UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CURSO DE ENGENHARIA DE COMPUTAÇÃO

ANDRÉIA ALENCAR CARVALHO DA SILVA

COMPRESSÃO DE SINAIS DE ELETROCARDIOGRAMAS UTILIZANDO DSP

TRABALHO DE CONCLUSÃO DE CURSO

CORNÉLIO PROCÓPIO

2015

ANDRÉIA ALENCAR CARVALHO DA SILVA

COMPRESSÃO DE SINAIS DE ELETROCARDIOGRAMAS UTILIZANDO DSP

Trabalho de conclusão de curso apresentada à disciplina de Trabalho de Conclusão de Curso 2 da Universidade Tecnológica Federal do Paraná como requisito parcial para obtenção do grau de Bacharel em Engenharia de Computação

Orientador: Prof. Dr. Cristiano Marcos Agulhari

CORNÉLIO PROCÓPIO

2015

Dedico este trabalho à minha família que sempre me apoiou e acreditou no meu potencial para chegar até aqui.

AGRADECIMENTOS

Primeiramente agradeço à Deus porque mesmo eu não merencendo, Ele permitiu que eu chegasse até aqui, me sustentando e me dando forças para superar cada dificuldade.

A esta universidade, direção e administração que oportunizaram a janela que hoje vislumbro um horizonte superior.

A todo o corpo docente por me proporcionar todo o conhecimento necessário, no processo de formação profissional, meus eternos agradecimentos não somente por terem me ensinado, mas sobretudo por terem me feito aprender. A palavra mestre nunca fará justiça aos professores dedicados aos quais sem nominar terão os meus eternos agradecimentos.

Em especial agradeço ao meu orientador, Prof. Dr. Cristiano Marcos Agulhari, por exercer com excelência o papel de orientador, sempre pronto a me auxiliar, disponibilizando seu tempo, e me atendendo com uma simpatia contagiante e por sua paciência na orientação além do grande incentivo que tornaram possível a conclusão deste trabalho.

Aos meus pais, Humberto e Gislene, por todo esforço, incentivo, dedicação e amor incondicional que nunca me deixaram desistir e sempre acreditaram no meu potencial, quando nem eu mesma acreditava.

Aos meus irmãos Murilo e Rafael e minha cunhada Geisa por estarem sempre presentes e a minha sobrinha Isabela, por alegrar os meus dias com seu sorriso mais puro e sincero.

Aos meus amigos que acompanharam minha trajetória desde o inicio da minha vida universitária, e aos que foram se achegando durante esses cinco anos, agradeço por sempre me apoiarem, acreditarem em mim, e principalmente, por alegrarem os meus dias mais dificeis.

Enfim, a todos que direta ou indiretamente fizeram parte da minha formação, o meu muito obrigada.

Entrega o teu caminho ao Senhor; confia nele, e ele tudo fará.

Salmos 37:5

RESUMO

ALENCAR, Andréia. COMPRESSÃO DE SINAIS DE ELETROCARDIOGRAMAS UTILIZANDO DSP. 42 f. Trabalho de conclusão de curso – Curso de Engenharia de Computação, Universidade Tecnológica Federal do Paraná. Cornélio Procópio, 2015.

Um algoritmo de compressão de dados de eletrocardiogramas(ECGs), baseada em wavelet e banco de filtros é proposto neste trabalho. O objetivo é comprimir os dados gerados por ECGs, de forma a ocupar pouco espaço e ser transmitido mais rapidamente. O algoritmo será implementado no DSP e os testes serão realizados com sinais de ECGs presentes na base de dados MIT-BIH.

Palavras-chave: Compressão, Wavelets, Banco de filtros, Eletrocardiogramas

ABSTRACT

ALENCAR, Andréia. ELECTROCARDIOGRAMS SIGNALS COMPRESSION USING DSP. 42 f. Trabalho de conclusão de curso – Curso de Engenharia de Computação, Universidade Tecnológica Federal do Paraná. Cornélio Procópio, 2015.

An algorithm for the compression of electrocardiogram (ECG) data, based on wavelet transforms implemented through a filter bank, is proposed in this work. The main objective is to compress the data generated by ECG signals, in a way to reduce the disk storage space needed and improve transmission speed. The algorithm will be implemented on a DSP and a series of tests will be conducted with ECG signals of a MIT-BIH database.

Keywords: Compression, Wavelets, Filter bank, Electrocardiogram

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

ECG	Eletrocardiograma
DSP	*Digital Signal Processor* - Processador digital de sinais
JPEG	*Joint Photographic Experts Group*
KB	*Kilobyte*
PRD	*Percent Root-mean-square Distortion*
CR	*Compression Ratio*
DCT	*Discrete Cosine Transform*
SPIHT	*Set Partitioning in Hierarchical Trees*
IDE	*Integrated Development Environment*
CCS	*Code Composer Studio v4*
USB	*Universal Serial Bus*

SUMÁRIO

1 INTRODUÇÃO

Eletrocardiogramas (ECG) são sinais de origem biológica, gerados a partir do registro da atividade elétrica do coração, que são utilizados em exames clínicos para análise da saúde cardíaca em um paciente. Um ECG gera uma quantidade de dados muito grande, devido à quantidade de eletrodos captando os sinais, e também devido à duração de alguns exames, como a monitoração Holter (JANE et al., 1997) que dura 24 horas. Com um volume tão grande de dados, é necessário tratá-los para seu devido armazenamento ou transmissão, por isso a necessidade da sua compressão (DENG; POOLE, 2003).

O processo de diminuição de tamanho de dados é chamado de compressão, e é definido como o ato de reduzir o espaço ocupado em um determinado dispositivo (ABENSTEIN; TOMPKINS, 1982). Em (LU et al., 2000), os autores usaram como base o algoritmo de compressão SPIHT, que é um algoritmo de compressão de imagens. Esse algoritmo foi modificado para se adequar as características dos sinais de ECGs, e foi testado em vários registros no banco de dados de arritmia MIT-BIH (PHYSIOBANK, 2015). Os resultados mostraram que o algoritmo comprime todos os tipos de dados de ECG de forma muito eficiente e que a codificação e decodificação são fáceis e rápidas de implementar.

Existem vários métodos para se comprimir dados, e eles podem ser divididos em duas categorias, que são os métodos diretos e os métodos de transformadas. Um exemplo de transformada que pode ser utilizado nesse processo é a transformada *wavelet* (LU et al., 2000). Em (RAJOUB, 2002), o autor desenvolveu um algoritmo de compressão baseado em *wavelets*, e testaram em registros no banco de dados de arritmia MIT-BIH (PHYSIOBANK, 2015). Os testes mostraram que o algoritmo comprime todos os tipos de ECGs com índices médios de compressão do PRD e médias que são muito melhores do que os outros algoritmos utilizados para comparação, e a codificação e decodificação são fáceis e rápidas de implementar. Em (MANIKANDAN; DANDAPAT, 2007), os autores apresentam um algoritmo de compressão baseado em *wavelet* para uma eficaz compressão de sinais de ECG ruidosos com base no algoritmo SPIHT. Os resultados obtidos mostraram que o ruído diminui a taxa de compressão para um determinado nível de distorção.

E em (HAO; MARZILIANO, 2004), os autores desenvolveram um algoritmo de compressão de dados de ECG híbrido. O algoritmo comprime e reconstitui os dados. Para compressão, alguns coeficientes significativos da tranformada *wavelet*, que representam os batimentos normalizados, são selecionados para a transmissão. E para reconstituição, as batidas normalizadas são reconstruídas a partir de tranformada *wavelet* inversa, e o sinal original é recuperado. O método funciona bem com dados normais, e alcança um taxa de compressão de alto desempenho com baixa distorção, e características morfológicas são bem preservadas no sinal reconstruído. Em comparação com alguns métodos recentes, a técnica é muito eficiente.

Wavelets são funções de pequena largura, que formam uma base de representação com certas características especiais, e podem ser vistas como mecanismos para decompor ou quebrar sinais nas suas partes constituintes (LIMA, 2002). Essas ferramentas podem ser implementadas através de algoritmos computacionais, com destaque em vários domínios como a engenharia, a física, a matemática entre outros (GRAPS, 1995). Uma das aplicações em que as *wavelets* podem ser utilizadas é o processo de compressão de sinais.

Atualmente, tem-se dado um destaque cada vez maior à telemedicina, que trata-se do conjunto de tecnologias e aplicações que permitem a realização de ações médicas a distância. É necessário, portanto, realizar compressões eficazes para uma transmissão mais rápida, porém, sem perder as informações necessárias para um diagnóstico correto.

Neste trabalho, é proposta a utilização de transformadas *wavelet* para comprimir sinais de ECG. O sinal de ECG será processado em um banco de filtros, responsável por gerar os coeficientes da transformada. Para finalizar a compressão, será definido um limiar para selecionar quais coeficientes são mais significativos para serem armazenados, e quais poderão ser descartados.

