Universidade Estadual de Londrina
Centro de Tecnologia e Urbanismo
Departamento de Engenharia Elétrica

Dário Guilherme Toginho

ECG Gerado por Microcontrolador

Londrina
2015

Universidade Estadual de Londrina

Centro de Tecnologia e Urbanismo
Departamento de Engenharia Elétrica

Dário Guilherme Toginho

ECG Gerado por Microcontrolador

Trabalho de Conclusão de Curso orientado pelo Prof. Dr. Aziz Elias Demian Junior intitulado "ECG Gerado por Microcontrolador" e apresentado à Universidade Estadual de Londrina, como parte dos requisitos necessários para a obtenção do Título de Bacharel em Engenharia Elétrica.

Orientador: Prof. Dr. Aziz Elias Demian Junior

Londrina
2015

Ficha Catalográfica

Dário Guilherme Toginho
ECG Gerado por Microcontrolador - Londrina, 2015 - 47 p., 30 cm.
Orientador: Prof. Dr. Aziz Elias Demian Junior
1. eletrocardiograma. 2.simulador. 3. PWM 4. arduino. 5. microcontrolador. I. Universidade Estadual de Londrina. Curso de Engenharia Elétrica. II. ECG Gerado por Microcontrolador.

Dário Guilherme Toginho

ECG Gerado por Microcontrolador

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia Elétrica da Universidade Estadual de Londrina, como requisito parcial para a obtenção do título de Bacharel em Engenharia Elétrica.

Comissão Examinadora

Prof. Dr. Aziz Elias Demian Junior
Universidade Estadual de Londrina
Orientador

Prof. Dr. Ernesto Fernando Ferreyra Ramírez
Universidade Estadual de Londrina

Prof. Dr. Leonimer Flávio de Melo
Universidade Estadual de Londrina

Londrina, 18 de fevereiro de 2016

A todos que acreditaram em meu potencial e me motivaram no meio do caminho.

Agradecimentos

Agradeço a minha família pelos sacrifícios, apoio e amor que recebi ao longo da minha vida. Aos meus amigos que me acompanharam ao longo dessa jornada chamada faculdade, e que me proporcionaram momentos dos quais levarei para sempre. Aos professores da Universidade Estadual de Londrina, por terem nos ensinado não apenas disciplinas do plano de curso, mas também como ser profissionais sérios e competentes.

Agradecimentos especiais ao meu pai, Dário de Oliveira Toginho, por sempre me incentivar a ir mais longe, por estar ao meu lado quando precisei, e por todo o carinho de uma vida. A minha tia, Daglie Margarete Toginho, por ter sido uma segunda mãe para mim, meiga e carinhosa, pelos últimos anos. Ao meu tio, Dari de Oliveira Toginho, por ter me ajudado e apoiado não só antes da minha graduação como durante ela.

Ao professor da Universidade Estadual de Londrina, Aziz Elias Demian Junior, pela orientação deste trabalho, por acreditar em mim, e por todo apoio ao longo do último ano.

Aos amigos da faculdade, Fernando, Gabriel, Fernanda, Jéssica, Luiz Ricardo, Jefferson, Daniel Galbes, Marco Aurélio e Edgar, por termos dividido uma sala de aula juntos, dividido gargalhadas, e por ter dividido com vocês essa jornada dos últimos anos. Aos amigos que me ajudaram quando eu estava em uma fase ruim da minha vida, Ana, Andréia, Andressa, Arthur, Jéssica e Natália, por terem me ajudado a sair do poço de amargura onde eu estava, a seguir em frente com a vida, e proporcionar as melhores risadas que tive ao realizar esse trabalho.

"Ninguém vai bater tão forte quanto a vida. Mas não se trata do quanto você bate de volta. Se trata do quanto você aguenta apanhar e continuar seguindo em frente. É assim que se conquista a vitória"

(Rocky Balboa)

Dário Guilherme Toginho. 2015. 47 p. Trabalho de Conclusão de Curso em Engenharia Elétrica - Universidade Estadual de Londrina, Londrina.

Resumo

O uso de eletrocardiógrafos na medicina atual é enorme. Durante o desenvolvimento de protótipos de tais aparelhos, por questões éticas e morais, não se utilizam cobaias humanas. Para contornar isso, há circuitos que simulam os sinais elétricos do coração. Neste trabalho, os sinais elétricos do coração foram gerados pela filtragem de sinais PWM de um microcontrolador. A amostra do ECG foi obtida do banco de dados do PTB (Physikalisch-Techchnische Bundesanstalt), e então foi modulada pelo MATLAB^{TM} para gerar o sinal PWM. As razões cíclicas dos PWMs correspondentes as derivações unipolares foram utilizada no Arduino^{TM} como argumento de saída das portas PWM. As derivações unipolares foram geradas simultaneamente, e a partir delas é possível obter as derivações bipolares. O sistema foi simulado no Proteus^{TM} e foram realizados testes em bancada com osciloscópio e em um módulo de obtenção de ECG. Os sinais foram reproduzidos com sucesso e mostraram-se coerentes com a amostra original.

Palavras-Chave: 1. eletrocardiograma. 2.simulador. 3. PWM 4. arduino. 5. microcontrolador.

ECG Generated by Microcontroller. 2015. 47 p. Monograph in Engenharia Elétrica - Universidade Estadual de Londrina, Londrina.

Abstract

The use of electrocardiographes nowadays in medicine is huge. During the development of prototypes of such devices, for ethical and moral matters, humans are not used as test subjects. To come around this, there are circuits that simulate electrical signals of the heart. In this work, the electrical signals of the heart were generated by filtering a microcontroller's PWM signals. The ECG sample was obtained from the PTB (Physikalisch-Techchnische Bundesanstalt) database, then modulated through MATLAB^{TM} to generate the PWM signal. The duty cycles of the PWMs corresponding to the unipolar leads were used in the Arduino^{TM} as an argument for the PWM output ports. The unipolar leads were generated simultaneously, and from them it's possible to obtain the bipolar leads. The system was simulated on Proteus^{TM} and tests were carried out in the workbench with oscilloscope and an ECG obtaining module. The signals were successfully reproduced and showed them selves coherent with the original sample.

Key-words: 1. electrocardiography. 2.simulator. 3. PWM. 4. arduino. 5. microcontroller.

Lista de ilustrações

Lista de Siglas e Abreviaturas

A/D	*Analógico-Digital*
avF	*Augmented Vector Foot* - Derivação Aumentada Pé
avL	*Augmented Vector Left* - Derivação Aumentada Esquerda
avR	*Augmented Vector Right* - Derivação Aumentada Direita
D/A	Digital-Analógico
ECG	Eletrocardiograma
PWM	*Pulse Width Modulation* - Modulação por Largurade Pulso

Lista de Símbolos e Notações

D	Razão cíclica do sinal PWM
f_{PWM}	Frequência do sinal PWM
T_{OFF}	Período em que o sinal PWM está em nível baixo
T_{ON}	Período em que o sinal PWM está em nível alto
T_{PWM}	Período do sinal PWM

Palavras em *itálico* são empregadas para identificar termos de língua inglesa não traduzidos.

Sumário

1 Introdução

1.1 Histórico do ECG

A eletrocardiografia teve seu invento entre o final do século XVIII e início do século XIX. Através dela foi possível compreender mecanismos e desenvolver tratamentos específicos para arritmias. Willem Einthoven é considerado o pai da eletrocardiografia por ter desenvolvido o método e famoso por ter criado o triângulo de Einthoven, que representa as três principais derivações. Einthoven registrou o primeiro ECG na Europa em 1892 usando o eletrômetro de Lippmann, em 1902 Einthoven realizou o primeiro registro eletrocardiográfico de um ser humano, utilizando um galvanômetro modificado(PORTAL-EDUCAÇÃO, 2013).

1.2 Objetivos do trabalho

Implantar um sistema que dispense o uso de seres vivos em testes e calibração de eletrocardiógrafos utilizando filtragem analógica de sinal PWM gerado por microcontrolador.

1.3 Justificativa do trabalho

Ao desenvolver ou calibrar um eletrocardiógrafo, testes em seres vivos levantam muitas questões, tanto éticas e morais quanto burocráticas. Um módulo que simule os sinais elétricos de um coração humano contornaria tais problemas, facilitando tanto o desenvolvimento de protótipos de eletrocardiógrafos quanto a calibração de aparelhos já prontos.

Atualmente há poucos sistemas que reproduzam sinais de ECG através de filtragem de sinal PWM. Um trabalho que utiliza filtragem de sinal PWM para conversão D/A é de (CANER; ENGIN; ENGIN, 2008), porém ele utiliza o módulo conectado a um computador por comunicação RS232 e UART (*Universal Asynchronous Receiver Transmitter*). Um simulador feito com microcontrolador e conversão D/A por rede R2R foi pode ser encontrado em (DRAGHICIU; CRETIU, 2013). Outra forma utilizada para reproduzir um sinal de ECG é através da resolução numérica, por método de Runge-Kutta, de um conjunto de equações diferenciais que representam o ECG, como feito em (WEI et al., 2012). Um exemplo de simulador de ECG com regulagem de vários parâmetros das ondas pode ser visto em (CHANG; TAIA, 2006), porém ele se limita somente a derivações bipolares vistas individualmente.

1.4 Materiais e métodos

Foi obtida uma amostra de ECG de um paciente anônimo, fornecida pelo *Physikalisch-Technische Bundesanstalt*, que é o instituto de pesos e medidas da Alemanha. A amostra foi importada através do *software* MATLAB^{TM}, e através dele foi normalizada, e realizada a modulação para obter um sinal digital do tipo PWM. Com o MATLAB^{TM} ainda, foram obtidos os valores de razão cíclica do PWM e exportados para um arquivo de texto. Esses valores foram inseridos no código em linguagem C do Arduino^{TM}, o microcontrolador escolhido, e utilizados nas saídas PWM disponíveis. Nessas saídas então foram cascateados filtros passa-baixas de 4ª ordem, utilizando resistores e capacitores. Na saída de cada filtro havia um *buffer* seguido de uma chave seletora. A chave possuía duas posições sendo uma com a saída do próprio *buffer* e a outra com a saída do *buffer* atenuada por um divisor de tensão, de forma que sua amplitude se encontrasse na faixa entre 10 mV e 25mV. Os testes iniciais foram realizados em simulações no *software* Proteus^{TM}, e em seguida foram realizados testes em bancada com osciloscópio.

