
        0.23524044702452745481
    };

    int csf=sizeof(sf)/sizeof(double);
    double *wf=new double[csf];
    for(int i=0;i<csf;i++)
    {
        wf[i]=sf[csf-i-1];
        if(i%2!=0)
            wf[i]*=-1;
    }
    int cwf=csf;
    long j=0;
    double* t = new double[n];
    for(long i=0;i<n;i+=2) //trend
    {
        t[j]=0;
        for(long k=0;k<csf;k++)
            t[j]+=f[(i+k)%n]*sf[k];
        j++;
    }
    for(long i=0;i<n;i+=2) //fluctuation

```

```

    {
        t[j]=0;
        for(long k=0;k<cwf;k++)
            t[j]+=f[(i+k)%n]*wf[k];
        j++;
    }
for(long i=0;i<n;i++)
    f[i]=t[i];
nivel--;
n/=2;
delete(t);
if(nivel>0)
    transformada_spikelet_nivel_k(&f[0],n,nivel);
}
//-----
short converte2de8para1de16(unsigned char lsb, unsigned char msb)
{
return(((msb&0x80)>>7)*(32768) +
        ((msb&0x40)>>6)*(16384) +
        ((msb&0x20)>>5)*(8192) +
        ((msb&0x10)>>4)*(4096) +
        ((msb&0x08)>>3)*(2048) +
        ((msb&0x04)>>2)*(1024) +
        ((msb&0x02)>>1)*(512) +
        ((msb&0x01))*(256) +
        ((lsb&0x80)>>7)*(128) +
        ((lsb&0x40)>>6)*(64) +
        ((lsb&0x20)>>5)*(32) +
        ((lsb&0x10)>>4)*(16) +
        ((lsb&0x08)>>3)*(8) +
        ((lsb&0x04)>>2)*(4) +
        ((lsb&0x02)>>1)*(2) +
        (lsb&0x01));
}
//-----
//-----

```

APÊNDICE III

Funcionamento Básico do Sistema Visual da Mosca

Apenas a título de informação, este Apêndice descreve o princípio básico de funcionamento do sistema visual da mosca.

Dentro do duto óptico da mosca, as informações são recebidas pelos omatídeos do olho composto, passando pelos fotorreceptores e estágios de processamento posteriores (lamina, medula, lóbula e placa lobular), conforme exhibe a figura FIG1-AP3.

Figura FIG1-AP3 – Cérebro da mosca, com destaque (acima à direita) para o percurso da informação a partir do olho. Figura obtida em <http://flybrain.neurobio.arizona.edu/> em 07/2003

A figura FIG2-AP3 apresenta com detalhes a placa lobular, onde existem cerca de 20 neurônios gigantes monitorando o movimento em torno de três eixos: horizontal (HS), vertical (VS) e longitudinal (GF) [HAU1].

Figura FIG2-AP3 – Vista ampliada da placa lobular. Figura obtida em <http://flybrain.neurobio.arizona.edu/Flybrain/html/atlas/cobalt/sha.html> em 07/2003

Dentro do complexo HS, o neurônio de largo campo H1 é responsável pela integração dos estímulos gerados pela movimentação em torno de um eixo vertical. De fato, existem dois neurônios H1 na mosca, sendo um em cada hemisfério, cada um respondendo a estímulos numa só direção: o H1 ipsilateral e o H1 contralateral. Na figura FIG3-AP3 pode-se observar, dentre os equipamentos, o padrão dos traços verticais que sensibilizam o neurônio H1.

Figura FIG3-AP3: padrão dos traços verticais que sensibilizam o neurônio H1 da mosca

APÊNDICE IV

Fotos dos experimentos realizados no Laboratório

Dipteralab – IFSC – USP

Figura FIG-AP4-1: vista ampliada da mosca e do eletrodo durante um experimento.

Figura FIG-AP4-2: osciloscópio com um spike capturado em tempo real.

Figura FIG-AP4-3: equipamentos utilizados no Laboratório Dipterlab preparado para um experimento.

APÊNDICE V

Coeficientes dos Filtros utilizados nas experiências (passa-baixa - na ordem $h_0, h_1, \dots, h_n$)

1 – Battle-Lemarié – suporte 12

0.578163, 0.280931, -0.0488618, -0.0367309, 0.012003, 0.00706442, -0.00274588, -0.00155701, 0.000652922,
0.000361781, -0.000158601, -0.0000867523

2 – Beylkin – suporte 18

0.099305765374, 0.424215360813, 0.699825214057, 0.449718251149, -0.110927598348, -0.264497231446,
0.026900308804, 0.155538731877, 0.017520746267, -0.088543630623, 0.019679866044, 0.042916387274,
-0.017460408696, -0.014365807969, 0.010040411845, 0.001484234782, -0.002736031626, 0.000640485329

3 – Coiflet – suporte 6

0.038580777748, -0.126969125396, -0.077161555496, 0.607491641386, 0.745687558934, 0.226584265197

4 – Daubechies – suporte 4

0.482962913145, 0.836516303738, 0.224143868042, -0.129409522551

5 – Daubechies – suporte 20

0.026670057901, 0.188176800078, 0.527201188932, 0.688459039454, 0.281172343661, -0.249846424327,
-0.195946274377, 0.127369340336, 0.093057364604, -0.071394147166, -0.029457536822, 0.033212674059,
0.003606553567, -0.010733175483, 0.001395351747, 0.001992405295, -0.000685856695, -0.000116466855,
0.000093588670, -0.000013264203

6 - Haar

0.7071067, 0.7071067

7 – Symmlet – suporte 8

-0.107148901418, -0.041910965125, 0.703739068656, 1.136658243408, 0.421234534204, -0.140317624179,
-0.017824701442, 0.045570345896

8 - Vaidyanathan – suporte 24

-0.000062906118, 0.000343631905, -0.000453956620, -0.000944897136, 0.002843834547, 0.000708137504,
-0.008839103409, 0.003153847056, 0.019687215010, -0.014853448005, -0.035470398607, 0.038742619293,
0.055892523691, -0.077709750902, -0.083928884366, 0.131971661417, 0.135084227129, -0.194450471766,
-0.263494802488, 0.201612161775, 0.635601059872, 0.572797793211, 0.250184129505, 0.045799334111

APÊNDICE VI

O formato WAV de arquivos de audio

Basicamente um arquivo de audio no formato WAV é composto por um cabeçalho sub dividido em 2 partes chamadas CHUNKS, seguidos pelos dados brutos (CHUNK 1, CHUNK 2, CHUNK de dados). De acordo com a especificação do formato, no caso de amostras com 16 bits, que são os utilizados aqui, a faixa de valores para cada amostra vai de -32768 até 32767, e ainda, os 8 bits finais que compõe a amostra, antecedem os 8 bits iniciais. Além disso, o arquivo utilizado nas experiências do presente trabalho foi gravado em mono canal sem nenhum tipo de compressão. A especificação dos CHUNKS consta abaixo:

CHUNK 1: 12 bytes

Byte	Descrição
0 a 3	String ascii "RIFF"
4 a 7	Comprimento do chunk
8 a 12	String ascii "WAVE"

CHUNK 2: 24 bytes (FMT CHUNK)

Byte	Descrição
0 a 3	String ascii "FMT"
4 a 7	Comprimento do chunk
8 a 9	0 para mono canal e 1 para stereo
10 a 11	Número de canais (1 para mono e 2 para stereo)
12 a 15	Taxa de amostragem em Hz
16 a 19	Bytes por segundo
20 a 21	Bytes por amostra (1 = 8 bits mono, 2 = 8 bits stereo ou 16 bit mono, 4 = 16 bits stereo)
22 a 23	Bits por amostra

CHUNK de DADOS

Byte	Descrição
0 a 3	String ascii "data"
4 a 7	Comprimento do chunk
> = 8	Dados brutos

Tabela AP6-T1: Formato WAV de audio

APÊNDICE VII

O formato BMP de arquivos de imagem

Uma imagem no formato BMP, como a gerada no programa em linguagem C++ que classifica os *spikes* e *overlaps*, consiste basicamente num cabeçalho com 54 bytes seguido pela matriz de cores da imagem. Numa imagem com 24 bits por píxel, que é o tipo utilizado neste trabalho, o cabeçalho é seguido pelos componentes b (*blue* = azul), g (*green* = verde) e r (*red* = vermelho) de cada píxel que compõe a imagem. Além disso, a imagem gerada pelo programa não possui nenhum tipo de compressão e tem largura e altura múltiplas de 4 (160 x 160 píxels), devido a uma particularidade do formato.

A especificação do cabeçalho consta abaixo:

Offset	Tamanho	Descrição
0	2	Assinatura (characters BM ascii)
2	4	Tamanho do arquivo em bytes
6	4	Reservado (valor 0)
10	4	Offset da imagem
14	4	Tamanho do cabeçalho, a partir deste ponto (valor = 40)
18	4	Largura da imagem em píxels
22	4	Altura da imagem em píxels
26	2	Número de planos da imagem (valor = 1)
28	2	Profundidade da cor (bits por píxel)
30	4	Tipo de compressão (valor = 0 para imagem sem compressão)
34	4	Tamanho da imagem comprimida em bytes
38	4	Resolução horizontal
42	4	Resolução vertical
46	4	Número de cores utilizadas
50	4	Número de cores importantes (0 indica que todas as cores são importantes)
54	variável	Matriz de cores (b,g,r)

Tabela AP7-T1: cabeçalho do formato BMP

APÊNDICE VIII

Código Fonte da Implementação *on the fly* do Algoritmo de Detecção e Classificação no DSP EZ-Kit Lite 21065L SHARC - Analog Devices