1.1 MOTIVAÇÃO

Eletrocardiogramas (ECGs) são ferramentas clínicas muito utilizadas no diagnóstico de cardiopatias, capazes de fornecer informações essenciais sobre a saúde cardíaca de um determinado paciente (BENZID et al., 2007). Um ECG é o registro da variação dos potenciais elétricos gerados pela atividade elétrica do coração, que pode ser medida por eletrodos colocados sobre a pele do paciente, registrando a atividade de diferentes partes do músculo cardíaco. Para obter esse registro o paciente é monitorado por um longo período, e o volume de dados obtidos é grande (LU et al., 2000). Um exemplo de ECG pode ser observado na Figura 1.

Com o avanço da telemedicina, uma vertente da medicina em que é necessário que

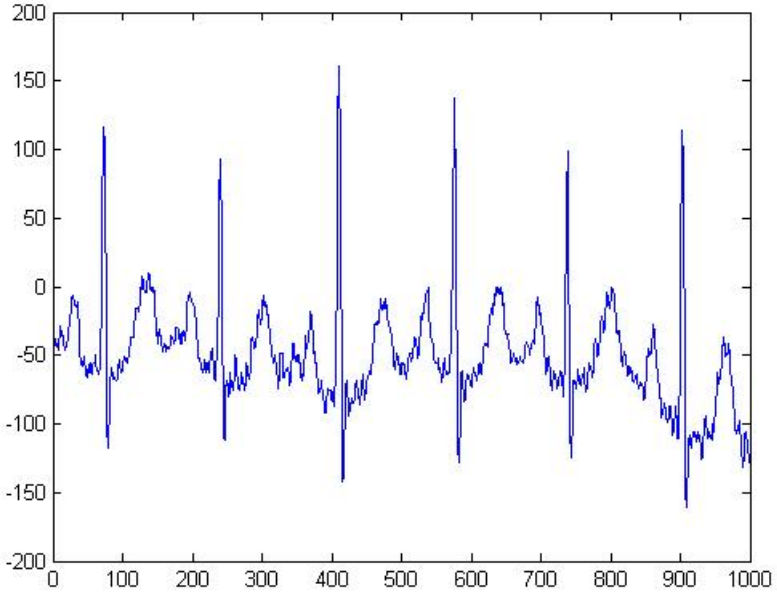


Figura 1: Exemplo de ECG

os dados do paciente possam ser obtidos remotamente e transmitidos para o destino desejado, torna-se necessário lidar com o volume grande de dados envolvido, que além de ocupar muito espaço para seu armazenamento, leva muito tempo para ser transferido.

Diante desse contexto, questiona-se: como ocupar pouco espaço e transferir dados rapidamente, quando o volume dos mesmos é muito grande?

Para tanto percebe-se, viabilidade em desenvolver um algoritmo que comprima esses dados mantendo as informações necessarias para um diagnóstico correto.

1.2 OBJETIVO GERAL

O trabalho proposto tem como objetivo comprimir sinais de eletrocardiogramas, através de uma aplicação, de forma a ocuparem pouco espaço de armazenamento e serem transferidos de forma ágil.

1.3 OBJETIVOS ESPECÍFICOS

Para atingir o objetivo principal deste trabalho, os seguintes objetivos específicos foram delimitados:

- Realizar análises bibliográficas

- Compreender as técnicas de compressão de dados
- Implementar o algoritmo de compressão
- Aplicar o algoritmo em um DSP (Digital Signal Processor)

1.4 ORGANIZAÇÃO DO TEXTO

Este trabalho foi dividido em 5 capítulos. O capítulo 2 trata-se da fundamentação teórica, e está subdividido em compressão e *wavelets* que são os principais conceitos que embasam este trabalho. Neste capítulo, também são apresentados os trabalhos relacionados. O capítulo 3, sobre o desenvolvimento do projeto, define os recursos utilizados para o desenvolvimento do projeto, tais como os métodos de análise e implementação realizados. O capítulo 4 refere-se aos resultados encontrados após a realização dos testes. Para finalizar, no capítulo 5 serão apresentadas as conclusões do trabalho, tais como limitações e trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta os principais conceitos envolvidos neste projeto para uma compreensão satisfatória do escopo do trabalho, e está dividido em duas seções que explicam de forma detalhada cada conceito: compressão e *wavelets*.

2.1 COMPRESSÃO DE SINAIS

Os processos de redução de dados podem ser divididos em dois tipos: compressão e compactação. Em termos gerais, na compressão pode haver perda de dados, e na compactação os dados permanecem os mesmos após sua recuperação (SALOMON, 2007).

Para aplicações em que é aceitável que haja uma perda nos dados, como por exemplo, sinais de imagens ou áudio, é comum aplicar técnicas de compressão antes de seu armazenamento ou transmissão. Como exemplo de compressão podem-se citar arquivos em formato JPEG, que é um formato de imagem comprimido. Na Figura 2, é possível ver duas imagens comprimidas usando o formato JPEG, utilizando diferentes taxas de compressão. A Figura (a) corresponde à imagem sob uma menor taxa de compressão, enquanto a Figura (b) corresponde à imagem comprimida sob uma maior taxa. Observando as duas imagens é possível perceber que a qualidade da Figura (b) é inferior, indicando a distorção resultante do processo de compressão. Por outro lado a Figura (a) tem $81,7$ KB e a Figura (b) possui $60,6$ KB, ou seja, a imagem com maior taxa de compressão ocupa um espaço menor, resultado direto da perda de dados no processo de compressão.

Há aplicações, no entanto, em que não é aceitável que haja perda de dados, como arquivos de texto ou binários executáveis. Nestes casos, recorre-se ao processo de compactação. Os métodos de compactação de dados consistem em eliminar redundâncias das sequências que possuem dados equivalentes. Quando trata-se de compactação tem-se como exemplo arquivos *zip*. Neste formato, os arquivos são compactados de forma a ocuparem pouco espaço, porém, sem que nenhum dado seja perdido.

(a) Imagem original (b) Imagem comprimida

Figura 2: Compressão com JPEG
Fonte: (FERNANDES, 2014)

Esse trabalho terá como foco o processo de compressão de dados, no qual pode-se obter uma economia de espaço em dispositivos de armazenamento e ganhar desempenho em transmissões, ou seja, diminuir o tempo para transmitir esses dados entre dispositivos.

O desafio nesse processo é a possibilidade de reconstituição dos dados sem que haja muita perda, pois o diagnóstico de cardiopatias é sensível à forma dos ECGs. Para que a reconstrução de sinais de ECGs seja de alta qualidade e com o menor nível de distorção possível, várias medidas de distorção são usadas (BRAUNWALD, 1997). Entre elas, destaca-se o percentual da raiz média quadrática das diferenças (PRD), que pode ser vista na Equação (1) (AGULHARI, 2009),

$$PRD = \sqrt{\frac{\sum_{i=1}^{N} \left(f[k] - \hat{f}[k]\right)^2}{\sum_{i=1}^{N} \left(f[k] - \bar{f}\right)^2}} \tag{1}$$

sendo $f[k]$ as amostras k do ECG original, $\hat{f}[k]$ as amostras k do ECG reconstruído, ambas de comprimento N, e $\bar{f}$ o valor médio do ECG original. Muitos autores não fazem a subtração do valor médio no denominador, ou subtraem o valor 1024, dado que, nos bancos de dados de arritmia do MIT-BIH (PHYSIOBANK, 2015) o valor 1024 é somado em todos os sinais para que todas as amostras sejam positivas e facilite o armazenamento dos dados (AGULHARI, 2009) (BENZID et al., 2007). A subtração de um valor menor que $\bar{f}$, ou não subtrair número nenhum, causa uma diminuição artificial no valor do PRD e, portanto, mascara a distorção do sinal reconstruído.

Para medir o desempenho da compressão dos dados e realizar comparações entre as técnicas de compressão existentes, a medida mais utilizada é a taxa de compressão (CR, dada por

$$CR = \frac{bits_{DO}}{bits_{DC}}, \tag{2}$$

sendo $bits_{DO}$, o tamanho dos dados originais e $bits_{DC}$ o tamanho dos dados comprimidos. Note que, quanto maior o valor da taxa, maior será o desempenho do processo de compressão.

Os métodos de compressão podem ser divididos em duas categorias: os métodos diretos e os métodos de transformadas. Nos métodos diretos, as amostras de sinal são analisadas diretamente para eliminação de redundâncias (JALALEDDINE et al., 1990), enquanto que, nos métodos de transformadas, o sinal é transformado em outra representação adequada para a eliminação de redundâncias (AGULHARI, 2009).

Os métodos de tranformadas consistem em escolher uma base de representação, calcular os coeficientes de projeção do sinal na base escolhida, reter os coeficientes mais significativos, quantizar e codificar os coeficientes. Para a escolha da transformada a ser utilizada, deve ser levado em conta que a energia do sinal esteja concentrada em poucos coeficientes de projeção, para que assim seja possível representar o sinal com poucos coeficientes e com uma distorção aceitável (AGULHARI, 2009).

Na compressão pelos métodos de transformadas pode-se citar a tranformada DCT (*Discrete Cosine Transform*) (AHMED et al., 1974), que consiste na representação de um sinal utilizando como base a função cosseno, e transformada de *wavelet*, que será utilizada neste trabalho e explicada em detalhes na seção 2.2.