Foram utilizados um kit de desenvolvimento Arduino^{TM} Uno, resistores e capacitores para filtragem, amplificador operacional LM741 para o *buffer*, chave seletora, e os *softwares* MATLAB^{TM}, Proteus^{TM} e o compilador de linguagem C do Arduino^{TM}.

1.5 Estrutura do trabalho

No capítulo 2 deste trabalho é apresentada a fundamentação teórica na qual ele se baseou. Nele há um introdução ao eletrocardiograma, explicação sobre PWM e conversão A/D através de PWM, teoria de filtro passa-baixas e conversão D/A utilizando filtro passa-baixas em sinal PWM, interpolação de sinais é uma breve introdução ao Arduino^{TM}.

No capítulo 3 é abordada a metodologia utilizada para realização deste trabalho. Nele há seções detalhando os materiais e equipamentos utilizados, a forma como a amostra do ECG foi obtida, a configuração dos *timers* do Arduino^{TM}, uma descrição sucinta de como o sinal foi modulado, para gerar o PWM, utilizando o MATLAB^{TM}, e também como foi feita a filtragem do PWM. Ao final do capítulo é apresentado o circuito utilizado.

No capítulo 4 são apresentados os resultados do trabalho. Nele temos as simulações do circuito, as ondas visualizadas no osciloscópio na bancada, e também o sinal em um módulo de obtenção de eletrocardiografia.

No capítulo 5 são apresentadas a discussão final e conclusões referentes ao trabalho realizado, e sugestões de melhora para ele.

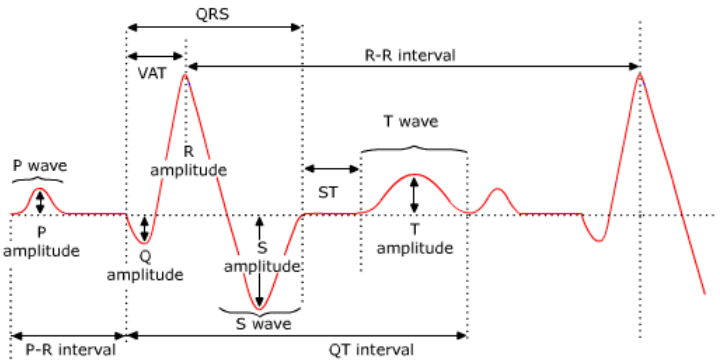
2 Fundamentaçao Teórica

2.1 Introdução ao Eletrocardiograma

O ECG é o registro da atividade elétrica do coração. Os eventos elétricos resultantes do ciclo sístole/diástole se propagam através do tórax. Sua seqüência é medida na superfície do corpo através de eletrodos. O ECG representa a somatória de todas as atividades elétricas que ocorrem a cada instante do ciclo cardíaco, e é produzido por um eletrocardiógrafo ou por um monitor cardíaco.

A atividade elétrica do coração humano apresenta amplitude de até algumas dezenas de milivolts e pode ser detectada na superfície do corpo do paciente.

Figura 1 – Eletrocardiograma típico, mostrando as ondas P, T e o complexo QRS.

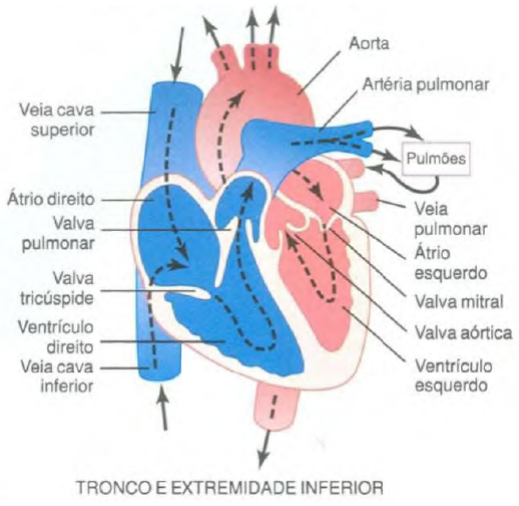


Fonte: (UNICAMP/UNIFESP, 2016)

Através do ECG é possível monitorar outros parâmetros vitais, tais como taxa respiratória, pressão sanguínea, débito cardíaco e oxigenação do sangue, entre outros.

Podemos ver as ondas que caracterizam um ECG na figura 1. A onda P representa a despolarização dos átrios, logo antes de sua contração. No intervalo entre as ondas P e Q, ocorre a passagem de sangue dos átrios para os ventrículos através da valva mitral e da valva tricúspide, vistas na figura 2. O complexo QRS representa a despolarização ventricular antes de sua contração. Simultaneamente ocorre a repolarização dos átrios, porém sua amplitude é desprezível comparada a despolarização ventricular. A onda T representa a repolarização dos ventrículos (GUYTON; HALL, 2006).

Figura 2 – Estrutura do coração.



Fonte: (GUYTON; HALL, 2006)

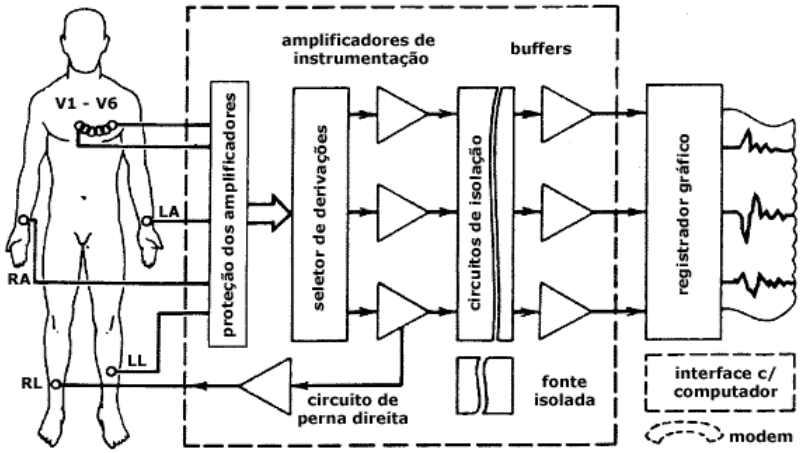
Obtido através do eletrocardiógrafo ou do monitor cardíaco, o ECG é usado para determinar frequência cardíaca e duração de cada elemento (P, QRS, T, P-R, S-T), determinar a frequência respiratória (que modula a frequência cardíaca), e determinar o eixo elétrico do coração.

Toda captação de biopotenciais é sujeita a interferências de outros sinais do ambiente ou do próprio corpo do paciente, sendo essas interferências de diversas naturezas. Fontes biológicas e artefatos são potenciais de pele, onde a interface pele-gel-eletrodo pode acumular potenciais superiores a 25 mV. Esses podem ser reduzidos pela raspagem ou punção da pele, que elimina sua camada morta superficial e, no caso da punção, ultrapassa a barreira de outras camadas da pele, aumentando o contato elétrico. Os potenciais CC também são eliminados por filtros passa-altas.

Artefatos de movimento são sinais produzidos pelo movimento relativo entre pele e eletrodo, com modificação da linha de base ou presença de ruído no traçado, dificultando sua interpretação.

Ruído muscular é quando os potenciais de ação da musculatura esquelética têm a mesma faixa de amplitude do ECG, mas sua faixa de frequência é maior. Podem ser eliminados com filtros passa-baixas, através da colocação adequada dos eletrodos e do repouso do paciente.

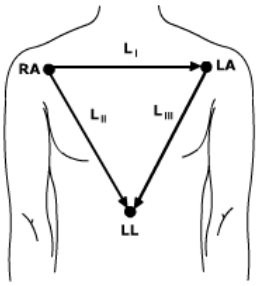
Figura 3 – Diagrama em blocos de um ECG diagnóstico.



Fonte: (UNICAMP/UNIFESP, 2016)

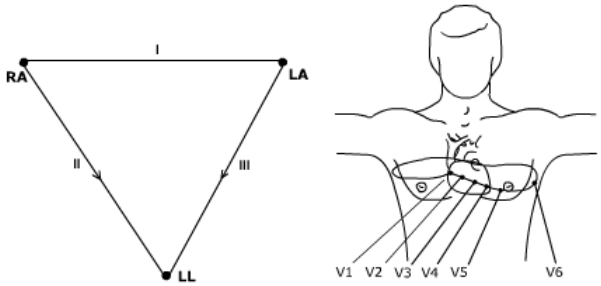
Um sistema típico de eletrocardiografia é composto de 12 derivações, o que significa o uso de 12 eletrodos distribuídos pela superfície corporal do paciente, como vistos na figura 3. Os eletrodos correspondentes ao equilátero de Einthoven (derivações I, II e III), visto na figura 4, são geralmente colocados sobre os pulsos (RA e LA) e no tornozelo esquerdo. Os 9 eletrodos restantes, vistos na figura 5, são colocados 1 em cada braço, 1 na perna esquerda, 6 no peito e 1 na perna direita, que é usado para reduzir interferências elétricas.

Figura 4 – Triângulo de Einthoven.



Fonte: (UNICAMP/UNIFESP, 2016)

Figura 5 – Colocação dos 9 eletrodos usados para obter as 12 derivações clássicas.