```
////////////////////////////////////
//spikelet.c////////////////////////////////////
//implementacao on-the-fly do algoritmo de analise do//
//sinal da mosca baseado na spikelet-4////////////////////////////////
//Autor: Rodrigo C. Guido - IFSC - USP - 2003////////////////////////////////
//guido@if.sc.usp.br////////////////////////////////
//Orientador da tese de doutorado: Prof. Dr. Jan Frans//
//Willem Slaets////////////////////////////////
//Colaborador: Lirio Onofre Batista Almeida////////////////////////////////
////////////////////////////////////
#include <21065L.h>
#include <ezkit/1819regs.h>
#include <def21065l.h>
#include <signal.h>
#include <macros.h>
#include <stdio.h>
/*-----
EXTERNAL DECLARATIONS
-----*/

/* These link to the same addresses as in the kernel if the .ldf is set up correctly */
/* Codec control variables */
extern volatile int user_tx_buf(6);
extern volatile int user_tx_ready;
extern volatile int user_rx_buf(6);
extern volatile int user_rx_ready;
/*-----
CONSTANT & MACRO DEFINITIONS
-----*/

/* Codec tag word to send out left and right samples */
#define DOUT_TAG 0x9800
/* Codec tag word to send out address and data slots */
#define REGOUT_TAG 0xe000
/* This is the codec register sspikeleting for line input on both channels */
#define SOURCE_IS_LINE 0x0404
/* This is the codec register sspikeleting for mic input on both channels */
#define SOURCE_IS_MIC 0
/* Codec addreses */
#define SOURCE_ADDR 0x1a00
#define RECGAIN_ADDR 0x1c0c
#define CODEC_ISR_VECT 0x9001
```