Dentre os métodos utilizando a transformada de *wavelet*, pode-se citar o SPIHT (*Set Partitioning in Hierarchical Trees*), desenvolvido por (LU et al., 2000), que codifica tanto os valores dos coeficientes significativos quanto suas posições em uma única string de bits, garantindo assim que os coeficientes mais significativos sejam codificados antes dos coeficientes menos significativos e que a transmissão ou o armazenamento da string de bits possam ser interrompidos a qualquer momento sem comprometer a compressão (LU et al., 2000). Outro método, que apresenta resultados satisfatórios, proposto por (RAJOUB, 2002), inicia-se com a retirada do valor médio do sinal e inserção de zeros nas bordas. A transformada wavelet é aplicada ao sinal e os coeficientes com valores absolutos maiores que um certo limiar são retidos (RAJOUB, 2002).

A Compressão via Transformada *wavelets*, que será utilizada nesse trabalho, pode ser ilustrada na Figura 3. Nesta técnica o sinal do ECG é coletado por um sensor, a transformada *wavelet* é aplicada sobre o sinal, os coeficientes são retidos, codificados e pronto para serem armazenados ou transmitidos.

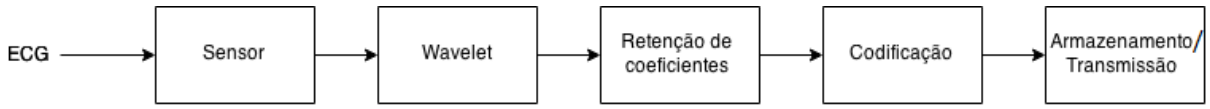


Figura 3: Compressão com *Wavelets*

O processo de retenção de coeficientes pode ser feito através de vários métodos, como por exemplo, reter os coeficientes com valor absoluto maior do que um limiar ou reter apenas um número fixo, definido a priori, dos maiores coeficientes (BENZID et al., 2007). O método que será utilizado neste trabalho consiste em reter os coeficientes que se encontram dentro de um valor limiar, conhecido como *threshold*.

Para obtenção do valor de *threshold*, será feito um estudo, descrito na seção 3.3.1, utilizando vários sinais de ECGs como entrada, aplicando a Equação (1), para analisar qual será o valor da distorção e assim determinar qual o melhor valor de *threshold* que poderá ser utilizado.

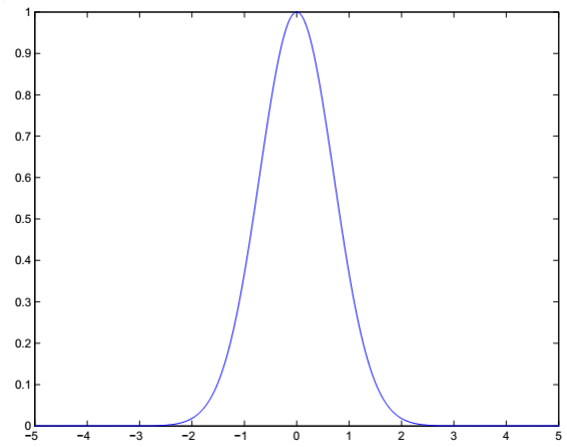
2.2 *WAVELETS*

As *wavelets* tem sido utilizadas em diversas aplicações, como no campo de processamento de sinais, para a compressão, transmissão e recepção dos dados pelos diversos dispositivos existentes no mercado. Aplicando a transformada *wavelet* em um dado sinal, os coeficientes resultantes são os coeficientes de projeção do sinal na base formada por funções *wavelet* e por funções escala (BURRUS et al., 1998) (WEEKS, 2012).

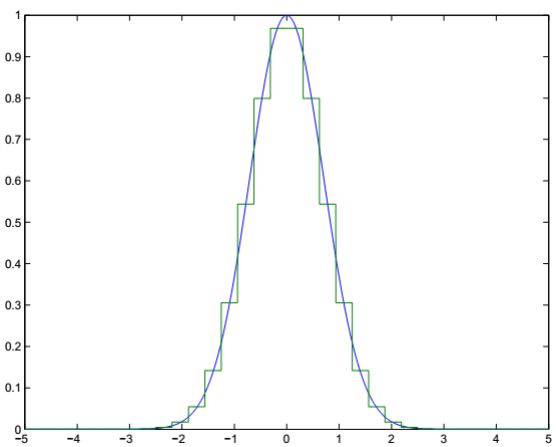
As funções escala são funções complementares às funções *wavelets* (BURRUS et al., 1998), e podem ser definidas por (3), sendo φ a representação da função escala e $H_0[k]$ um vetor composto pelos coeficientes do filtro escala. Pode-se notar que a função escala é uma função que pode ser descrita em termos da soma de deslocamentos da própria função contraída. Essa propriedade torna a função escala uma base de representação boa para aproximar funções.

$$\varphi(x) = \sum_{k=0}^{m-1} H_0[k]\sqrt{2}\varphi(2x-k) \tag{3}$$

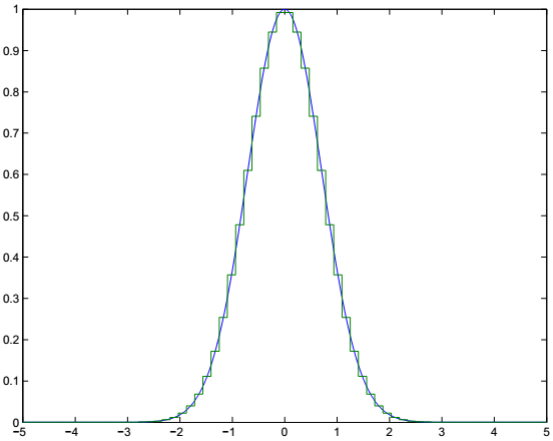
Para ilustrar a ideia da utilização da função escala como base de representação, pode-se utilizar a função gaussiana como exemplo. Na Figura 4, é possível observar que quanto menor a largura da função escala, maior a resolução e melhor a aproximação. Essa resolução é definida na Equação (4), representada por η.



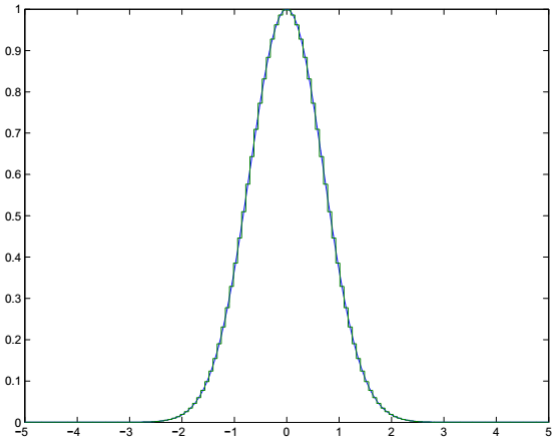
(a) Função gaussiana original



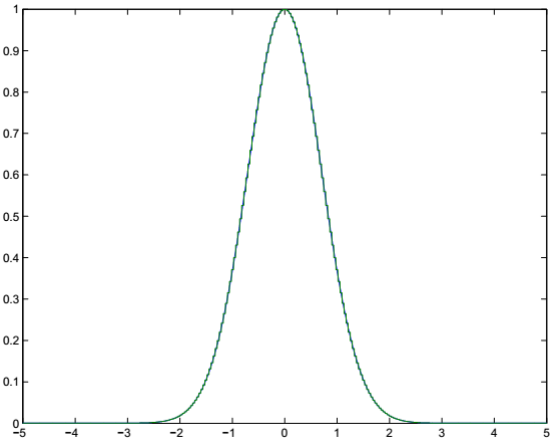
(b) Função escala com $\eta = 1$



(c) Função escala com $\eta = 2$



(d) Função escala com $\eta = 3$



(e) Função escala com $\eta = 4$

Figura 4: Aproximações com função escala

$$\varphi(2^{\eta}x) = \sum_{k=0}^{m-1} H_0[k]\sqrt{2^{\eta+1}}\varphi(2^{\eta+1}x-k) \tag{4}$$

Através da Figura 4, pode-se observar em 3(a) a função gaussiana na sua forma ori-

ginal, em 3(b) a representação da função gaussiana e da função escala com $\eta = 1$, em 3(c) a representação da função gaussiana e da função escala com $\eta = 2$ e assim por diante até $\eta = 4$. Analisando as cinco imagens representadas, pode-se observar que quanto maior o valor de η na função escala, mais próxima sua representação está da função gaussiana original.

Para que o sinal seja bem representado, o valor de η deve ser muito alto, contudo, existem as funções *wavelets*, que são funções ortonormais à função escala (STRANG; NGUYEN, 1997), e tem como finalidade aproximar o erro. A função *wavelet* é dada por (5), e possibilita que o sinal seja bem representado, com um valor de η pequeno.

$$\psi(2^{\eta}x) = \sum_{k=0}^{m-1} H_1[k]\sqrt{2^{\eta+1}}\varphi(2^{\eta+1}x-k) \tag{5}$$

Para ilustrar a representação da função escala e da função *wavelet*, tem-se a Figura 5.