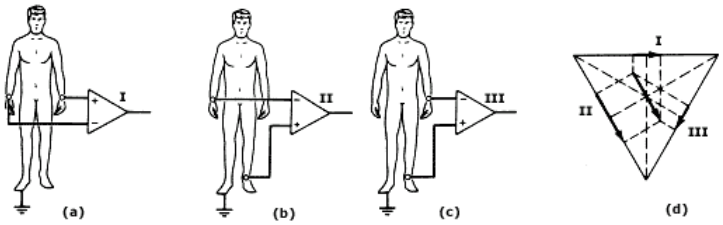


Fonte: (UNICAMP/UNIFESP, 2016)

Os eletrodos LL, LA e RA são conectados à rede de resistores conhecida como central de Wilson, a partir da qual se obtém as derivações dos membros I, II, III e as aumentadas aVL, aVR e aVF. O isolamento elétrico deve fornecer proteção ao pacientes contra riscos de choque elétrico.

O sistema padrão de 12 derivações inclui 3 colocações diferentes de eletrodos: derivação bipolar, derivação aumentada e derivação pré-cordial.

Figura 6 – Derivação bipolar: (a) derivação I; (b) derivação II; (c) derivação III; (d) técnica para derivar o vetor cardíaco (no centro do triângulo), a partir das projeções geométricas dos vetores de membros.



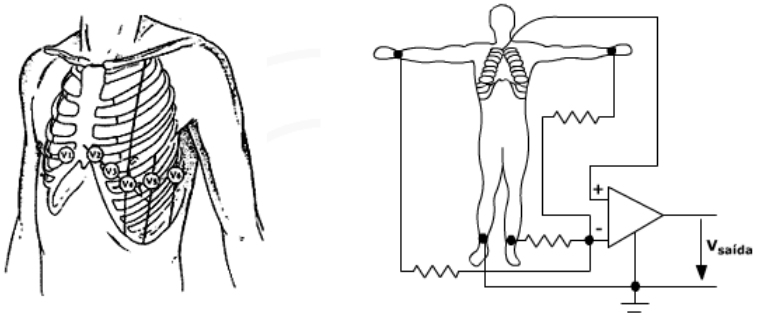
Fonte: (UNICAMP/UNIFESP, 2016)

As derivações bipolares podem ser obtidas através de operações das derivações unipolares, como visto na figura 6, sendo:

- I = avL - avR;
- II = avF - avR;
- III = avF - avL;

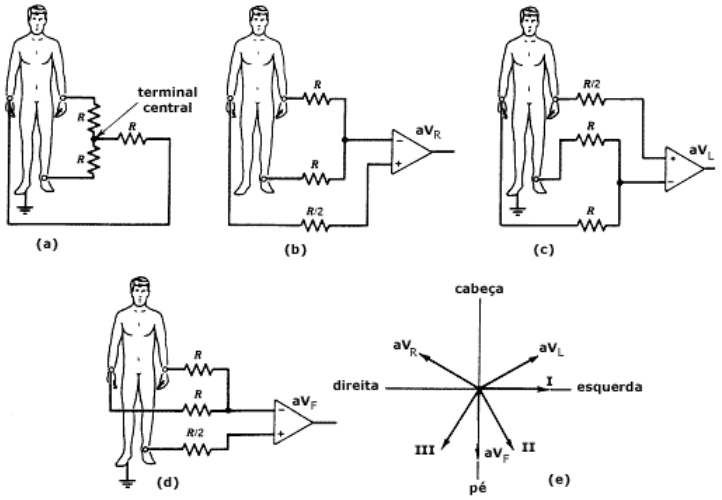
O terminal central de Wilson, visto na figura 7, é um ponto com potencial elétrico zero no corpo. A partir dele podemos obter as derivações unipolares aumentadas.

Figura 8 – Derivações pré-cordiais: o eletrodo ligado à entrada não inversora do amplificador é o eletrodo explorador, colocado em uma das posições pré-cordiais, um de cada vez.



Fonte: (UNICAMP/UNIFESP, 2016)

Figura 7 – (a) Central de Wilson; (b) a (d) derivações aumentadas aVR, aVL e aVF; (e) relações entre os vetores das derivações de membros e as derivações aumentadas no plano frontal.



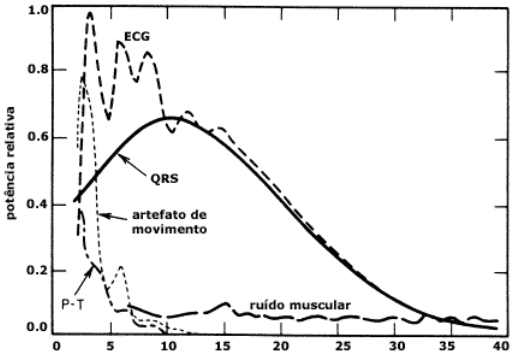
Fonte: (UNICAMP/UNIFESP, 2016)

As derivações pré-cordiais, vistas na figura 8, tem como finalidade observar as diversas paredes ventriculares.

A faixa de frequências utilizada para filtragem posterior a aquisição do sinal deve eliminar os ruídos ambientais e biológicos sem distorcer o sinal de ECG. O espectro dos sinais captados pelos eletrodos podem ser vistos na figura 9. Valores típicos para essa faixa são:

- 0,5 a 40 Hz para monitoração;
- 0,01 a 150 Hz para ECG diagnóstico;

Figura 9 – Composição espectral dos sinais captados pelos eletrodos de ECG e composições espectrais de artefatos de movimento e ruído muscular - sinais que interferem no registro do ECG.

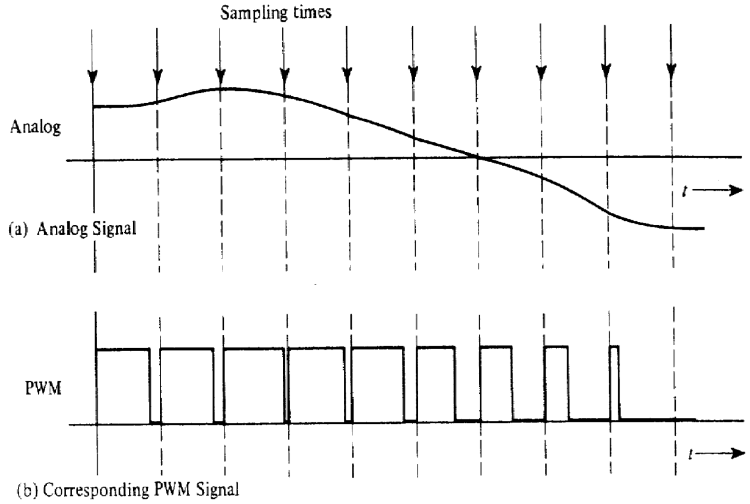


Fonte: (UNICAMP/UNIFESP, 2016)

2.2 PWM - Modulação por Largura de Pulso

Modulação consiste em alterar características de um onda (amplitude, frequência, fase), de forma que tais mudanças representem algum tipo de informação portada por tal sinal. Uma forma de modulação existente é a PWM, que significa modulação por largura de pulso. Podemos ver na figura 10 que o sinal PWM possui amplitude e frequência fixas, porém o tempo em que o sinal fica em nível alto durante o tempo total do período, é proporcional ao sinal analógico. (COUCH II, 2002).

Figura 10 – (a)Sinal analógico e (b)Sinal PWM correspondente.



Fonte: (COUCH II, 2002).

O tempo do sinal em nível alto dividido pelo tempo do período dele, é definido por

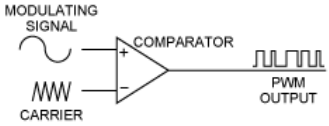
razão cíclica. Nesta monografia, razão cíclica será definida pela letra D.

$$D = \frac{T_{ON}}{T_{PWM}} \tag{2.1}$$

2.3 Conversão Analógica Digital por PWM

Uma forma de converter um sinal analógico para digital é através de PWM. As componentes essenciais de um sinal PWM são a frequência (f_{PWM}) e a razão cíclica (D).

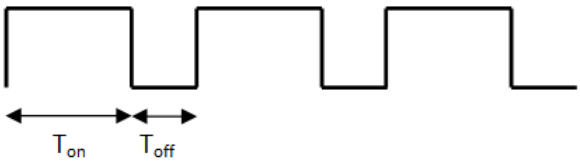
Figura 11 – Circuito comparador que geral um sinal PWM.



Fonte: Página da Maxim Integrated[1].

Uma forma de realizar tal conversão é através de um circuito comparador, com o sinal analógico de interesse, e uma onda triangular como entradas, como na figura 11. O tempo em que o sinal analógico for maior que a onda triangular, será o tempo T_{ON} do sinal digital PWM. Os períodos relevantes de um PWM podem ser vistos na figura 12, sendo $T_{PWM} = T_{ON} + T_{OFF}$. Para contornar a possibilidade de que pulsos se tornem um sinal puramente DC, utiliza-se a onda triangular com amplitude um pouco maior do que o pico do sinal analógico (10% a 20% do valor de pico). Embora conceitualmente a amplitude da onda triangular pode ser o valor de pico do sinal modulante, na prática é utilizado uma amplitude um pouco maior por segurança.

Figura 12 – Sinal PWM com T_{ON} e T_{OFF} indicados, sendo $T_{PWM} = T_{ON} + T_{OFF}$.



Fonte: Página da BVSYSTEMS[2].

Com esse procedimento, o sinal de interesse foi modulado através da largura variável dos pulsos da onda quadrada (MALVINO, 1995).

[1] Disponível em <https://www.maximintegrated.com/en/app-notes/index.mvp/id/1860> Acesso em fev. 2016.

[2] Disponível em <http://www.bvsystems.be/k8055hardwareTutorial3.php> Acesso em fev. 2016.