```

/*-----
GLOBAL DECLARATIONS
-----*/

int TALK;
void Flag4_On(void);
void Flag4_Off(void);
void Flag5_On(void);
void Flag5_Off(void);
void Flag6_On(void);
void Flag6_Off(void);
void Flag7_On(void);
void Flag7_Off(void);
void Flag8_On(void);
void Flag8_Off(void);
void Flag9_On(void);
void Flag9_Off(void);
void Flags_Off();
void spikelet(double*, int, int);
void coloca_spike_no_canal_direito(int*);
void liga_discriminador();
void desliga_discriminador();

void liga_discriminador() //flag4
{
Flag4_On();
}

void desliga_discriminador()
{
Flag4_Off();
}

void coloca_spike_no_canal_direito(int* am_or)
{
int i;
for(i=0;i<32;i++)
{
user_tx_buf(TAG) = DOUT_TAG;
user_tx_buf(RIGHT_CHNL)=am_or(i);/* put data in output buf */
user_tx_ready = 1;
idle(); /* wait for isr */
while (user_tx_ready); /* wait for isr */
}
}

void transformada_spikelet_nivel_k(double* f, int n, int nivel)
{
double sf()= {
0.26964482896235847376,
0.76237548312490721614,
0.73270322306815560687,
0.23524044702452745481 };

```

```

int csf;
int cwf;
double wf(4);
double t(32);
int i,j,k;

csf=sizeof(sf)/sizeof(double);
for(i=0;i<csf;i++)
{
    wf(i)=sf(csf-i-1);
    if(i%2!=0)
        wf(i)*=-1;
}
cwf=csf;
j=0;

for(i=0;i<n;i+=2) //trend
{
    t(j)=0;
    for(k=0;k<csf;k++)
        t(j)+=f((i+k)%n)*sf(k);
    j++;
}
for(i=0;i<n;i+=2) //fluctuation
{
    t(j)=0;
    for(k=0;k<cwf;k++)
        t(j)+=f((i+k)%n)*wf(k);
    j++;
}
for(i=0;i<n;i++)
    f(i)=t(i);
nivel--;
n/=2;
if(nivel>0)
    transformada_spikelet_nivel_k(&f(0),n,nivel);
}

void Flag4_On(void)
{
    asm("R0 = DM(IOSTAT);\n
    R0 = BCLR R0 BY 0;\n
    DM(IOSTAT) = R0;");
}

void Flag4_Off(void)
{
    asm("R0 = DM(IOSTAT);\n
    R0 = BSET R0 BY 0;\n
    DM(IOSTAT) = R0;");
}

```

```

void Flag5_On(void)
{
    asm("R0 = DM(IOSTAT);\n
    R0 = BCLR R0 BY 1;\n
    DM(IOSTAT) = R0;");
}

```

```

void Flag5_Off(void)
{
    asm("R0 = DM(IOSTAT);\n
    R0 = BSET R0 BY 1;\n
    DM(IOSTAT) = R0;");
}

```

```

void Flag6_On(void)
{
    asm("R0 = DM(IOSTAT);\n
    R0 = BCLR R0 BY 2;\n
    DM(IOSTAT) = R0;");
}

```

```

void Flag6_Off(void)
{
    asm("R0 = DM(IOSTAT);\n
    R0 = BSET R0 BY 2;\n
    DM(IOSTAT) = R0;");
}

```

```

void Flag7_On(void)
{
    asm("R0 = DM(IOSTAT);\n
    R0 = BCLR R0 BY 3;\n
    DM(IOSTAT) = R0;");
}

```

```

void Flag7_Off(void)
{
    asm("R0 = DM(IOSTAT);\n
    R0 = BSET R0 BY 3;\n
    DM(IOSTAT) = R0;");
}

```

```

void Flag8_On(void)
{
    asm("R0 = DM(IOSTAT);\n
    R0 = BCLR R0 BY 4;\n
    DM(IOSTAT) = R0;");
}

```

```

void Flag8_Off(void)
{
    asm("R0 = DM(IOSTAT);\n

```

```

R0 = BSET R0 BY 4;\
DM(IOSTAT) = R0;");
}

void Flag9_On(void)
{
    asm("R0 = DM(IOSTAT);\
R0 = BCLR R0 BY 5;\
DM(IOSTAT) = R0;");
}

void Flag9_Off(void)
{
    asm("R0 = DM(IOSTAT);\
R0 = BSET R0 BY 5;\
DM(IOSTAT) = R0;");
}

void Flags_Off()
{
    Flag4_Off();
    Flag5_Off();
    Flag6_Off();
    Flag7_Off();
    Flag8_Off();
    Flag9_Off();
}

main()
{
    double spikelet(32);
    int tempdata;
    int amostras_originais(32);
    int i; //indice corrente
    int imenor; //indice do menor valor dentre as 32 amostras originais
    int menor; //menor valor dentre as 32 amostras originais
    int amostra_de_maior_declividade;
    int momento_de_ligar_discriminador;
    int spike_ou_overlap_detectado=0;
    asm("#include <def21065l.h>");
    interrupt(SIG_SPT11,(void (*)(int))CODEC_ISR_VECT);
    asm("BIT SET IMASK SPT11;"); /* unmask sport interrupt */

    /* Set input source in codec as line */
    user_tx_buf(TAG) = REGOUT_TAG;
    user_tx_buf(ADDR) = SOURCE_ADDR;
    user_tx_buf(DATA) = SOURCE_IS_LINE;
    user_tx_ready = 1;
    idle();
    idle();

    /* Set record gain in codec to 12 */

```

```

user_tx_buf(TAG) = REGOUT_TAG;
user_tx_buf(ADDR) = RECGAIN_ADDR;
user_tx_buf(DATA) = 0;
user_tx_ready = 1;
idle();
idle();
user_rx_ready = 1; /* Get first data from codec */
idle();          /* wait for data */

for(i=0;i<32;i++)
    amostras_originais(i)=0;