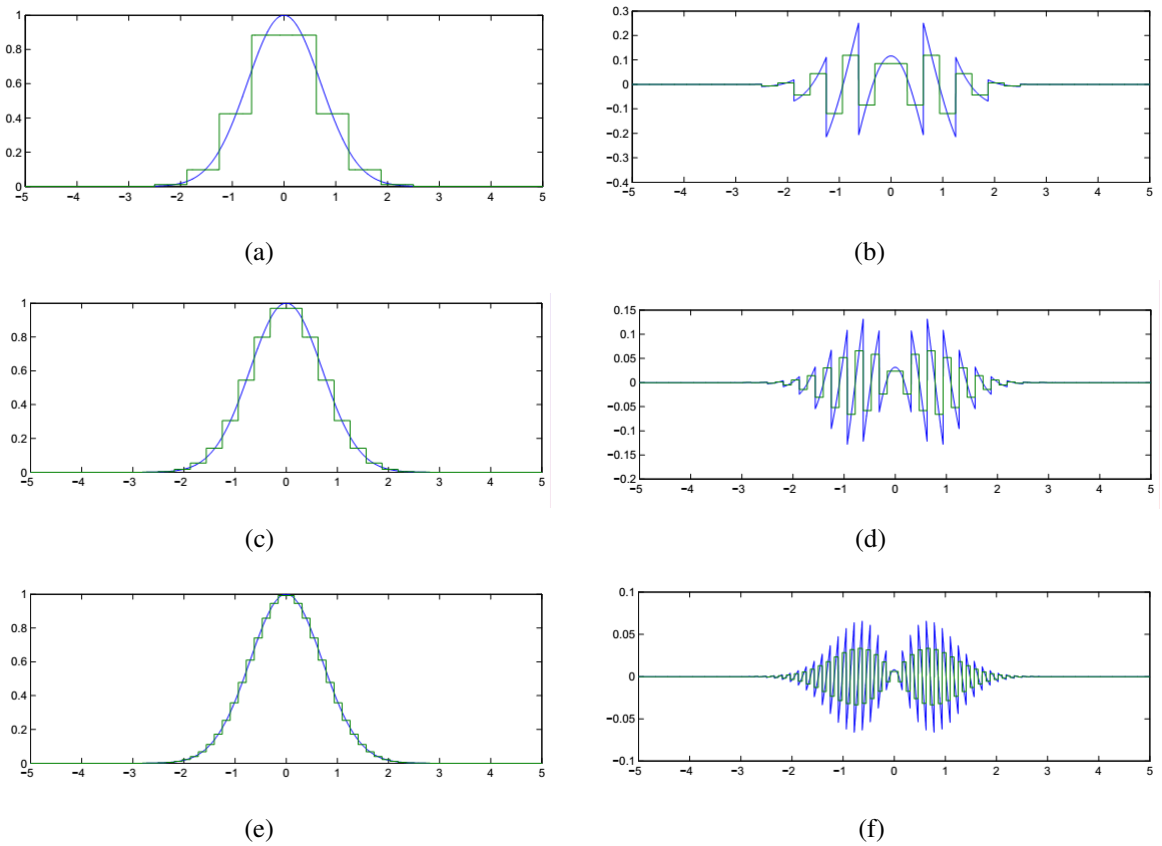


Figura 5: Aproximações com funções escalas e *wavelets*

A transformada *wavelet* consiste em representar o sinal na base de funções escala para um valor pré-determinado de η, e também representar o erro de aproximação na base *wavelet* nas resoluções seguintes.

Quando se trata de sinais discretos, há técnicas que facilitam o cálculo dos coeficientes

de projeção do sinal nas bases escala e *wavelet*. Suponha que $a_\eta[n]$ são os coeficientes de projeção de $f[n]$ na base escala, e $b_\eta[n]$ os coeficientes de projeção de $f[n]$ na base *wavelet*. De acordo com (STRANG; NGUYEN, 1997), tem-se que

$$a_{\eta-1}[n] = \sum_k H_0[k-2n]a_\eta[k] = (H_0[-n] * a_\eta[n])_{\downarrow 2}, \tag{6}$$

$$b_{\eta-1}[n] = \sum_k H_1[k-2n]a_\eta[k] = (H_1[-n] * a_\eta[n])_{\downarrow 2} \tag{7}$$

e $f[n] = a_\eta[n]$, sendo $f[n]$ as amostras do sinal original. Note que as equações 6 e 7 representam um processo de filtragem.

Os coeficientes de projeção $a_\eta[n]$, $b_\eta[n]$ e a resolução η das funções escala e *wavelet* são utilizados na implementação de um banco de filtros. A Figura 6 ilustra a ideia do banco de filtros utilizado, sendo H_1 o filtro *wavelet*, ou seja, o filtro que resulta nos coeficientes de representação *wavelet* que serão armazenados, e H_0 o filtro escala, que resulta nos coeficientes de representação escala. Como pode ser observado na Figura 6, os coeficientes escala obtidos, são sub-amostrados e passam novamente pelo processo de filtragem no filtro *wavelet* e no filtro escala, de forma recorrente. Tal processo ocorre de acordo com o valor de η, ou seja, os níveis do banco de filtros correspondem ao número de vezes em que esse processo ocorre. O banco de filtros representado na Figura 6, por exemplo, possui três níveis, ou seja $\eta = 3$.

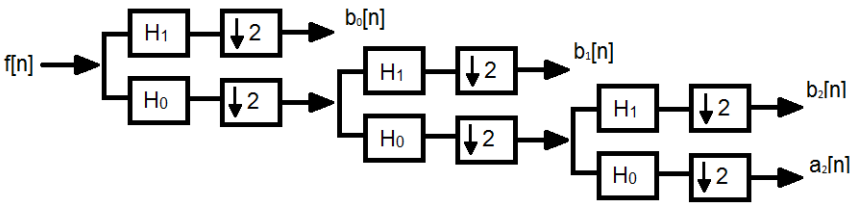


Figura 6: Banco de filtros

Para determinar o valor de η são necessários testes preliminares com um conjunto de ECGs, aplicando a transformada *wavelet* usando vários valores para η, retendo um número fixo de coeficientes, e analisando a distorção resultante para cada valor de η. Tal procedimento é descrito em detalhes na seção 3.3.1.

Após o sinal de entrada passar pelo banco de filtros, os coeficientes *wavelet* (y_1, y_{01}, y_{001}) e escala (y_{000}) são armazenados. Para comprimir o sinal, um valor de *threshold* (*limiar*) deve ser estipulado definindo quais coeficientes podem ser descartados e quais devem ser armazenados, sem que haja uma grande distorção. Para obter esse valor será realizado um estudo utilizando

os coeficientes da *wavelet de Daubechies-3*, porque, em testes realizados anteriormente (AGULHARI, 2009), pode-se perceber que essa é a *wavelet* clássica que melhor comprime os sinais de eletrocardiogramas. Essa *wavelet* será utilizada também na implementação do banco de filtros, pela sua maior simplicidade. Na tabela 1 encontram-se alguns exemplos de coeficientes *wavelets*, inclusive os coeficientes da *wavelet de Daubechies-3*.

Tabela 1: Coeficientes *wavelets*
Fonte: (AGULHARI, 2009)(adaptado)

Nome	Coeficientes
Haar	$[-0.7071, 0.7071]$
Daubechies-2	$[-0.4830, 0.8365, -0.2241, -0.1294]$
Daubechies-3	$[-0.3327, 0.8069, -0.4599, -0.1350, 0.0854, 0.0352]$

3 DESENVOLVIMENTO

Neste capítulo será descrito como foi desenvolvido o projeto proposto, e está dividido em três seções: recursos utilizados, métodos e análise e implementação.

3.1 RECURSOS UTILIZADOS

Os recursos utilizados para o desenvolvimento do trabalho proposto foram a IDE Code Composer Studio v4 (CCS), o software Matlab®, versão 2012a, o sistema operacional Windows 8.1 e um DSP, modelo TMS320C5515 eZdsp USB Stick da Texas Instruments™.

3.2 MÉTODOS

Inicialmente o código foi desenvolvido e compilado na IDE, em seguida o algoritmo foi implementado e processado no DSP. O algoritmo implementado consiste, em termos gerais, na aplicação do ECG recebido no banco de filtros apresentado na seção 2.2, na retenção dos coeficientes com valor absoluto maior que um *threshold* pré-determinado, o qual será responsável pela compressão do sinal, e o descarte dos coeficientes presentes no intervalo do valor absoluto deste *threshold*, e na codificação dos coeficientes retidos e seu posterior armazenamento. O Fluxograma apresentado na Figura 7, representa passo a passo a execução da aplicação de compressão.

Após a implementação do algoritmo, foi necessário realizar um estudo para o valor dos parâmetros necessários para uma melhor compressão, no caso no nível do banco de filtros e no valor de *threshold*. Com o algoritmo adaptado aos valores obtidos, foi estudada a melhor forma para transmitir os dados dos ECGs do computador para o DSP. A forma encontrada foi através da ferramenta *Simulink* do software *Matlab*, transmitindo o ECG pela saída de áudio do computador diretamente para a entrada de áudio do DSP.

O último passo foi a realização de testes com dados de domínio público.