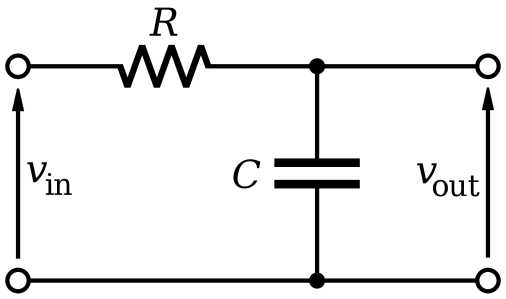
2.4 Filtro Passa-Baixas

Filtros são topologias de circuito usadas em grandes campos da engenharia elétrica, com aplicações para áreas de controle, eletrônica, engenharia biomédica, processamento de sinais, telecomunicações, entre outras. Sua função é permitir a passagem de certas frequências que compõem o sinal que será filtrado. O filtro do tipo passa-baixas permite a passagem das frequências abaixo de sua frequência de corte.

Um filtro pode ser tanto digital quanto analógico. A filtragem digital normalmente é realizada por DSPs (*Digital Signal Processors*), e a analógica utiliza resistores, capacitores, indutores e amplificadores operacionais.

A topologia mais simples de filtro passa-baixas é o filtro RC de primeira ordem, que é basicamente um divisor de tensão entre um resistor e um capacitor, como visto na figura 13.

Figura 13 – Filtro passa-baixas RC de primeira ordem.



Fonte: Artigo sobre Filtro Passa-Baixas na pagina inglesa da Wikipedia[3].

Para obtermos a função de transferência do filtro, dada por $\frac{v_{out}}{v_{in}}$, é feito um divisor de tensão entre os dois componentes e o resultado é dividido por v_{in}.

$$\frac{v_{out}}{v_{in}} = v_{in}\frac{X_C}{R+X_C}\frac{1}{v_{in}} = \frac{X_C}{R+X_C} \tag{2.2}$$

Onde X_C é a reatância do capacitor, dada por:

$$X_C = -\frac{j}{2\pi fC} \tag{2.3}$$

Sendo $j = \sqrt{-1}$, f a frequência em que o circuito está operando e C a capacitância. Substituindo na equação da função de transferência:

$$\frac{v_{out}}{v_{in}} = \frac{-\frac{j}{2\pi fC}}{R-\frac{j}{2\pi fC}} \tag{2.4}$$

Em seguida, multiplica-se numerador e denominador por $-\frac{2\pi fC}{j}$:

$$\frac{v_{out}}{v_{in}} = \frac{1}{1+j2\pi fRC} \tag{2.5}$$

[3] Disponível em <https://en.wikipedia.org/wiki/Low-pass_filter> Acesso em fev. 2016.

Calculando o módulo dessa equação, temos que:

$$\left|\frac{v_{out}}{v_{in}}\right| = \frac{1}{\sqrt{1+(2\pi fRC)^2}} \tag{2.6}$$

A frequência de corte e definida quando a função de transferência é igual a $\frac{1}{\sqrt{2}}$, ou seja, quando $f = \frac{1}{2\pi RC}$. Essa atenuação e o equivalente a 3 dB. Há também um decaimento de 20 db/década.

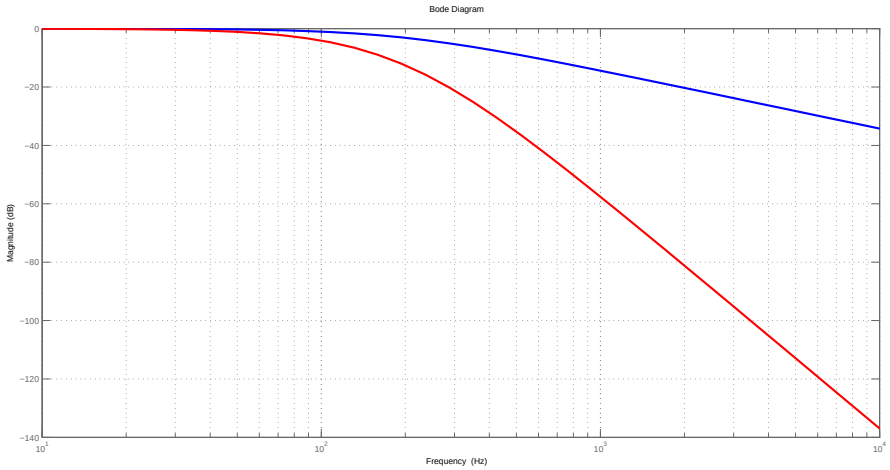
Um cascateamento no domínio da frequência equivale a uma multiplicação das funções de transferência, de forma que a função de transferência do filtro de 4^a ordem é dado por:

$$\frac{v_{out}}{v_{in\ 4^a\ \text{ordem}}} = \left(\frac{1}{1+sRC}\right)^4 = \frac{1}{1+4sRC+6(sRC)^2+4(sRC)^3+(sRC)^4} \tag{2.7}$$

Onde $s = j2\pi f$.

Quando o circuito é feito em cascata, a ordem do filtro aumenta, mantendo a frequência de corte, porém aumentando a atenuação e a queda em dB/década. Ou seja, um filtro de 4^a ordem possui atenuação de 12 dB na frequência de corte e decaimento de 80 dB/década.

Figura 14 – Em azul o diagrama de bode do filtro de 1^a ordem e em vermelho o de 4^a.

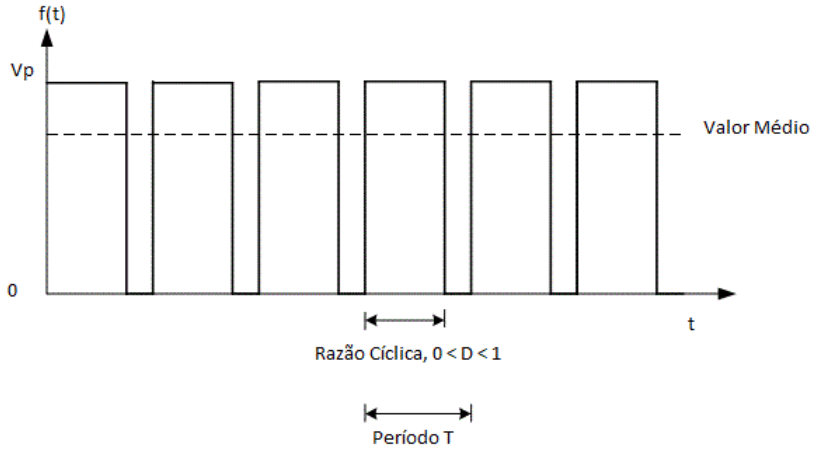


Fonte: Elaborada pelo autor utilizando MATLAB^{TM}.

2.5 Conversão Digital Analógica do sinal PWM através de FPB

Uma forma simples de realizar a conversão D/A de um sinal PWM é através de filtros passa baixas. Na figura 15, podemos ver um sinal PWM comum, com parâmetros genéricos para razão cíclica, período e valor de pico. Através desse sinal, é possível deduzir a série de Fourier do sinal PWM.

Figura 15 – Típico sinal PWM.



Fonte: Página TechTeach (Figura editada)[4].

Todo sinal periódico pode ser representado como uma série de Fourier. Seja um sinal $f(t)$ de período T, temos abaixo sua representação em série de Fourier:

$$f(t) = a_0 + \sum_{n=1}^{\infty} \left(a_n cos\left(\frac{2\pi nt}{T}\right) + b_n sin\left(\frac{2\pi nt}{T}\right)\right) \tag{2.8}$$

Sendo:

$$a_0 = \frac{1}{T}\int_0^T f(t)dt \tag{2.9}$$

$$a_n = \frac{2}{T}\int_T^T f(t) cos\left(\frac{2\pi nt}{T}\right) dt \tag{2.10}$$

$$b_n = \frac{2}{T}\int_0^T f(t) sin\left(\frac{2\pi nt}{T}\right) dt \tag{2.11}$$

Ao calcular os termos da série de Fourier para um sinal PWM, obtemos:

$$f(t) = D.V_p + \sum_{n=1}^{\infty} \frac{V_p}{\pi n}\left(sin(2\pi nD) cos\left(\frac{2\pi nt}{T_{PWM}}\right) + (1 - cos(2\pi nD)).sin\left(\frac{2\pi nt}{T_{PWM}}\right)\right) \tag{2.12}$$

Idealmente, ao filtrar o sinal com uma frequência de corte $\frac{1}{T_{PWM}}$, somente a componente contínua $D.Vp$ continuará presente. E conforme a razão cíclica do PWM varia, a tensão da componente DC também varia. Como a razão cíclica do PWM filtrado varia em proporção ao sinal analógico original, a filtragem resultará, idealmente, no sinal original.

[4] Disponível em <http://techteach.no/simview/pwm_control/index.php> Acesso em fev. 2016.

2.6 Interpolação

Interpolação é o processo de elevar a taxa de amostragem de um sinal já amostrado. Seja um sinal continuo no tempo representado por $x_c(t)$, ele pode ser representado como um sinal discreto no tempo como uma sequencia de amostras dado por:

$$x[n] = x_c(nT_s) \tag{2.13}$$

Sendo $T_s = \frac{1}{f_s}$ o período de amostragem e n uma sequência de números inteiro positivo.