Flags_Off();
while(1)
{
    while (user_rx_ready);
    tempdata = (user_rx_buf(RIGHT_CHNL)+ user_rx_buf(LEFT_CHNL))/2; /* Get codec data */
    //////////////////////////////////////
    //discriminador
    if((momento_de_ligar_discriminador>0)&&(spike_ou_overlap_detectado))
        momento_de_ligar_discriminador--;
    else if((momento_de_ligar_discriminador==0)&&(spike_ou_overlap_detectado))
    {
        liga_discriminador();
        momento_de_ligar_discriminador--; //vai para -1
    }
    else if ((momento_de_ligar_discriminador== -1)&&(spike_ou_overlap_detectado))
    {
        desliga_discriminador();
        spike_ou_overlap_detectado=0;
    }
    //////////////////////////////////////
    for(i=1;i<32;i++)
        amostras_originais(i-1)=amostras_originais(i);
    amostras_originais(31)=tempdata;
    if((amostras_originais(13)<-6600)&&(amostras_originais(13)<amostras_originais(12))&&(amostras_originais(13)<amostras_originais(14)))
    {
        //////////////////////////////////////
        //devido a classificação do formato ser feita com base no pico, ele deve sempre estar na amostra 13.
        imenor=13;
        menor=amostras_originais(13);
        for(j=14;j<32;j++)
            if(amostras_originais(j)<menor)
            {
                menor=amostras_originais(j);
                imenor=j;
            }
        if(imenor!=13) //pico tinha sido pego erroneamente...
        for(i=0;i<(imenor-13);i++) //rotacionar o sinal de 32 amostras, o numero de vezes necessario para deixar o pico
            // de fato na amostra 13
            {
                for(i=1;i<32;i++)

```

```

amostras_originais(i-1)=amostras_originais(i);
while (user_rx_ready);
tempdata = (user_rx_buf(RIGHT_CHNL)+ user_rx_buf(LEFT_CHNL))/2; /* Get codec data */
amostras_originais(31)=tempdata;
//se amostrou, tem que colocar na saída left para que ela nao
// contenha amostras puladas.....
////////////////////////
user_tx_buf(TAG) = DOUT_TAG;
user_tx_buf(LEFT_CHNL) = tempdata; /* put data in output buf */
user_rx_ready = 1; /* tell isr that we're ready */
user_tx_ready = 1;
idle(); /* wait for isr */
while (user_tx_ready); /* wait for isr */
}

////////////////////////////////////
//encontrando o momento em que o spike/overlap ocorreu
//a variavel momento_de_ligar_discriminador contem o numero da amostra, apos o inicio
//do spike/overlap, onde ocorre o ponto de > declividade ("instante"
//onde ocorreu o spike/overlap...)
momento_de_ligar_discriminador=0;
amostra_de_maior_declividade=abs(amostras_originais(0)-amostras_originais(1));
for(i=1;i<13;i++)
{
if (abs(amostras_originais(i) - amostras_originais(i+1) )> amostra_de_maior_declividade)
{
amostra_de_maior_declividade=abs(amostras_originais(i) - amostras_originais(i+1));
momento_de_ligar_discriminador=i;
}
}

////////////////////////////////////
//classificando o formato do spike/overlap
for(i=0;i<32;i++)

spikelet(i)=(double)amostras_originais(i);
transformada_spikelet_nivel_k(&spikelet(0),32,3);
Flags_Off();
if((spikelet(1)*spikelet(2)<0)&&(spikelet(4)*spikelet(5)<0)&&(spikelet(5)*spikelet(6)>0)&&(spikelet(6)*spikelet(7)>0))
{
//overlap acentuado a esquerda
Flag5_On();
coloca_spike_no_canal_direito(&amostras_originais(0));
spike_ou_overlap_detectado=1;
}
else if ((spikelet(1)*spikelet(2)<0)&&(spikelet(4)*spikelet(5)<0)&&(spikelet(5)*spikelet(6)<0)&&(spikelet(6)*spikelet(7)>0))
{
//overlap a esquerda
Flag6_On();
coloca_spike_no_canal_direito(&amostras_originais(0));
spike_ou_overlap_detectado=1;
}
else if ((spikelet(1)*spikelet(2)<0)&&(spikelet(4)*spikelet(5)<0)&&(spikelet(5)*spikelet(6)<0)&&(spikelet(6)*spikelet(7)<0))
{
//spike
Flag5_On();
Flag6_On();
coloca_spike_no_canal_direito(&amostras_originais(0));
}

```

```

        spike_ou_overlap_detectado=1;
    }
else if ((spikelet(1)*spikelet(2)<0)&&(spikelet(4)*spikelet(5)>0)&&(spikelet(5)*spikelet(6)<0)&&(spikelet(6)*spikelet(7)<0))
    {
        //overlap a direita
        Flag7_On();
        coloca_spike_no_canal_direito(&amostras_originais(0));
        spike_ou_overlap_detectado=1;
    }
else if ((spikelet(1)*spikelet(2)<0)&&(spikelet(4)*spikelet(5)>0)&&(spikelet(5)*spikelet(6)>0)&&(spikelet(6)*spikelet(7)<0))
    {
        //overlap acentuado a direita
        Flag5_On();
        Flag7_On();
        coloca_spike_no_canal_direito(&amostras_originais(0));
        spike_ou_overlap_detectado=1;
    }
else if ((spikelet(1)*spikelet(2)<0)&&(spikelet(4)*spikelet(5)<0)&&(spikelet(5)*spikelet(6)>0)&&(spikelet(6)*spikelet(7)<0))
    {
        //overlap a esquerda e a direita
        Flag7_On();
        Flag6_On();
        coloca_spike_no_canal_direito(&amostras_originais(0));
        spike_ou_overlap_detectado=1;
    }
else if ((spikelet(1)*spikelet(2)>0)&&((spikelet(0)*spikelet(1)<0) || (spikelet(2)*spikelet(3)<0)))
    {
        //spike anormal
        Flag7_On();
        Flag6_On();
        Flag5_On();
        coloca_spike_no_canal_direito(&amostras_originais(0));
        spike_ou_overlap_detectado=1;
    }