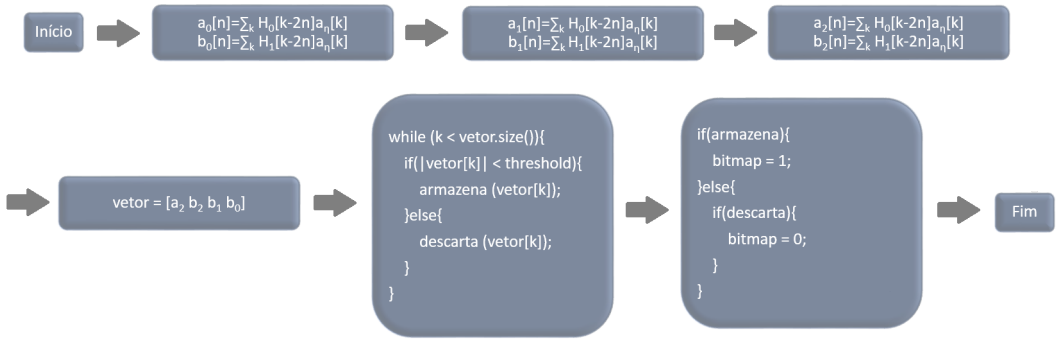


Figura 7: Fluxograma do algoritmo de compressão

3.3 ANÁLISE E IMPLEMENTAÇÃO

Nesta seção serão descritas detalhadamente as etapas de desenvolvimento do projeto, em quatro subseções: determinação dos parâmetros, validação do banco de filtros, implementação da compressão e transporte de dados.

3.3.1 DETERMINAÇÃO DOS PARÂMETROS

Para determinar o melhor valor para o nível do banco de filtros(η) e o melhor valor de *threshold* a serem utilizados pelo algoritmo de compressão, foi necessário realizar testes com vários eletrocardiogramas, utilizando a ferramenta Matlab®.

Os testes foram realizados utilizando vários ECGs de três diferentes tipos da base de dados MIT-BIH (PHYSIOBANK, 2015). O valor máximo de PRD aceitável para a distorção entre o sinal original e o sinal comprimido é de 10% uma vez que estudos na literatura (AGULHARI et al., 2008) afirmam que o diagnóstico não é prejudicado para tal valor. Em cada ECG foi calculado o valor de distorção PRD variando os valores para os níveis, de 2 a 6, e os valores de threshold, de 0 a 100, e armazenando o valor de threshold que gerou o PRD mais próximo do valor 10%. Cada tipo de ECG analisado gerou uma matriz de dados em que as linhas correspondem aos níveis e as colunas a cada ECG analisado.

Para escolha dos valores que serão utilizados, de forma a comprimir o máximo possível o sinal, sem interferir no diagnóstico de cardiopatias, foi realizada uma análise de qual o menor valor de *threshold* encontrado em cada nível para vários ECGs de três tipos diferentes, como pode ser observado na Tabela 2, onde cada Tipo representa vários ECGs com o mesmo tipo de cardiopatia.

Escolheu-se o nível em que o menor valor de *threshold* foi maior que o menor *threshold*

dos outros níveis, ou seja, o nível escolhido foi o nível 2, pois 15.66 é o maior valor entre os menores *thresholds* encontrados. E para a escolha do *threshold* que será utilizado, foi analisado o maior valor encontrado para cada tipo de ECG, e foi escolhido o maior valor para o nível escolhido, ou seja, o *threshold* que será utilizado será o 32.29.

Tabela 2: Menor valor de *threshold* para cada nível η de decomposição

ECG	Nível 2	Nível 3	Nível 4	Nível 5	Nível 6
Tipo 1	32.29	20.16	15.12	13.11	12.64
Tipo 2	15.66	10.88	9.71	9.29	9.21
Tipo 3	60.85	71.49	63.88	61.49	60.86

A Figura 8 ilustra a aplicação do *threshold* para compressão dos ECGs, cada coluna representa um valor de coeficiente, após a passagem do ECG pelo banco de filtros, e a linha preta corresponde ao valor do *threshold*, a Figura 9 representa o gráfico dos coeficientes resultantes do banco de filtros, após a aplicação do *threshold*.

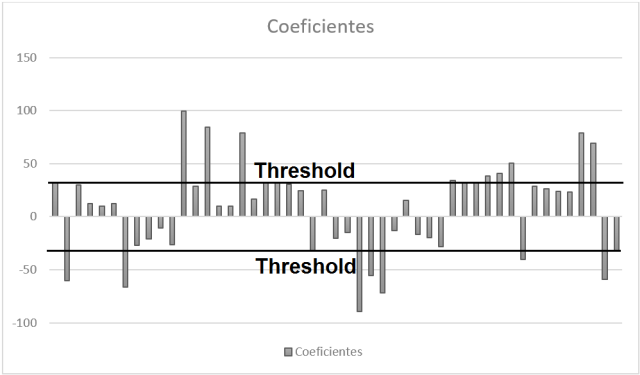


Figura 8: Gráfico de corte de coeficientes

Figura 9: Gráfico resultante do corte de coeficientes

3.3.2 VALIDAÇÃO DO BANCO DE FILTROS

Para validar a funcionalidade do banco de filtros implementado, foram utilizados alguns coeficientes de alguns ECGs, testando-os diretamente no código, ou seja, sem que haja a transmissão do computador para o DSP, e comparado com o resultado obtido no Matlab®.

Após realizar os testes com vários ECGs, foi possível comprovar que o banco de filtros foi implementado corretamente, e já seria possível implementar a compressão dos sinais.

3.3.3 IMPLEMENTAÇÃO DA COMPRESSÃO

Após serem definidos os parâmetros e validada a implementação do banco de filtros, foi possível implementar a compressão dos sinais. A implementação foi realizada analisando o vetor final com os vetores de coeficientes resultantes do banco de filtros (y_1,y_{01},y_{00}), implementado de acordo com o valor do nível obtido nos testes para determinação dos parâmetros ilustrado na Figura 10, e descartando os coeficientes dentro do limiar $-32.29 < x < 32.29$, ou seja, x representa os valores que serão descartados.

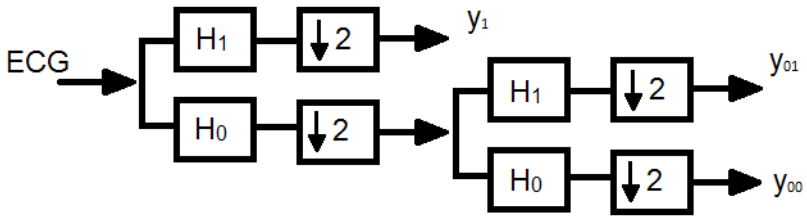


Figura 10: Banco de filtros
Fonte: (VETTERLI; HERLEY, 1992)Adaptado

3.3.4 TRANSPORTE DE DADOS

A aquisição dos ECGs foi obtida pelo computador, através de uma base de dados ja existente (PHYSIOBANK, 2015). Para transportar os dados do ECG para o DSP foram utilizados dois métodos, um método direto em que os dados do ECG foram definidos diretamente no código de compressão, e um método via *Simulink*, em que os dados do ECG foram enviados para o DSP, através da ferramenta e da saida de áudio do computador, para a entrada de áudio do DSP.

Pelo método direto, os dados do ECG foram copiados do *Matlab* e colados em um vetor no código de compressão. E pelo método via *Simulink*, foi utilizada a ferramenta e a IDE

CCS. O transporte foi realizado utilizando no *Simulink*, o modelo ilustrado na Figura 11, onde o *Signal From Workspace* (ECG), é um ECG do banco de dados MIT-BIH (PHYSIOBANK, 2015), que foi carregado no *workspace* da ferramenta *Matlab*, e esse ECG foi enviado para a saída de áudio do computador e para o *workspace* novamente, para futura comparação.

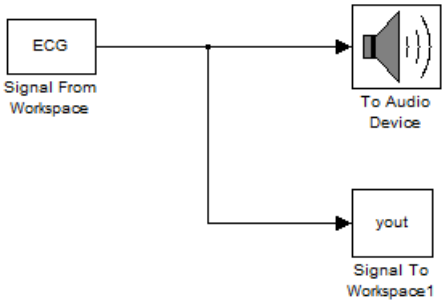


Figura 11: Modelo Simulink

Para obter um sinal de ECG o mais próximo possível do ECG original, no CCS, foi necessário adaptar o sinal de entrada original, dividindo o valor dos seus coeficientes por 1000 e alterar alguns parâmetros nos blocos do modelo do *Simulink* e no código do CCS.

No modelo a taxa de envio de dados foi de 1000 dados por segundo, e a saída de dados do dispositivo de áudio foi por repetição cíclica, para evitar perda de amostras. Após muitos testes, pôde-se observar que era necessário sincronizar o *Simulink* ao DSP, pois o tempo de simulação transcorria diferente do tempo real. Os testes realizados mostraram que 1 segundo no DSP equivalia a 0.767 segundos no *Simulink*. Para sincronizar o tempo em que as amostras são enviadas pelo *Simulink* e o tempo em que elas são recebidas no DSP, foi necessário utilizar o método *waitusec()*, existente na biblioteca *usbstk5515.h*, que é a biblioteca que inicializa a comunicação da IDE com o DSP, com parâmetro igual a 0.767. Com isso, pode-se obter no DSP um sinal bem próximo ao sinal do ECG original.