Aumentando a taxa de amostragem do sinal por um fator L temos:

$$x_i[n] = x\left[\frac{n}{L}\right] = x_c\left(\frac{nT_s}{L}\right) \tag{2.14}$$

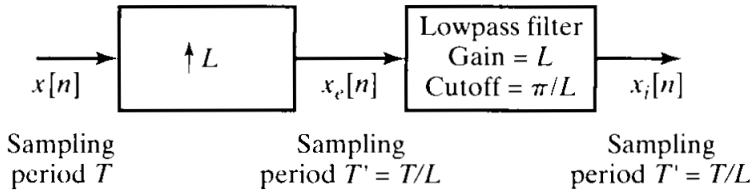
O sistema utilizado para obter x_i através de $x[n]$ pode ser visto na figura 16. O bloco onde está escrito "$\uparrow L$", é um sistema chamado *expansor de taxa de amostragem*. A sua saída é dada por:

$$x_e[n] = \begin{cases} x[\frac{n}{L}], & n = 0, \pm L, \pm 2L, ..., \\ 0, & \text{c.c.}, \end{cases} \tag{2.15}$$

Sendo L um número inteiro positivo. Outra forma de representar $x_e[n]$ é através de uma série de impulsos multiplicados por $x[n]$.

$$x_e[n] = \sum_{k=-\infty}^{\infty} x[k]\delta[n - kL] \tag{2.16}$$

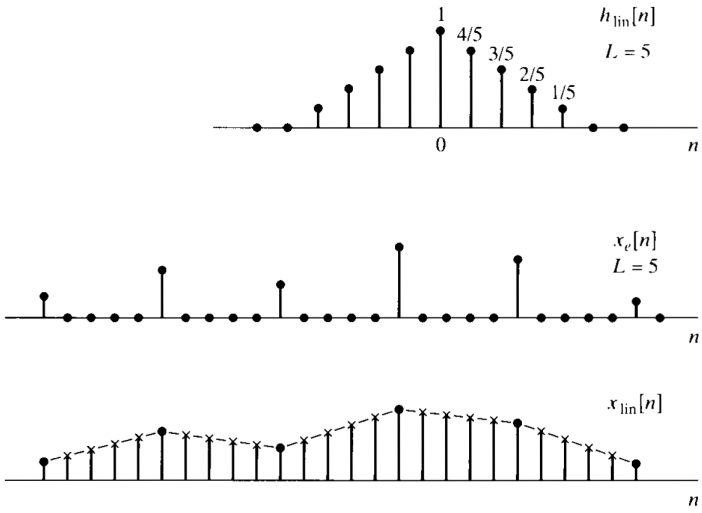
Figura 16 – Sistema utilizado para interpolação.



Fonte: (OPPENHEIM; SCHAFER; BUCK, 1999).

Ou seja, o sinal após o expansor se torna mais longo ao adicionar uma quantidade adequada de zeros entre as amostras anteriores. Pode-se observar um exemplo na figura 17.

Figura 17 – Exemplo de interpolação com $L = 5$.



Fonte: (OPPENHEIM; SCHAFER; BUCK, 1999).

O sinal $x_e[n]$ no domínio da frequência é obtido pela transformada de Fourier no tempo discreto, dada por:

$$X(e^{j\omega}) = \sum_{n=-\infty}^{\infty} x[n]e^{-j\omega n} \tag{2.17}$$

Aplicando a transformada em $x_e[n]$:

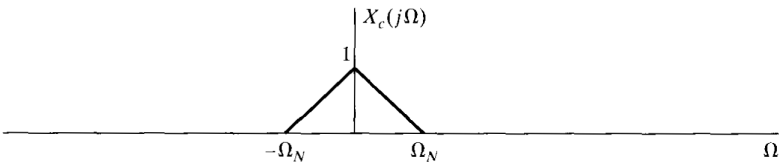
$$X_e(e^{j\omega}) = \sum_{n=-\infty}^{\infty} \left(\sum_{k=-\infty}^{\infty} x[k]\delta[n - kL] \right) e^{-j\omega n} \tag{2.18}$$

$$X_e(e^{j\omega}) = \sum_{n=-\infty}^{\infty} x[k]e^{-j\omega Lk} = X(e^{j\omega L}) \tag{2.19}$$

Vê-se que $X_e(e^{j\omega})$ é uma mudança de escala na frequência do sinal original $x[n]$.

Após o expansor, o sinal e aplicado a um filtro passa-baixas com frequência de corte igual a $\frac{\pi}{L}$, sendo $\frac{\pi}{T} = \Omega_N$ onde, como visto na figura 18, Ω_N é a largura de banda de $x_c(t)$.

Figura 18 – Espectro do sinal $x_c(t)$.



Fonte: (OPPENHEIM; SCHAFER; BUCK, 1999).

A resposta ao impulso do filtro utilizado é:

$$h_i[n] = \frac{sin(\pi n/L)}{\pi nL} \tag{2.20}$$

E possui as seguintes propriedades:

- $h_i[0] = 1$,
- $h_i[n] = 0, \quad n = \pm L, \pm 2L, ...$

Convoluindo $x_e[n]$ com $h_i[n]$, obtemos a saída do sistema:

$$x_i[n] = \sum_{k=-\infty}^{\infty} x[k] \frac{sin[\pi(n - kL)/L]}{\pi(n - kL)/L} \tag{2.21}$$

Para um filtro passa-baixas de interpolação ideal, nós temos:

$$x_i[n] = x[n/L] = x_c(nT/L), \quad n = 0, \pm L, \pm 2L, ... \tag{2.22}$$

Ou seja, a quantidade de amostras do sinal é aumentada(OPPENHEIM; SCHAFER; BUCK, 1999).

2.7 Arduino^{TM}

Arduino^{TM} é uma plataforma de código aberto para desenvolvimento em eletrônica e programação. Por ser código aberto, torno-se muito popular no mundo inteiro, sendo utilizado por estudantes, artistas, entusiastas e ate mesmo profissionais(ARDUINO, 2016c).

O modelo utilizado neste trabalho é o Arduino^{TM} UNO. Esse kit utiliza um microcontrolador ATmega328P, possui 14 entradas/saídas digitais, sendo que 6 delas podem ser utilizadas como saídas PWM. Possui 6 entradas analógicas, um cristala de 16 MHz, conexão USB, entrada de alimentação, pinos de protocolo de comunicação ICSP (*In-Circuit Serial Programming*) e botão de *reset*(ARDUINO, 2016b).

Seu uso é muito simples, necessitando apenas conectar a placa em um computador através de cabo USB, e utilizar o *software* disponibilizado no site (<https://www.arduino.cc/en/Main/Software>).

3 Metodologia

3.1 Materiais, equipamentos e *softwares* utilizados

Neste trabalho foram utilizados resistores e capacitores para montagem do filtro passa-baixas, resistores para divisor de tensão, amplificadores operacionais como *buffers* para isolação na saída do filtro, chaves seletoras de duas posições para selecionar entre amplitude mais elevada para medição no osciloscópio e baixa amplitude para uso em eletrocardiógrafo, kit de desenvolvimento Arduino^{TM} UNO, Osciloscópio Tektronix TDS 1001C-30EDU para testes, os *softwares* Proteus^{TM} para simulação do circuito, MATLAB^{TM} para processamento do sinal antes do microcontrolador, e o compilador Arduino^{TM} 1.0.5.

3.2 Amostragem do sinal

Para gerar o sinal de ECG, foram utilizadas amostras do banco de dados *Physikalisch-Technische Bundesanstalt* (PTB)[1], que é o Instituto Alemão de pesos e medidas. O banco contem dados de ECG de diversos pacientes e pode ser exportado como matriz de dados para o MATLAB^{TM}. Cada linha da matriz contém aproximadamente 10 períodos de amostras referentes as derivações utilizadas. Foi realizado um processo de interpolação nos dados para elevar a frequência de amostragem, e utilizado somente os primeiros elementos do vetor contendo o primeiro batimento cardíaco. O processo de interpolação realizado pela função **interp** do MATLAB (MATHWORKS, 2016) pode ser vista na seção 2.6.

Foi escolhido um paciente cujo diagnóstico não apresentou cardiopatias, de modo que fosse uma amostra de controle. Os dados do paciente escolhido são:

- Sexo: Feminino;
- Idade: 67 anos;
- Data de aquisição do ECG: 28/02/1997;

Em seguida, essas amostras foram comparadas com uma onda triangular, e a saída dessa comparação gerou o sinal modulado (PWM). Utilizando o MATLAB^{TM} ainda, foram obtidas as razões cíclicas de cada período do sinal modulado. O código foi escrito de tal forma que 255 represente 100% de razão cíclica, mantendo o mesmo padrão de razão cíclica do PWM do Arduino^{TM}.

Com isso, foi obtida uma matriz contendo as razões cíclicas de todas as derivações utilizadas.

[1] <http://www.physionet.org/cgi-bin/atm/ATM> DATABASE:*PTB Diagnostic ECG Database (ptbdb)*, Record: *patient267/s0504_re*

3.3 Configuração dos *timers*

Os três *timers* disponíveis, *timer0*, *timer1* e *timer2* possuem frequências de PWM diferentes entre si (2 *timers* de 31250 Hz e 1 *timer* de 62500 Hz), mas é possível alterar tais frequências utilizando divisores (1, 8, 64, 256 e 1024 para o *timer0* e *timer1*, e 1, 8, 32, 64, 128, 256 e 1024 para o *timer2*.

Os *timers* escolhidos foram o *timer0* e o *timer1*, ambos com o divisor de frequência 64, de forma que obtemos $f_{PWM} = 520,8333Hz$.

3.4 Geração e filtragem do sinal PWM

Com a matriz de razão cíclica gerada pelo MATLAB^{TM}, foi feito um código que a percorresse e utilizasse os valores de D como argumento de saída dos pinos PWM. Esse código altera a frequência de amostragem dos sinais importados, de forma que fiquem elevados o suficiente para gerar o PWM (256 elementos a cada período do PWM), elimina as amostras seguintes a primeira de todas as derivações, e normaliza suas amplitudes. Com a frequência de amostragem alterada, foi gerada uma onda triangular com a frequência do PWM e amplitude duas vezes maior que a dos sinais importados. Com os sinais já normalizados, eles foram comparados com a onda triangular, elemento a elemento, gerando uma saída para cada derivação, em forma de uma onda quadrada cuja razão cíclica varia com o tempo.