    //////////////////////////////////////
    //coloca o sinal original na saída left
}

user_tx_buf(TAG) = DOUT_TAG;
user_tx_buf(LEFT_CHANL) = tempdata; /* put data in output buf */
user_rx_ready = 1; /* tell isr that we're ready */
user_tx_ready = 1;
idle(); /* wait for isr */
while (user_tx_ready); /* wait for isr */
}
}

////////////////////////////////////
//spikelet.ldf////////////////////////////////////
//implementacao on-the-fly do algoritmo de analise do//
//sinal da mosca baseado na spikelet-4////////////////////////////////////
//Autor: Rodrigo C. Guido - IFSC - USP – 2003////////////////////////////////////
//guido@ifsc.usp.br////////////////////////////////////
//Orientador da tese de doutorado: Prof. Dr. Jan Frans//
//Willem Slaets////////////////////////////////////
//Colaborador: Lirio Onofre Batista Almeida////////////////////////////////////
////////////////////////////////////

```



```

seg_rth
{
    INPUT_SECTIONS( $OBJECTS(seg_rth) $LIBRARIES(seg_rth))
} >seg_rth

seg_init
{
    INPUT_SECTIONS( $OBJECTS(seg_init) $LIBRARIES(seg_init))
} >seg_init

seg_pmco
{
    INPUT_SECTIONS( $OBJECTS(seg_pmco) $LIBRARIES(seg_pmco))
} >seg_pmco

seg_pmda
{
    INPUT_SECTIONS( $OBJECTS(seg_pmda) $LIBRARIES(seg_pmda))
} >seg_pmda

seg_bnk3
{
    INPUT_SECTIONS( $OBJECTS(seg_bnk3) $LIBRARIES(seg_bnk3))
} >seg_bnk3

seg_dmda
{
    INPUT_SECTIONS( $OBJECTS(seg_dmda) $LIBRARIES(seg_dmda))
} > seg_dmda

stackseg
{
    // allocate a stack for the application
    ldf_stack_space = .;
    ldf_stack_length = MEMORY_SIZEOF(seg_stak);
} > seg_stak

heap
{
    // allocate a heap for the application
    ldf_heap_space = .;
    ldf_heap_end = ldf_heap_space + MEMORY_SIZEOF(seg_heap) - 1;
    ldf_heap_length = ldf_heap_end - ldf_heap_space;
} > seg_heap
}
}

```

```

////////////////////////////////////
//buffers.asm////////////////////////////////////
//implementacao on-the-fly do algoritmo de analise do//
//sinal da mosca baseado na spikelet-4////////////////////////////////////
//Autor: Rodrigo C. Guido - IFSC - USP - 2003////////////////////////////////////
//guido@ifsc.usp.br////////////////////////////////////
//Orientador da tese de doutorado: Prof. Dr. Jan Frans///
//Willem Slaets////////////////////////////////////
//Colaborador: Lirio Onofre Batista Almeida////////////////////////////////////
////////////////////////////////////

```

```

.GLOBAL _user_tx_buf;
.GLOBAL _user_tx_ready;
.GLOBAL _user_rx_buf;
.GLOBAL _user_rx_ready;
.GLOBAL _user_data_out_ptr;
.GLOBAL _user_num_data;
.GLOBAL _user_data_type;

.SEGMENT/DM    seg_bnk3;
! make the buffers line up the same as in the kernel
.var _user_tx_buf(6);
.var _user_tx_ready;
.var _user_rx_buf(6);
.var _user_rx_ready;
.var _user_data_out_ptr;    !pointer to data to send to the host
.var _user_num_data;        !says how many data items there are
.var _user_data_type;       ! 0 = ascii, 1 = decimal, 2 = hex

.ENDSEG;

////////////////////////////////////
//065L_hdr.asm////////////////////////////////////
//implementacao on-the-fly do algoritmo de analise do/////
//sinal da mosca baseado na spikelet-4////////////////////////////////
//Autor: Rodrigo C. Guido - IFSC - USP - 2003////////////////////////////////
//guido@if.sc.usp.br////////////////////////////////
//Orientador da tese de doutorado: Prof. Dr. Jan Frans////////
//Willem Slaets////////////////////////////////
//Colaborador: Lirio Onofre Batista Almeida////////////////////////////////
////////////////////////////////////
#define KERNEL_UART_ISR 0x9000
#define KERNEL_CODEC_ISR 0x9001
#define KERNEL_SWI3_ISR 0x9002
#define __ADSP21065L__
#undef __ADSP21020__
#include <ezkit/sig_glob.h>
#define INT(irp) \
    BIT CLR MODE1 0x1000;           /*Disable interrupts          */ \
    JUMP __z3_int_determiner(DB);    /*jmp to finish sspikeleting up */ \
    DM(I7,M7)=R0;                   /*Save r0 (scratch dreg)       */ \
    R0=SIGMASK(SIG_##irp);          /*Base of int table            */
#define RESERVED_INTERRUPT NOP;NOP;NOP;NOP

.GLOBAL      __lib_prog_term;        /* Termination address          */
.GLOBAL      __done_execution;
.EXTERN      __lib_setup_c;
.SEGMENT/PM  seg_rth;                /* Runtime header segment      */
RESERVED_INTERRUPT;

__lib_RSTl:      NOP;                /* Not really executed          */
                JUMP __lib_start;
                NOP;

```

```

                                NOP;

#ifdef __ADSP21160__
__lib_IICDI:      INT(IICDI); /* Access to illegal IOP space */
#else

                                RESERVED_INTERRUPT;