Para poder calcular o PRD nos testes foi necessário normalizar o sinal do ECG reconstruído, centralizando-o em zero e movendo-o circularmente em torno do eixo y, para posicioná-lo de acordo com a posição do sinal original, pois por enviar o sinal de forma cíclica para o DSP, sua posição final pode não ser equivalente a posição inicial do sinal original. Os coeficientes resultantes da compressão, foram obtidos via o *console* do CCS, e copiado para o Matlab®, para que o sinal fosse reconstruído, utilizando o comando *waverec* da ferramenta. Os resultados obtidos nos testes são apresentados no capítulo seguinte.

4 RESULTADOS E DISCUSSÕES

Foram realizados testes com as duas métodos de tranporte de dados, descritos na sessão 3.3.4. Primeiramente foram feitos testes utilizando os dados dos ECGs diretamente em um vetor no código de compressão, o resultado da compressão foi obtido através do console da IDE, copiado e colado no Matlab, para realizar a reconstrução do sinal, por meio do comando *waverec* e calcular a distorção(PRD) obtido no sinal comprimido em relação ao sinal original.

Em seguida, foram realizados testes enviando os dados dos ECGs via *Simulink* pela saída de áudio do computador para a entrada de áudio do DSP. O resultado da compressão também foi obtido via console da IDE, copiado e colado no Matlab, para reconstrução do sinal e calculo do PRD entre o sinal original e o sinal comprimido.

Os resultados obtidos com esses testes podem ser observados na Tabela 3. Onde foram utilizados nove ECGs de três tipos diferentes, e calculado o PRD para os dois métodos de transporte de dados.

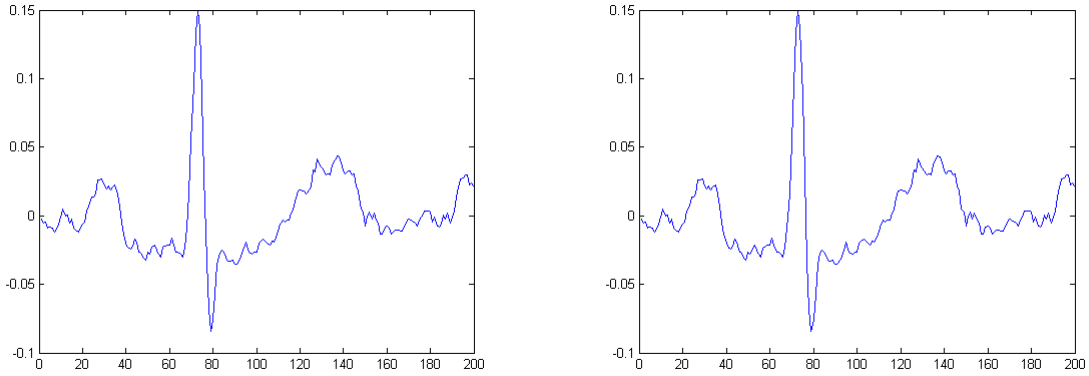
Tabela 3: Resultados do PRD após a compressão dos ECGs

ECG	TIPO DE ECG	SINAL DIRETO	SINAL VIA SIMULINK
1	TIPO 1	1,57E-04	4,0115
2	TIPO 1	1,35E-04	7,7714
3	TIPO 1	1,39E-04	7,6067
4	TIPO 2	1,56E-04	8,0932
5	TIPO 2	1,88E-04	1,9304
6	TIPO 2	2,43E-04	5,0818
7	TIPO 3	1,56E-04	0,7045
8	TIPO 3	1,57E-04	0,7784
9	TIPO 3	1,55E-04	0,8369

Como pode ser observado nos valores apresentados, o PRD do ECG comprimido com os valores dos coeficientes aplicados diretamente no código são muito menores que os valores do ECG enviados via simulink, isso ocorreu por alguma interferência no momento de transferir os dados. Outro fator para que o PRD resultante com os coeficientes aplicados diretamente no código seja tão pequeno é o tamanho de cada ECG utilizado. O tamanho da amostra utilizada

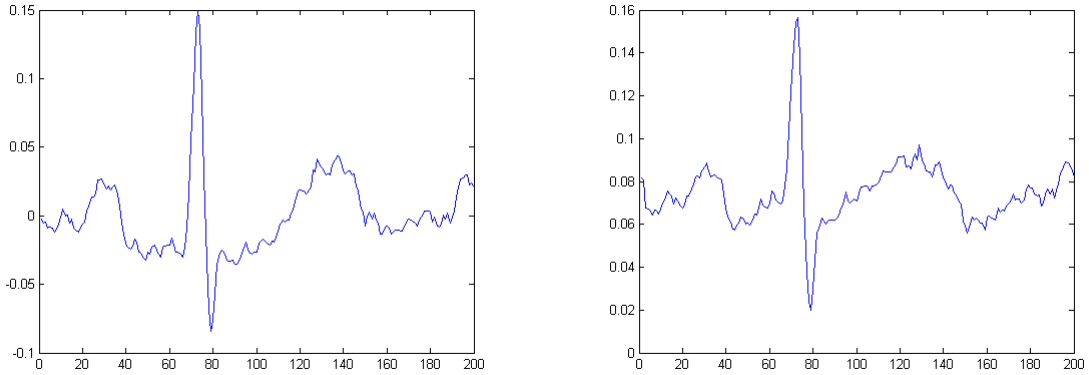
foi de 200 coeficientes, isso devido a quantidade de memória suportada pelo DSP, mais que 200 coeficientes resultava em estouro de memória no momento de executar o código recebendo o ECG via simulink.

Note que mesmo havendo interferências ao transferir os dados do computador para o DSP via *Simulink*, os resultados obtidos para o valor do PRD, são menores que os 10% considerados para não prejudicar o diagnóstico correto do ECG.



(a) Sinal original (b) Sinal comprimido

Figura 12: Resultado com código direto



(a) Sinal original (b) Sinal comprimido

Figura 13: Compressão via *Simulink*

As Figuras 12 e 13 ilustram os ECGs em sua forma original e em sua forma comprimida através dos dois métodos utilizados. Na Figura 12(a) pode-se observar o ECG no seu formato original e na Figura 12(b) o sinal comprimido, utilizando o sinal diretamente no código. E na Figura 12(a) tem-se o ECG no seu formato original também e na Figura 12(b) o sinal comprimido, obtido através da saída de áudio do computador.

5 CONCLUSÃO

O presente trabalho apresentou um algoritmo para a compressão de sinais de ECGs, com a finalidade de reduzir o máximo possível o tamanho do ECG, de forma a não interferir no diagnóstico de cardiopatias.

Primeiramente, foi realizada a implementação de um algoritmo, no DSP, que consiste na aplicação do ECG recebido no banco de filtros. Em seguida, foram estudados valores para o nível do banco de filtros e o *threshold* para ser utilizado na compressão dos ECGs. Para finalizar, foram realizados testes com dados de domínio público.

Através dos testes realizados, foram obtidos alguns resultados satisfatórios, como a eficiência na compressão dos sinais, porém, com algumas limitações, entre elas encontra-se o transporte dos dados do ECG, que será comprimido, para o DSP, pois durante esse processo, ocorrem algumas interferências que contribuem para o aumento da distorção do sinal reconstruído após a compressão e o tamanho do trecho de sinal que pode ser comprimido, devido à quantidade de memória disponível no DSP, portanto, só foi possível comprimir os ECGs em partes.

Como sugestões de trabalhos futuros tem-se o estudo de uma melhor forma de comunicação entre o computador e o DSP, para o transporte dos ECGs que serão comprimidos, o estudo para aumentar a quantidade de memória suportada pelo DSP, para conseguir comprimir o ECG inteiro em uma única execução do algoritmo, o desenvolvimento de uma técnica para coleta dos dados diretamente do paciente e o desenvolvimento de um algoritmo que comprima os ECGs e retorne eles comprimidos por um canal de saída de dados do DSP.

REFERÊNCIAS

ABENSTEIN, J. P.; TOMPKINS, W. J. A new data-reduction algorithm for real-time ECG analysis. **Biomedical Engineering, IEEE Transactions on**, BME-29, n. 1, p. 43–48, Jan 1982. ISSN 0018-9294.

AGULHARI, C. M. **Compressão de eletrocardiogramas usando wavelets**. Dissertação (Mestrado) — Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação. , UNICAMP, 2009.

AGULHARI, C. M.; SILVEIRA, R. M. R.; BONATTI, I. S. Compressing electrocardiogram signals using parameterized wavelets. In: **Proceedings of the 2008 ACM Symposium on Applied Computing**. New York, NY, USA: ACM, 2008. (SAC '08), p. 1348–1352. ISBN 978-1-59593-753-7. Disponível em: <http://doi.acm.org/10.1145/1363686.1363999>.

AHMED, N.; NATARAJAN, T.; RAO, K. Discrete cosine transform. **Computers, IEEE Transactions on**, C-23, n. 1, p. 90–93, Jan 1974. ISSN 0018-9340.