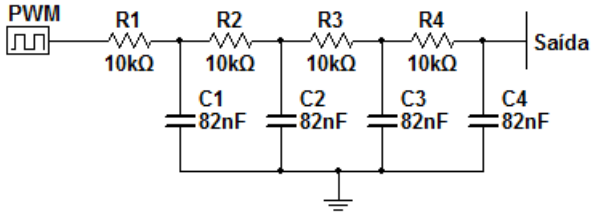
A parte seguinte do código tem como objetivo obter as razões cíclicas das ondas quadradas geradas.

```
for i=1:length(ecg(1,:)) %Laço percorrendo o sinal de interesse a ser modulado;
    if auxcont == 1  %Variável auxilar utilizada para contagem no laço;
        if saida(1,i) == 1
            contagem = contagem + 1; %Variável de contagem de elementos em nível
        else                                                    %alto;
            dutycycle_i(end+1) = contagem; %Assim que é encontrado um nível baixo,
            contagem = 0;                  %significa a contagem é encerrada, e se
            auxcont = 0;                   %inicia novamente no próximo período do
        end %PWM;
    else
        if saida(1,i) == 0 %Caso o sinal de interesse tenha nível baixo, ele man-
            contagem = 1;  %tem a contagem igual a 1. Senão, ela é incrementada,
        else               %e a variável auxiliar muda para 1, retornando a con-
            contagem = contagem + 1; %dição anterior do laço.
            auxcont = 1;
        end
    end
end

dutycycle_i = dutycycle_i(dutycycle_i ~= 0); %Código para normalizar, eliminando
                                             %os elementos diferentes de 0;
```

Foram utilizados laços em cada onda para contar o número de elementos em nível alto a cada período de PWM, de forma que a razão cíclica máxima fosse 255, e a mínima 0, mantendo coerência com o PWM do Arduino^{TM}. Com as razões cíclicas obtidas, foram gerados vetores para cada derivação contendo suas respectivas. Em seguida, foram utilizados comandos para exportar tais dados em arquivo de texto. Foi possível exportar separando cada elemento por "**,**"facilitando para implementar no microcontrolador.

O Arduino^{TM} possui uma função em linguagem C chamada **analogWrite(pin,value)** (ARDUINO, 2016a). Seus argumentos de entrada são o pino de PWM que será utilizado, e a razão cíclica do PWM, indo de 0 até 255. Os dados do arquivo de texto foram copiados para o código em linguagem C, e utilizados como argumento **value** da função.

Figura 19 – Circuito de filtragem utilizado.



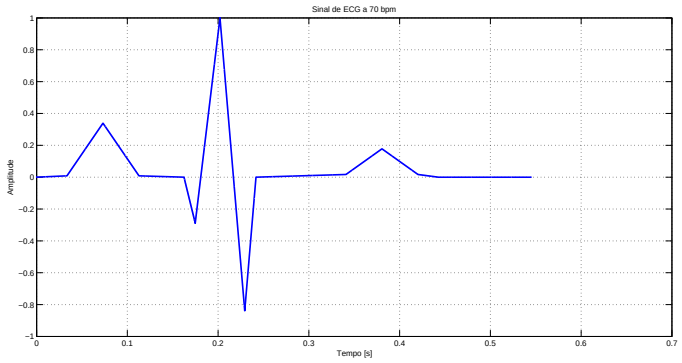
Fonte: Elaborada pelo autor utilizando Proteus^{TM}.

Esses sinais digitais foram posteriormente filtrados por filtros RC Passa-Baixas de 4^a ordem, visto na figura 19, feitos com capacitores de 82nF e resistores de 10kΩ, e com frequência de corte de aproximadamente 194 Hz.

A frequência de corte foi escolhida baseando-se no espectro da onda PQRST gerada pelo MATLAB**?**. A onda no domínio do tempo pode ser vista na figura 20 e no domínio da frequência na figura 21. A frequência desejada era de 200 Hz, pois nessa faixa e possível obter grande parte da potência do sinal, porém utilizando valores comerciais de componentes, o mais próximo obtido foi de 194 Hz, como visto na equação abaixo:

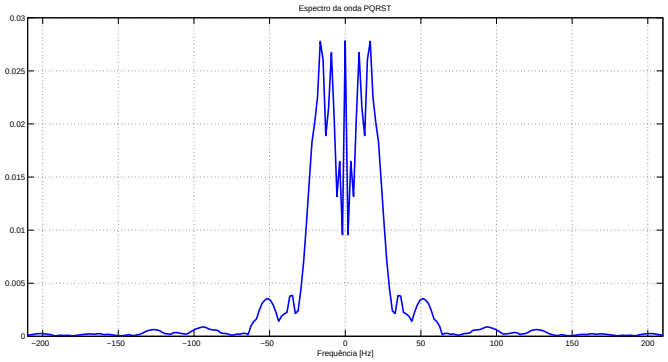
$$f_{corte} = \frac{1}{2\pi RC} = \frac{1}{2.\pi.10.10^{3}.82.10^{-9}} \simeq 194,0914 \tag{3.1}$$

Figura 20 – Onda PQRST gerada pela função *ecg()* do MATLAB^{TM}.



Fonte: Elaborada pelo autor utilizando MATLAB^{TM}.

Figura 21 – Espectro da onda PQRST gerada pelo MATLAB^{TM}.



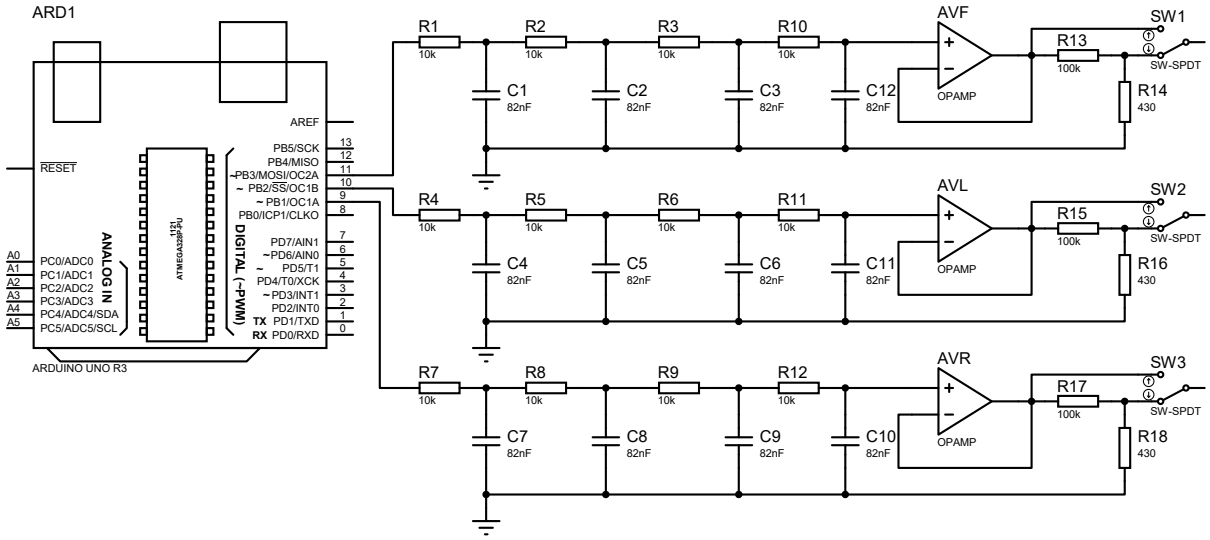
Fonte: Elaborada pelo autor utilizando MATLAB^{TM}.

A escolha da ordem do filtro foi realizada de maneira empírica, com simulações no MATLAB^{TM}. Os testes realizados com 1^a, 2^a e 3^a ordem ainda apresentação oscilações no sinal de saída, de forma que $4^a a$ ordem foi onde o sinal tornou-se mais fidedigno.

3.5 Circuito utilizado

O circuito utilizado pode ser visto na figura 22. Nela podemos ver os pinos utilizados do Arduino^{TM}, os filtros utilizados, os *buffers* e os potenciômetros na saída para regulagem de amplitude do sinal. As chaves são para determinar se a saída será com amplitude máxima para visualização em osciloscópio ou com amplitude atenuada para uso em eletrocardiógrafo.

Figura 22 – Circuito com Arduino^{TM}, filtros, *buffer*, chaves e divisores de tensão de saída.



Fonte: Elaborada pelo autor utilizando Proteus^{TM}.

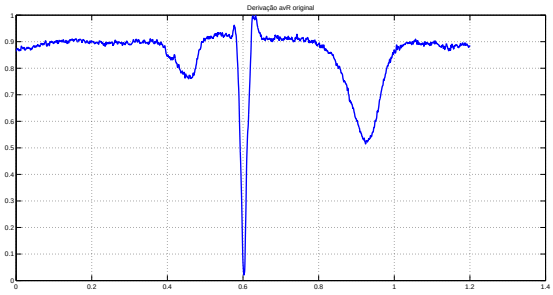
Com o *upload* do código já realizado, e em seguida em osciloscópio de bancada para visualizar os sinais correspondentes das derivações unipolares. Também foram realizados testes em um módulo de aquisição de ECG (ARAI, 2016).

4 Resultados

4.1 Testes iniciais

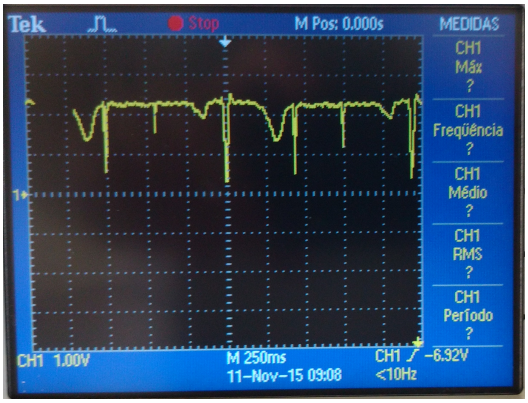
Durante as primeiras tentativas, houveram problemas ao gerar os três sinais simultaneamente. Ocorriam atrasos entre cada ciclo, e os valores do vetor de razão cíclica estouravam, causando distorção nos sinais. Com dois sinais simultâneos, os atrasos diminuíam consideravelmente, porém ainda havia distorção. Eles só eram reproduzidos de maneira fiel ao serem gerados individualmente.