#endif

__lib_SOVI:      INT(SOVI); /* status/loop/PC stack overflow */
__lib_TMZHI:     INT(TMZ0); /* high priority timer */
__lib_VIRPTI:    INT(VIRPTI); /* external interrupts */
__lib_IRQ2I:     INT(IRQ2);
__lib_IRQ1I:     INT(IRQ1);

// IRQ0
// This vector is used for the UART.
// It is wrispikeleten during power-on and protected from over-writing
// by rewriting it after writing data to memory (so this declaration
// is redundant, but a good reminder).
// Over-writing this vector will stop the kernel.
// The user may over-write it from his/her code - but will loose
// the capability to communicate over the RS-232 interface.
// IRQ0, IRPTL bit 8, Vector 0x20
__lib_IRQ0I:
    JUMP KERNEL_UART_ISR;      RTI;      NOP;      RTI;

                                RESERVED_INTERRUPT;

__lib_SPR0I:     INT(SPR0I); /* serial port DMA channel interrupts */
__lib_SPR1I:     INT(SPR1I);
__lib_SPT0I:     INT(SPT0I);

// If you wish to use the kernel's codec services, do not write
// over the following vector
__lib_SPT1I:
    JUMP KERNEL_CODEC_ISR;    RTI;      RTI;      RTI;

#ifdef __ADSP21160__
#ifdef __ADSP21065L__

                                RESERVED_INTERRUPT;
                                RESERVED_INTERRUPT;

#else
__lib_LP2I:      INT(LP2I); /* link port DMA channel interrupts */
__lib_LP3I:      INT(LP3I);
#endif
#else
__lib_LP0I:      INT(LP0I); /* link port DMA channel 4 */
__lib_LP1I:      INT(LP1I); /* link port DMA channel 5 */
__lib_LP2I:      INT(LP2I); /* link port DMA channel interrupts */
__lib_LP3I:      INT(LP3I);
__lib_LP4I:      INT(LP4I); /* link port DMA channel 8 */
__lib_LP5I:      INT(LP5I); /* link port DMA channel 9 */
#endif
__lib_EP0I:      INT(EP0I); /* ext port DMA channel interrupts */

```

```

__lib_EP1I:                INT(EP1I);
#ifdef __ADSP21065L__
__lib_EP2I:                INT(EP2I);
__lib_EP3I:                INT(EP3I);
__lib_LSRQ:                INT(LSRQ);                /* link service request */
#else
RESERVED_INTERRUPT;
RESERVED_INTERRUPT;
RESERVED_INTERRUPT;

#endif

__lib_CB7I:                INT(CB7);                /* circular buffer #7 overflow */
__lib_CB15I:               INT(CB15);               /* circular buffer #15 overflow */
__lib_TMZLI:               INT(TMZ);               /* low priority timer */
__lib_FIXI:                INT(FIX);               /* fixed point overflow */
__lib_FLTOI:               INT(FLTO);              /* floating point overflow */
__lib_FLTUI:               INT(FLTU);              /* floating point underflow */
__lib_FLTI:                INT(FLT);              /* floating point invalid */
__lib_SFT0I:               INT(USR0);              /* user interrupts 0..3 */
__lib_SFT1I:               INT(USR1);
__lib_SFT2I:               INT(USR2);

// If you wish to use the kernel's host output services, do not write
// over the following vector
__lib_SFT3I:
JUMP KERNEL_SWI3_ISR;      RTI;      RTI;      RTI;

__z3_int_determiner:
DM(I7,M7)=R1;
R1=I2;
DM(I7,M7)=R1;                /* Save I2 (scratch reg) */
I2=R0;
DM(I7,M7)=I13;              /* Save I13 (scratch reg) */
I13=DM(5,I2);               /* get disp to jump to */
JUMP (M13, I13) (DB);        /* Jump to dispatch */
BIT SET MODE2 0x80000;       /* Freeze cache */
I13=DM(2,I2);               /* rd handler addr (base+2) */

/* Note: It's okay to use PM in gspikeleting the above values b'cse z3 has a */
/* linear memory. Therefore dm and pm are the same and we can use either.*/

__lib_start:
CALL __lib_setup_c;          /* Setup C runtime model */
#ifdef MAIN_RTS
CJUMP _main (DB);            /* Begin C program */
DM(I7,M7)=R2;
DM(I7,M7)=PC;
#else
JUMP _main;                  /* Begin C program */
#endif

/* Sespikeleting the __done_execution flag indicates that this processor is */
/* finished executing, for the benefit of anyone who may be watching. */

```

```
__lib_prog_term:    PM(__done_execution)=PC;
                   IDLE;
                   JUMP __lib_prog_term;          /* Stay put          */
.VAR __done_execution = 0;