BENZID, R.; MARIR, F.; BOUGUECHAL, N. Electrocardiogram compression method based on the adaptive wavelet coefficients quantization combined to a modified two-role encoder. **Signal Processing Letters, IEEE**, v. 14, n. 6, p. 373–376, June 2007. ISSN 1070-9908.

BRAUNWALD, E. **Heart Disease: A Textbook of Cardiovascular Medicine**. 5th edition. ed. : W. B. Saunders Company, 1997.

BURRUS, C. S.; GOPINATH, R. A.; GUO, H. **Introduction to Wavelets and Wavelet Transforms**. 1st edition. ed. : A Primer (Prentice Hall), 1998.

DENG, L.; POOLE, M. Learning through telemedicine networks. In: **System Sciences, 2003. Proceedings of the 36th Annual Hawaii International Conference on**. 2003. p. 8 pp.–.

FERNANDES, W. J. A. An image compression technique using wavelets. In: **IEEE TechSym Satellite Conference**. 2014. p. 5–6.

GRAPS, A. An introduction to wavelets. **IEEE Computational Science & Engineering**, p. 50–61, jun. 1995.

HAO, Y.; MARZILIANO, P. An efficient wavelet-based pattern matching scheme for ECG data compression. In: **Biomedical Circuits and Systems, 2004 IEEE International Workshop on**. 2004. p. S2/4–S5–8.

JALALEDDINE, S. et al. ECG data compression techniques-a unified approach. **Biomedical Engineering, IEEE Transactions on**, v. 37, n. 4, p. 329–343, April 1990. ISSN 0018-9294.

JANE, R. et al. Evaluation of an automatic threshold based detector of waveform limits in holter ecg with the qt database. In: **Computers in Cardiology 1997**. 1997. p. 295–298. ISSN 0276-6547.

LIMA, P. C. Wavelets: uma introdução. **ICEX**, Belo Horizonte, v. 1, p. 13–44, dez. 2002.

LU, Z.; KIM, D. Y.; PEARLMAN, W. Wavelet compression of ECG signals by the set partitioning in hierarchical trees algorithm. **Biomedical Engineering, IEEE Transactions on**, v. 47, n. 7, p. 849–856, July 2000. ISSN 0018-9294.

MANIKANDAN, M.; DANDAPAT, S. Wavelet-based ECG and PCG signals compression technique for mobile telemedicine. In: **Advanced Computing and Communications, 2007. ADCOM 2007. International Conference on**. 2007. p. 164–169.

PHYSIOBANK. **PhysioBank - Physiologic signal archives for biomedical research**. 2015. Disponível em: <http://www.physionet.org/physiobank>.

RAJOUB, B. A. An efficient coding algorithm for the compression of ECG signals using the wavelet transform. **IEEE Transactions on Biomedical Engineering**, v. 49, p. 355–362, 2002.

SALOMON, D. **Data Compression - The Complete Reference**. : Springer, 2007.

STRANG, G.; NGUYEN, T. **Wavelets and Filter Banks**. : Wellesley-Cambridge Press, 1997.

VETTERLI, M.; HERLEY, C. Wavelets and filter banks: theory and design. **Signal Processing, IEEE Transactions on**, v. 40, n. 9, p. 2207–2232, Sep 1992. ISSN 1053-587X.

WEEKS, M. **Processamento Digital de Sinais Utilizando Matlab e Wavelets**. 2ª. ed. : LTC, 2012.