Figura 23 – Derivação avR original



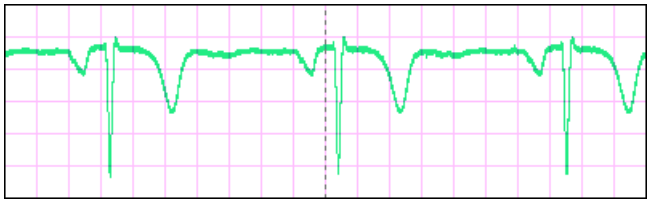
Fonte: Importada para o MATLAB^{TM} do PTB.

Figura 24 – Derivação avR obtida com distorção com total de dois sinais simultâneas.



Fonte: Obtida do osciloscópio.

Figura 25 – Derivação avR simulada via software com escala de amplitude de 500mV/div e de tempo de 200ms/div.

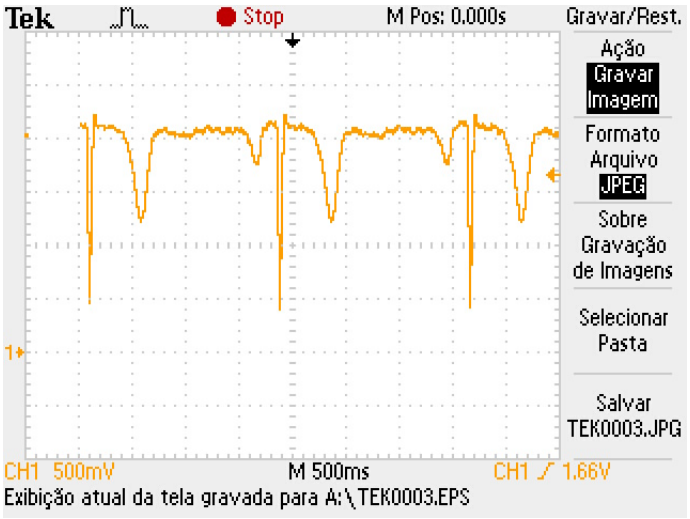


Fonte: Elaborada pelo autor utilizando Proteus^{TM}.

Ao simular no Proteus^{TM} os sinais simultaneamente, o problema não foi distorção, mas sim o fato de não haver sinal algum, tendo como saída dos pinos nível de tensão de 0 V. Na figura 23, temos o sinal avR original, como deveria ser reproduzido, já na figura 24, vemos o sinal com distorção ao ser reproduzido simultaneamente com outras derivações, e na figura 25 o sinal simulado com uma única saída, sem outras em paralelo.

A mudança realizada que corrigiu tais problemas, foi alterar o tipo de variável dos vetores de razão cíclica. Em vez de utilizar "**unsigned int**", foi utilizada a variável do tipo "**byte**". Com isso, os elementos do vetor não estouravam, e não houve mais atraso, pois **unsigned int** é maior do que apenas um único **byte** (KERNIGHAN; RITCHIE, 1986). Isso foi confirmado ao executar o código utilizando o computador para visualizar os valores de saída dos pinos PWM ciclo a ciclo. Haviam valores que saiam do limite de 255 necessário para a função "**analogWrite**", vista na seção 3.4. Com a mudança de variável, o limite não foi mais estourado permitindo a execução do código.

Figura 26 – Derivação avR gerada pelo microcontrolador.

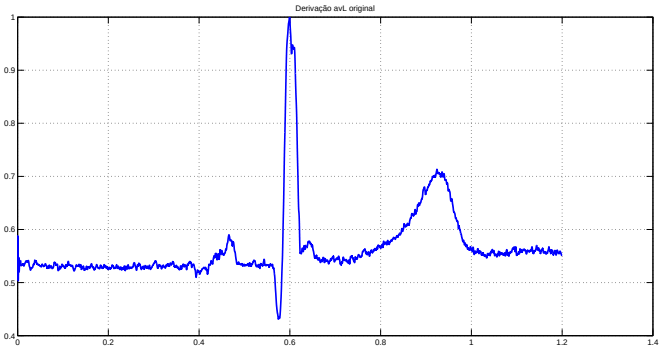


Fonte: Obtida do osciloscópio.

Na figura 26, podemos observar o sinal no osciloscópio após essa mudança. Nota-se

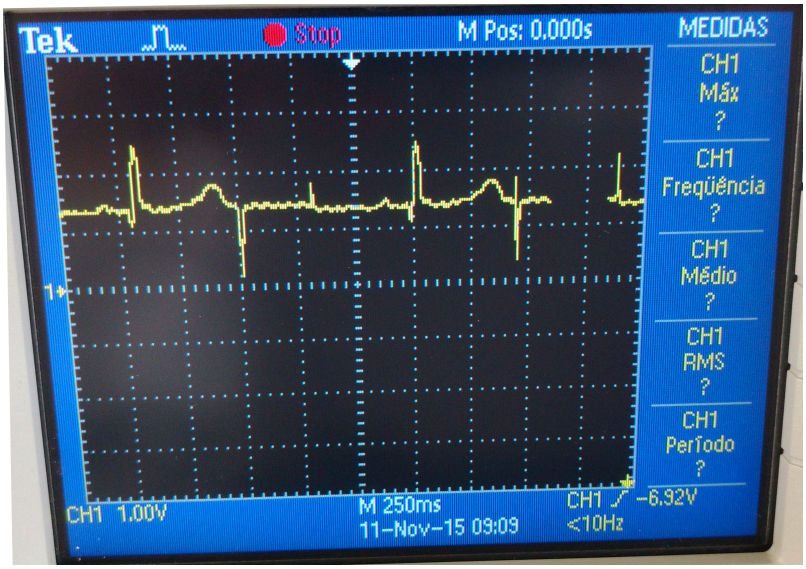
que não há mais a presença dos *spikes* causando distorção.

Figura 27 – Derivação avL original



Fonte: Importada para o MATLAB^{TM} do PTB.

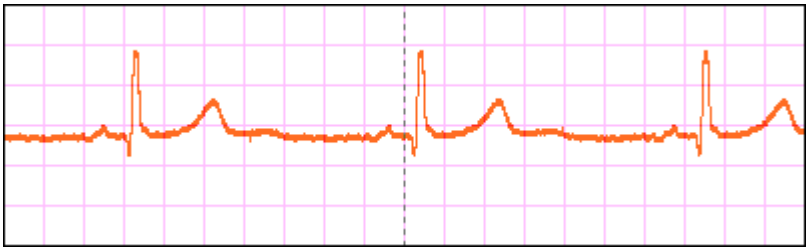
Figura 28 – Derivação avL obtida com distorção com total de dois sinais simultâneas.



Fonte: Obtida do osciloscópio.

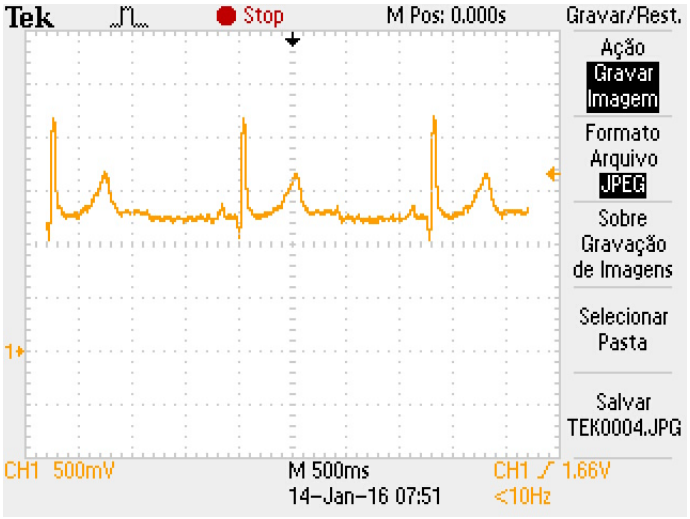
Podemos ver que o mesmo ocorreu com a derivação avL. Na figura 27 temos o sinal original, na figura 28 o sinal distorcido com os *spikes* e o sinal simulado na figura 29. Na figura 30 temos o sinal após a mudança no código removendo os *spikes*.

Figura 29 – Derivação avL simulada via software com escala de amplitude de 500mV/div e de tempo de 200ms/div.



Fonte: Elaborada pelo autor utilizando Proteus^{TM}.

Figura 30 – Derivação avL gerada pelo microcontrolador.

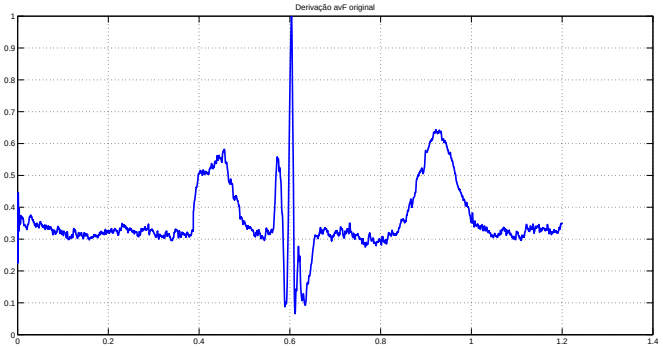


Fonte: Obtida do osciloscópio.

Assim como nas derivações avR e avL, na avF ocorreram os mesmo eventos. Na figura 31 temos o sinal original da derivação avF, e na figura 32 temos a derivação avF com seus *spikes*. Na figura 33 vemos a simulação do sinal no Proteus^{TM} e na figura 33 temos a derivação avF obtida no osciloscópio após a mudança no código.