.ENDSEG;
```

Apêndice IX

Trabalhos Publicados e Apresentados

Publicações em Periódicos:

GUIDO,R.C.; SLAETS, J. F. W.; ALMEIDA, L.O.B. "Transformadas *Wavelet* e Fourier no Processamento Digital de Sinais: Uma Introdução". Revista UNORP: Computação, SJRPeto: Editora do Centro Universitário do Norte Paulista,v.5, p.9-17 ,Jun-2003. ISSN:1678-1902.

GUIDO,R.C.; SLAETS, J. F. W.; ALMEIDA, L.O.B. "Processamento Digital de Sinais: Wavelets X Fourier". Revista Saber Eletrônica, São Paulo: Saber, n. 361, p.66-70, Fev-2003. ISSN: 0101-6717.

Apresentação de Trabalho em Congresso:

GUIDO,R.C. "Criação de uma Nova Transformada *Wavelet* Aplicada ao Processamento Digital de Sinais Neurofisiológicos do Campo Visual da Mosca". In: XXVI Congresso Nacional de Matemática Aplicada e Computacional da Sociedade Brasileira de Matemática Aplicada e Computacional. SJRioPreto-SP, Set- 2003.

Publicação em Periódico Internacional Submetida:

GUIDO,R.C.; SLAETS, J. F. W.; ALMEIDA, L.O.B. "A New Technique to construct a wavelet transform to match a specified signal, with applications to real-time spike and overlap pattern recognition". Digital Signal Processing Letters. Elsevier Science Publishing Co., Nov-2003.

Artigo completo em Seleta de Congresso Submetido:

GUIDO,R.C.; SLAETS, J. F. W.; ALMEIDA, L.O.B. "Criação de uma Nova Transformada *Wavelet* Aplicada ao Processamento Digital de Sinais Neurofisiológicos do Campo Visual da Mosca". Seleta do XXVI Congresso Nacional de Matemática Aplicada e Computacional. Rio de Janeiro: Editora da SBMAC, Ago-2003.

REFERÊNCIAS

[ALG1]: CHAPA, J.O.; RAO, R.M. "Algoritms for Designing Wavelets to Match a Specified Signal". In: IEEE Transactions on Signal Processing. V 48. N 12. USA, Dez-2000.

[ALG2]: LIPSCHUTZ, S. *Álgebra Linear*. 3.ed. São Paulo: Makron Books do Brasil, 1994.

[ANA1]: FRANCO, A.S. *Análise Espectral Contínua e Discreta*. 1. ed. São Paulo: IPT - Instituto de Pesquisas Tecnológicas do Estado de São Paulo, 1982.

[BAT1]: TENTZERIS, E.M.; ROBERTSON, R.L. HARVEY, J.F. KATEHI, P.B. "Stability and Dispersion Analysis of Battle-Lemarie-Based MRTD Schemes". In: IEEE Transactions on Microwave Theory and Techniques. V.47 n.7, p.1004-1013, Jul.1999.

[BEY1]: BEYLKIN: "Wavelets and Fast Numerical Algorithms". In: Different Perspectives on Wavelets (edited by Ingrid Daubechies). EUA: American Mathematical Society, 1995.

[BMP1]: MARQUES, O. F.; NETO, H. V. *Processamento Digital de Imagens*. Rio de Janeiro: Brasport, 1999.

[CAL1]: CLAUDIO, D. M.; MARINS, J. M. *Cálculo Numérico Computacional: Teoria e Prática*. 3. ed. São Paulo: Atlas, 2000.

[COI1]: WEI, D.; BOVIK, A.; EVANS, B. "Generalized Coiflets: A New Family of Orthonormal Wavelets". Laboratory of Image and Video Compression – Department of Electrical and Computer Engineering – University of Texas: Austin – Texas, 1999.

[COM1]: HWANG, K.; BRIGGS, F. A. *Computer Architecture and Parallel*

Processing. New York-USA: McGraw-Hill International Editions, 1984.

[CRO1]: REBRIK, S. P.; WRIGHT, B. D.; EMONDI, A. A.; MILLER, K. D. "Cross channel correlations in tetrode recordings: implications for spike sorting". USA, 2002. Disponível na Internet em <http://mccoy.ucsf.edu/Papers/Cns98/CNS98-paper.html>

[DAU1]: DAUBECHIES, I. *Ten Lectures on Wavelets*. EUA: Society for Industrial & Applied Mathematics, 1992.

[DIG1]: SCHLICHTHÄRLE, D. *Digital Filters: Basic and Design*. Germany: Springer-Verlag, 2000.

[DIP1]: *Homepage* do Laboratório Dipterlab – IFSC – USP disponível na Internet em <http://www.dipteralab.if.sc.usp.br> . Revisto em 07/2003.

[DSP1]: LANE, J; DATTA, J; KARLEY, B.; NORWOOD, J. *DSP Filters*. Indianapolis – EUA: Prompt Publications, 2001.

[DSP2]: *Homepage* da Analog Devices na Internet com link para o manual: http://www.analog.com/Analog_Root/static/library/dspManuals/32BitIndex.html. Revisto em 07/2003.

[EQU1]: BOYCE, W.E.; DI PRIMA, R.C. *Equações Diferenciais Ordinárias e Problemas de Valores de Contorno*. 5. ed. Rio de Janeiro: Guanabara Koogan, 1994.

[EST1]: GUPTA, A; JOSHI, S.D.; PRASAD, S. "On a New Approach for Estimating Wavelet Matched to Signal". Proceedings Eighth National Conference on Communications, IIT Bombay, Jan. 25-27, p.180-184, 2002.

[EST2]: TENENBAUM, A.; LANGSAM, Y.; AUGENSTIEN, M. *Estruturas de Dados Usando C*. São Paulo: Makron Books do Brasil, 1995.

[FIL1]: HAMMING, R.W. *Digital Filters*. 3. ed. New York: Dover Publications Inc, 1989.

[FIL2]: VETTERLI, M. *Wavelets and Filter Banks: Theory and Design*. In: IEEE Transactions on Signal Processing. Vol 40. Nr. 9.USA, set-1992.

[FIL3]: PETERSON, S. *Filtering and Wavelet Regression Methods with Application to Exercice ECG*. Sweden: Lund University, 1996. (Tese de Doutorado).

[HAA1]: DEVORE, R.A.; LUCIER, B.J. "Wavelets". In: Acta Numerica. n.92 p.1-56. Cambridge University Press. New York: A. Iserles, 1992.