ANEXO A – CÓDIGO PARA COMPRESSÃO DO ECG

```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
#include "stdio.h"
#include "usbstk5515.h"
#include "usbstk5515_gpio.h"
#include "usbstk5515_i2c.h"

#define Rcv 0x08
#define Xmit 0x20
#define AIC3204_I2C_ADDR 0x18a
#define n 200
#define m 6

Int16 vetor[n];

Int16 AIC3204_rset( Uint16 regnum, Uint16 regval ){

    Uint8 cmd[2];
    cmd[0] = regnum & 0x007F;       // 7-bit Register Address
    cmd[1] = regval;                // 8-bit Register Data

    return USBSTK5515_I2C_write( AIC3204_I2C_ADDR, cmd, 2 );
}


void carregandoECG(){

    Int16 sample, data1, data2, data3, data4;
    int l=0;
```

```
31
32      AIC3204_rset( 0, 0 );
33      AIC3204_rset( 1, 1 );
34      AIC3204_rset( 0, 1 );
35      AIC3204_rset( 1, 8 );
36      AIC3204_rset( 2, 1 );
37      AIC3204_rset( 0, 0 );
38
39      AIC3204_rset( 27, 0x0d );
40      AIC3204_rset( 28, 0x00 );
41      AIC3204_rset( 4, 3 );
42      AIC3204_rset( 6, 7 );
43      AIC3204_rset( 7, 0x06 );
44      AIC3204_rset( 8, 0x90 );
45      AIC3204_rset( 30, 0x88 );
46
47      AIC3204_rset( 5, 0x91 );
48      AIC3204_rset( 13, 0 );
49      AIC3204_rset( 14, 0x80 );
50      AIC3204_rset( 20, 0x80 );
51      AIC3204_rset( 11, 0x82 );
52      AIC3204_rset( 12, 0x87 );
53      AIC3204_rset( 18, 0x87 );
54      AIC3204_rset( 19, 0x82 );
55
56      AIC3204_rset(  0, 0x01 );
57      AIC3204_rset( 12, 0x08 );
58      AIC3204_rset( 13, 0x08 );
59      AIC3204_rset(  0, 0x00 );
60      AIC3204_rset( 64, 0x02 );
61      AIC3204_rset( 65, 0x00 );
62      AIC3204_rset( 63, 0xd4 );
63      AIC3204_rset(  0, 0x01 );
64      AIC3204_rset( 16, 0x00 );
65      AIC3204_rset( 17, 0x00 );
66      AIC3204_rset(  9, 0x30 );
67      AIC3204_rset(  0, 0x00 );
68      USBSTK5515_wait( 500 );
```

```
69
70      AIC3204_rset( 0, 1 );
71      AIC3204_rset( 0x34, 0x30 );
72
73      AIC3204_rset( 0x37, 0x30 );
74      AIC3204_rset( 0x36, 3 );
75      AIC3204_rset( 0x39, 0xc0 );
76      AIC3204_rset( 0x3b, 0 );
77      AIC3204_rset( 0x3c, 0 );
78      AIC3204_rset( 0, 0 );
79      AIC3204_rset( 0x51, 0xc0 );
80      AIC3204_rset( 0x52, 0 );
81
82      AIC3204_rset( 0, 0 );
83      USBSTK5515_wait( 200 );
84
85      I2S0_SRGR = 0x0;
86      I2S0_CR = 0x8010;
87      I2S0_ICMR = 0x3f;
88
89      printf("\nInicio de carrega vetor");
90      for ( sample = 0 ; sample < n ; sample++ ){
91          while((Rcv & I2S0_IR) == 0);
92                  data3 = I2S0_W0_MSW_R;
93              data1 = I2S0_W0_LSW_R;
94              data4 = I2S0_W1_MSW_R;
95              data2 = I2S0_W1_LSW_R;
96                          vetor[sample] = data3;
97                          USBSTK5515_waitusec( 767 );
98      }
99
100
101     I2S0_CR = 0x00;
102
103 }
104
105 float filtro(float H[], float *X,int k){
106
```

```
107         int j=0,l=0;
108         float total=0;
109
110         for(j=0;j<m;j++){
111                 l = k - j;
112                 if(l>-1){
113                         total+=H[j]*X[l];
114                 }
115         }
116
117         return total;
118 }
119
120 int main(int argc, char * argv[]){
121
122         int     a=0, b=0, c=0, d=0, e= 0, f=0, g=0, h=0, i=0, j=0, k
                =0, l=0, o=0, p=0, q=0, r=0, s=0, t=0, u=0, v=0, w=0,
123                         a1=0, b1=0, c1=0, d1=0, d2=0, e1=0, f1=0, g1
                            =0, i1=0, j1=0, k1=0, o1=0, p1=0, q1=0, t1
                            =0, u1,
124                         tam=0;
125
126         float   H0[m] = {0.0352, -0.0854, -0.1350, 0.4599, 0.8069,
                0.3327},
127                         H1[m] = {-0.3327, 0.8069, -0.4599, -0.1350,
                            0.0854, 0.0352},
128                         *x2,
129                         *y2,
130                         *y0, *y1, *y00, *y01,
131                         *aux0, *aux1, *aux00, *aux01,
132                         *saidaBanco,
133                         *vetorComprimido;
134
135         float   vetor[n] = {-0.090205, -0.070205, -0.061205,
                -0.092205, -0.161205, -0.170205, -0.115205, -0.108205,
                -0.097205, -0.077205, -0.064205, -0.056205, -0.083205,
                -0.085205, -0.074205, -0.042205, -0.028205, -0.007205,
                0.040795, 0.059795, 0.129795, 0.227795, 0.332795,
```

```
        0.437795, 0.539795, 0.642795, 0.737795, 0.844795,
        0.964795, 1.090795, 1.180795, 1.265795, 1.363795,
        1.477795, 1.594795, 1.642795, 1.680795, 1.726795,
        1.777795, 1.813795, 1.757795, 1.699795, 1.660795,
        1.585795, 1.480795, 1.367795, 1.277795, 1.150795,
        0.935795, 0.729795, 0.573795, 0.420795, 0.303795,
        0.201795, 0.097795, 0.018795, -0.041205, -0.076205,
        -0.119205, -0.151205, -0.142205, -0.141205, -0.166205,
        -0.203205, -0.189205, -0.153205, -0.172205, -0.175205,
        -0.163205, -0.183205, -0.202205, -0.235205, -0.258205,
        -0.244205, -0.260205, -0.288205, -0.280205, -0.278205,
        -0.265205, -0.248205, -0.268205, -0.264205, -0.201205,
        -0.189205, -0.215205, -0.200205, -0.218205, -0.266205,
        -0.312205, -0.320205, -0.269205, -0.240205, -0.243205,
        -0.259205, -0.276205, -0.254205, -0.220205, -0.228205,
        -0.244205, -0.234205, -0.243205, -0.253205, -0.258205,
        -0.261205, -0.248205, -0.218205, -0.216205, -0.275205,
        -0.302205, -0.319205, -0.316205, -0.268205, -0.258205,
        -0.265205, -0.250205, -0.237205, -0.243205, -0.257205,
        -0.269205, -0.269205, -0.275205, -0.261205, -0.256205,
        -0.273205, -0.276205, -0.272205, -0.267205, -0.252205,
        -0.221205, -0.218205, -0.230205, -0.235205, -0.227205,
        -0.226205, -0.219205, -0.215205, -0.281205, -0.319205,
        -0.284205, -0.266205, -0.274205, -0.275205, -0.270205,
        -0.255205, -0.225205, -0.212205, -0.236205, -0.241205,
        -0.249205, -0.269205, -0.258205, -0.252205, -0.259205,
        -0.289205, -0.317205, -0.293205, -0.297205, -0.298205,
        -0.303205, -0.311205, -0.243205, -0.237205, -0.253205,
        -0.242205, -0.249205, -0.250205, -0.240205, -0.237205,
        -0.266205, -0.263205, -0.241205, -0.244205, -0.276205,
        -0.299205, -0.280205, -0.266205, -0.267205, -0.261205,
        -0.255205, -0.263205, -0.258205, -0.254205, -0.257205,
        -0.251205, -0.254205, -0.245205, -0.267205, -0.287205,
        -0.235205, -0.201205, -0.228205, -0.252205, -0.254205,
        -0.269205, -0.266205, -0.246205, -0.260205, -0.236205,
        -0.213205, -0.227205};
136
137     int *auxCompressao;
```

```
138
139         FILE *Bitmap;
140         FILE *ECG_Comprimido;
141
142         USBSTK5515_init( );
143
144         printf("\nInicio main");
145
146         //carregandoECG();
147
148         //CALCULANDO OS TAMANHOS DOS VETORES PARA NÍVEL 1
149         a = n + 2*m - 2;
150         b = a + m - 1;
151         c = n + m - 1;
152         d = c/2;
153
154         //CALCULANDO OS TAMANHOS DOS VETORES PARA NÍVEL 2
155         e = d + 2*m - 2;
156         f = e + m - 1;
157         g = d + m - 1;
158         h = g/2;
159
160         tam = 2*h + d;
161
162         //ALOCANDO MEMÓRIA PARA O VETOR ESTENDIDO
163         x2 = (float*) calloc (a, sizeof(float));
164         y2 = (float*) calloc (e, sizeof(float));
165         y0 = (float*) calloc (d, sizeof(float));
166         y1 = (float*) calloc (d, sizeof(float));
167         y00 = (float*) calloc (h, sizeof(float));
168         y01 = (float*) calloc (h, sizeof(float));
169         aux0 = (float*) calloc (c, sizeof(float));
170         aux1 = (float*) calloc (c, sizeof(float));
171         aux00 = (float*) calloc (g, sizeof(float));
172         aux01 = (float*) calloc (g, sizeof(float));
173         saidaBanco = (float*) calloc (tam, sizeof(float));
174         vetorComprimido = (float*) calloc (tam, sizeof(float));
175         auxCompressao = (int*) calloc (tam, sizeof(int));
```

```
176
177         if (x2 == NULL) {       printf("\n1Memória insuficiente.
               Fechando o programa...\n");    exit(1);        }
178         if (y2 == NULL) {       printf("\n2Memória insuficiente.
               Fechando o programa...\n");    exit(1);        }
179         if (y0 == NULL) {       printf("\n3Memória insuficiente.
               Fechando o programa...\n");    exit(1);        }
180         if (y1 == NULL) {       printf("\n4Memória insuficiente.
               Fechando o programa...\n");    exit(1);        }
181         if (y00 == NULL) {      printf("\n5Memória insuficiente.
               Fechando o programa...\n");    exit(1);        }
182         if (y01 == NULL) {      printf("\n6Memória insuficiente.
               Fechando o programa...\n");    exit(1);        }
183         if (aux0 == NULL) {     printf("\n7Memória insuficiente.
               Fechando o programa...\n");    exit(1);        }
184         if (aux1 == NULL) {     printf("\n8Memória insuficiente.
               Fechando o programa...\n");    exit(1);        }
185         if (aux00 == NULL) {    printf("\n9Memória insuficiente.
               Fechando o programa...\n");    exit(1);        }
186         if (aux01 == NULL) {    printf("\n10Memória insuficiente.
               Fechando o programa...\n");   exit(1);        }
187         if (saidaBanco == NULL) {       printf("\n11Memória
               insuficiente. Fechando o programa...\n");   exit(1);
                       }
188         if (vetorComprimido == NULL) {  printf("\n12Memória
               insuficiente. Fechando o programa...\n");   exit(1);
                       }
189         if (auxCompressao == NULL) {    printf("\n13Memória
               insuficiente. Fechando o programa...\n");   exit(1);
                       }
190
191         //TAMANHO DAS VARIAVEIS PARA PREENCHER O VETOR ESTENDIDO
192         l = m-1;
193         p = l;
194         r = n + l;
195         u = n;
196
197         for(o=0;o<l;o++){
```

```
198             p--;
199             x2[o] = vetor[p];
200     }
201     for(q=l;q<r;q++){
202             x2[q] = vetor[s];
203             s++;
204     }
205     for(t=r;t<a;t++){
206             u--;
207             x2[t] = vetor[u];
208     }
209
210     //CALCULO DO NÍVEL 1 DO BANCO DE FILTROS
211     b1 = n + m - 1 + l;
212
213     for(w=0;w<b;w++){
214             aux0[w]=filtro(H0,x2,w);
215             aux1[w]=filtro(H1,x2,w);
216             if(w>=l && w<b1){
217                     if(w%2==0){
218                             y0[d1] = aux0[w];
219                             y1[d1] = aux1[w];
220                             d1++;
221                     }
222             }
223     }
224
225     c1 = l;
226     o1 = d + l;
227     e1 = d;
228
229     for(i=0;i<l;i++){
230             c1--;
231             y2[i] = y0[c1];
232     }
233     for(j=l;j<o1;j++){
234             y2[j] = y0[d2];
235             d2++;
```

```
236         }
237         for(k=o1;k<e;k++){
238                 e1--;
239                 y2[k] = y0[e1];
240         }
241
242         //CALCULO DO NÍVEL 2 DO BANCO DE FILTROS
243         j = d + m - 1 + l;
244
245         for(f1=0;f1<f;f1++){
246                 aux00[f1]=filtro(H0,y2,f1);
247                 aux01[f1]=filtro(H1,y2,f1);
248                 if(f1>=l && f1<j){
249                         if(f1%2==0){
250                                 y00[g1] = aux00[f1];
251                                 y01[g1] = aux01[f1];
252                                 g1++;
253                         }
254                 }
255         }
256
257         for(j1=0;j1<h;j1++){
258                 saidaBanco[v] = y00[j1];
259                 v++;
260         }
261
262         for(i1=0;i1<h;i1++){
263                 saidaBanco[v] = y01[i1];
264                 v++;
265         }
266
267         for(a1=0;a1<d;a1++){
268                 saidaBanco[v] = y1[a1];
269                 v++;
270         }
271
272         for(p1=0;p1<tam;p1++){
273                 if(saidaBanco[p1] > 32.29 || saidaBanco[p1] < -32.29)
```

```
                        {
274                             vetorComprimido[p1] = saidaBanco[p1];
275                             auxCompressao[p1] = 1;
276                     }else{
277                             auxCompressao[p1] = 0;
278                             vetorComprimido[p1] = saidaBanco[p1];
279                     }
280             }
281
282             printf("\n\nc = [");
283             for(k1=0;k1<tam;k1++){
284                     printf("        %f",vetorComprimido[k1]);
285             }
286             printf("]");
287
288             printf("\n\nBitmap = [");
289             for(t1=0;t1<tam;t1++){
290                     printf("        %d",auxCompressao[t1]);
291             }
292             printf("]");
293
294             printf("\nfim main");
295             return(0);
296     }
```