[HAN1]: ADDISON, P. S. *The Illustrated Wavelet Transform Handbook: Introductory Theory and Applications in Science, Engineering, Medicine and Finance*. Edinburg-UK: Institute of Physics Publishing, 2002.

[HAU1]: HAUSEN, K. "Motion sensitive interneurons in the optomotor system of the fly. The horizontal cells: receptive field organization and response characteristics". Biological Cybernetics, Berlin, n.46, p.67-79, 1982.

[INT1]: WILLIAMS, J. R.; AMARATUNGA, K. "Introduction to Wavelets in Engineering". In: International Journal for Numerical Methods in Engineering, V.37 pp. 2365-2388. Cambridge-USA, 1994.

[JAV1]: JAIN, A.K.; MURTY, M.N.; FLYNN, P.J. "Data Clustering: a review". ACM Computing Surveys, 31(3), p. 264-323, 1999.

[LIN1]: NEGUS, C. *Red Hat Linux 8.0 Bible*. Indianapolis-EUA: John Wiley & Sons Incorporated, 2002.

[MIC1]: SOTO, H.; MANJARREZ, E.; VEGA, R. A microcomputer program for automated neuronal spike detection and analysis. In: International Journal of

Medical Informatics. V.44 pp.203-212. USA, 1997.

[OCT1]: EATON, J. *GNU Octave Manual*. Boston-USA: Network Theory, 2002.

[OPE1]: OPPENHEIM, A.V.; SCHAFER, R.W. *Discrete Time Signal Processing*. 2. ed. New York: Prentice Hall, 1999.

[ORI1]: *Homepage* do fabricante na Internet: <http://www.originlab.com>.
Revisto em 07/2003.

[PAC1]: TRIMECHE, K. "Generalized Harmonic Analysis and Wavelet Packets: An Elementary Treatment of Theory and Applications". London – UK: Taylor & Francis, 2001.

[PDS1]: BATEMAN, A.; PATERSON-STEPHENS, L. *The DSP Handbook: Algorithms, Applications and Design Techniques*. New York: Prentice-Hall, 2002.

[PRI1]: WALKER, J. S. *A Primer on Wavelets and Their Scientific Applications*. Whashington-USA: Chapman & Hall/CRC, 1999.

[PRO1]: GONZALEZ, R.; WOODS, R. *Processamento de Imagens Digitais*. São Paulo: Edgard Bluscher, 2000.

[RIP1]: JENSEN, A.; COUR-HARBO, A. *Ripples in Mathematics: The Discrete Wavelet Transform*. New York-USA: Springer-Verlag, 2000.

[REP1]: STEVENINCK, R. R. R. V.; LEWEN, G. D.; STRONG, S. P.; KOBERLE, R.; BIALEK, W. "Reproducibility and Variability in Neural Spike Trains". In: *Science*. V. 275. USA, mar 1997.

[REV1]: LEWICKI, M. S. "A review of methods for spike sorting: the detection and classification of neural action potentials". In: *Network Computing Neural Systems*. R.53-78. UK, 1998.

[SIG1]: DEVASAHAYAM, S. R. *Signals and Systems in Biomedical Engineering: Signal Processing and Physiological System Modeling*. New York: Kluwer Academic Publishers, 2000.

[SIN1]: HAYKIN, S.; VEEN, B.V. *Sinais e Sistemas*. 1. ed. Porto Alegre: Bookman, 2001.

[SMD1]: DAVIDSON, H.L.; HEUSEL, K. *SMD Electronics Projects*. New York-USA: Delmar Learning, 2000.

[SPK1]: RINBERG, D.; BIALEK, W.; DAVIDOWITZ, H.; TISHBY, N. "Spike sorting in the frequency domain with overlap detection". NEC Research Institute. Princeton-NJ, Jun-2003

[SYM1]: ZHANG, X.; YOSHIKAWA, T. "A New Class of Complex Compact-Supported Orthonormal Symmlets". In: IEICE Transactions Fundamentals. V.E84-A, n.7, Jul.2001.

[TES1]: MENNE, K. M. L.; FOLKERS, A. MA, T. MAEX, R., HOFMANN, U. G. "Test of spike sorting algorithms on the basis of simulated network data". In: Neurocomputing, 2001.

[TIM1]: AKAY, M. *Time Frequency and Wavelets in Biomedical Signal Processing*. New York: IEEE Press in Biomedical Engineering, 1998.

[VAI1]: GONON, G.; MONTRESOR, S.; BAUDRY, M. "Extended Wavelet Packet Decomposition and Best Basis Search Algorithm, Evaluation of Entropic Gain for Audio Signals". Le Mans – France: Laboratoire d'Acoustique, Université du Maine, 2000.

[WAV1]: POLIKAR, R. *The Wavelet Tutorial: Fundamental Concepts and an overview of the Wavelet theory*. 2.ed. New Jersey: Rowan University, 2001.

Disponível na Internet em <http://engineering.rowan.edu/~polikar/WAVELETS/WTtutorial.html> . Revisto em 07/2003.

[WAV2]: STRANG, G.; NGUYEN, T. *Wavelets and Filter Banks*. Wellesley-MA-USA: Wellesley-Cambridge Press, 1997.

[WAV3]: VETTERLI, M. *Wavelets and Filter Banks*. In: IEEE Transactions on Signal Processing. V.40 n.9. USA, set. 1992.

[WAV4]: WILLIAMS, J.R.; AMARATUNGA, K. "Introduction to Wavelets in Engineering". Intelligent Engineering System Laboratory. In: International Journal for Numerical Methods in Engineering. Vol 37. pp 2365-2388. Cambridge-USA: Massachusetts Institute of Technology , 1994

[WAV5]: BORN, G. *File Formats Handbook*. Crystal City, VA-USA: Van Nostrand Reinhold, 1995.

[WAV6]: STRANG, G. "Wavelets and Dilation Equations: a brief introduction". In: SIAM Rev.,31(4), 614-627, 1989.

[WAV7]: KRIM, H.; TUCKER, D.; MALLAT, S.; DONOHO, D. "On Denoising and Best Signal Representation". In: IEEE Transactions on Information Teory. V. 45 N. 7, nov. de 1999.

[WIN1]: SPALDING, G. *Windows 2000 Administration*. New York-USA: McGraw-Hill International, 2002.