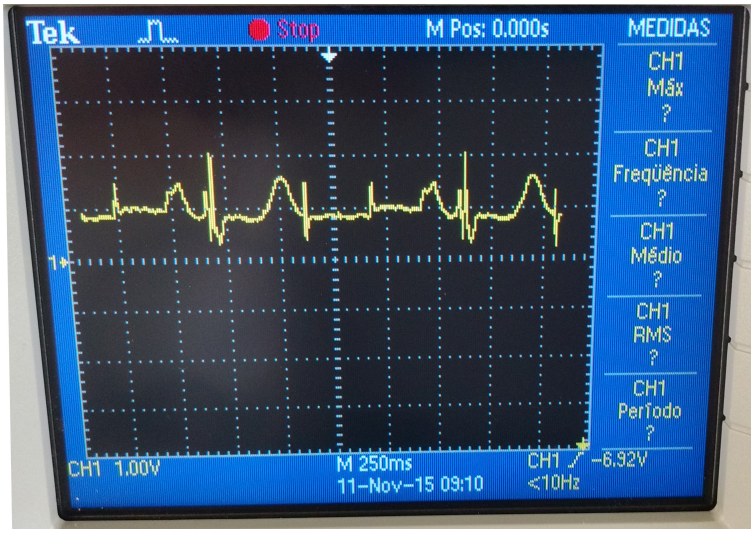
A mudança do código também fez com que as simulações, assim como os sinais no osciloscópio, reproduzissem as 3 derivações unipolares ao mesmo tempo.

Figura 31 – Derivação avF original



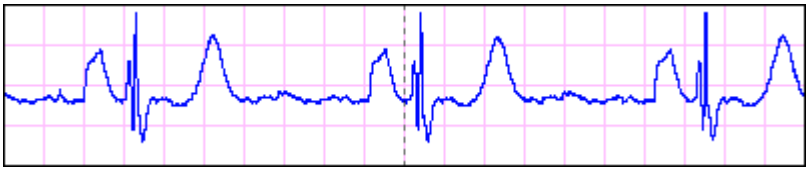
Fonte: Importada para o MATLAB^{TM} do PTB.

Figura 32 – Derivação avF obtida com distorção com total de dois sinais simultâneas.



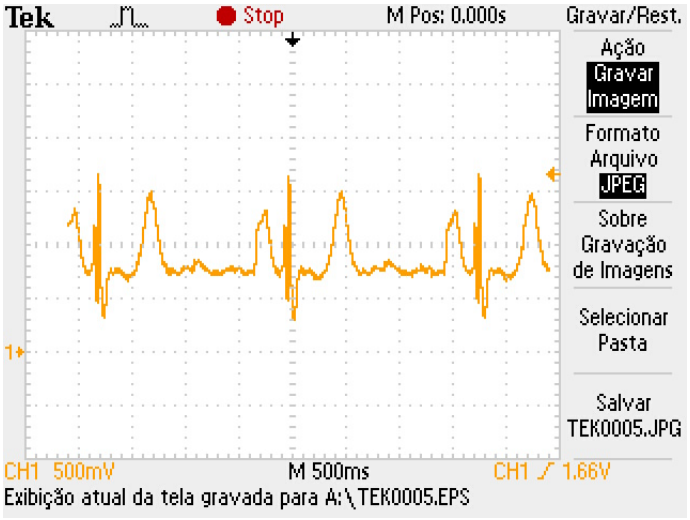
Fonte: Importada para o MATLAB^{TM} do PTB.

Figura 33 – Derivação avF simulada via software com escala de amplitude de 500mV/div e de tempo de 200ms/div.



Fonte: Elaborada pelo autor utilizando Proteus^{TM}.

Figura 34 – Derivação avF gerada pelo microcontrolador.



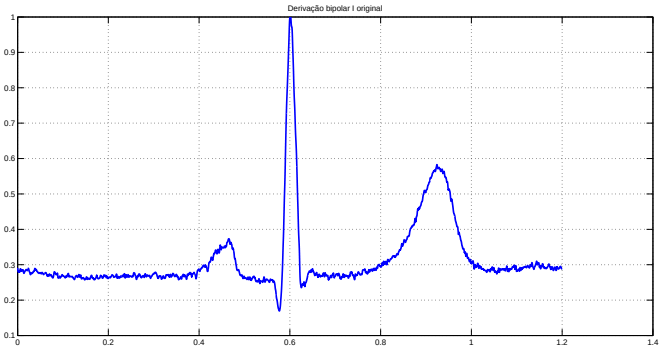
Fonte: Obtida do osciloscópio.

4.1.1 Derivações bipolares simuladas I, II, III.

Para fins comparativos, foram realizadas operações no Proteus^{TM}, para obter as derivações I, II e III. Abaixo podemos ver tanto as derivações bipolares da amostra original, e a obtida através de operações realizadas no Proteus^{TM} das derivações unipolares aumentadas. Sendo que:

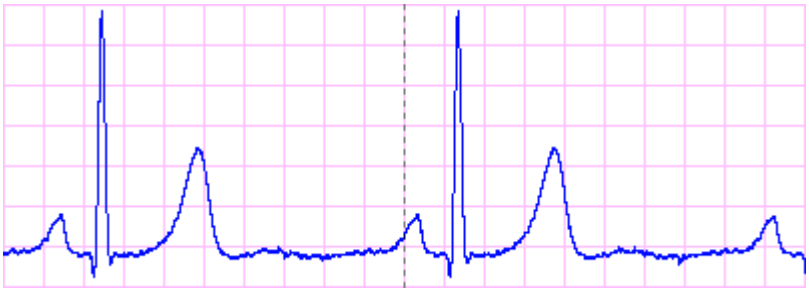
- Bipolar I = avL - avR;
- Bipolar II = avF - avR;
- Bipolar III = avF - avL;

Figura 35 – Derivação bipolar I original.



Fonte: Importada para o MATLAB^{TM} do PTB.

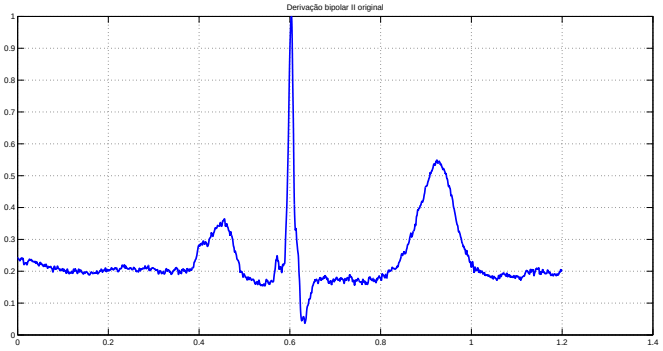
Figura 36 – Derivação bipolar I simulada no Proteus^{TM}.



Fonte: Elaborada pelo autor utilizando Proteus^{TM}.

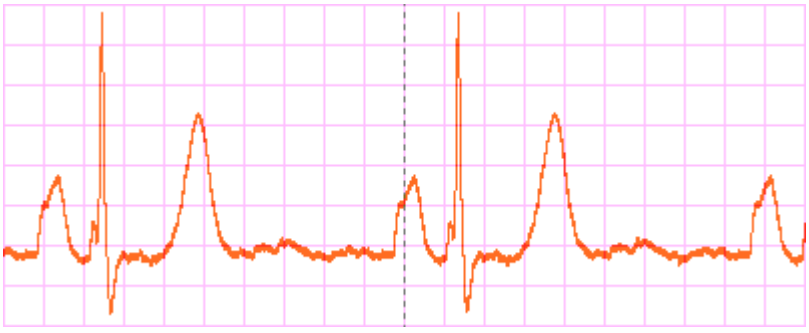
Na figura 35 vemos a derivação bipolar I importada para o MATLAB^{TM}, e na figura 36, temos a derivação simulada. Esse sinal simulado foi obtido através da subtração dos sinais avL com avR, através de um amplificador operacional ideal.

Figura 37 – Derivação bipolar II original.



Fonte: Importada para o MATLAB^{TM} do PTB.

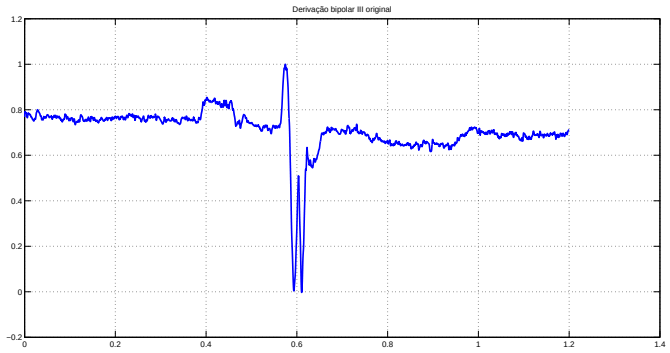
Figura 38 – Derivação bipolar II simulada no Proteus^{TM}.



Fonte: Elaborada pelo autor utilizando Proteus^{TM}.

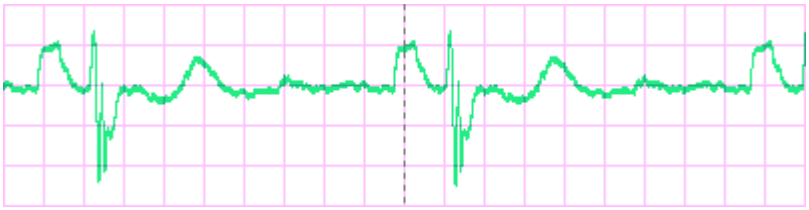
O mesmo ocorreu na derivação bipolar II. O sinal da figura 38 foi simulado com subtração dos sinais avF e avR através de um amplificador operacional ideal. Para comparar, na figura 37, vemos a derivação II original.

Figura 39 – Derivação bipolar III original.



Fonte: Importada para o MATLAB^{TM} do PTB.

Figura 40 – Derivação bipolar III simulada no Proteus^{TM}.



Fonte: Elaborada pelo autor utilizando Proteus^{TM}.

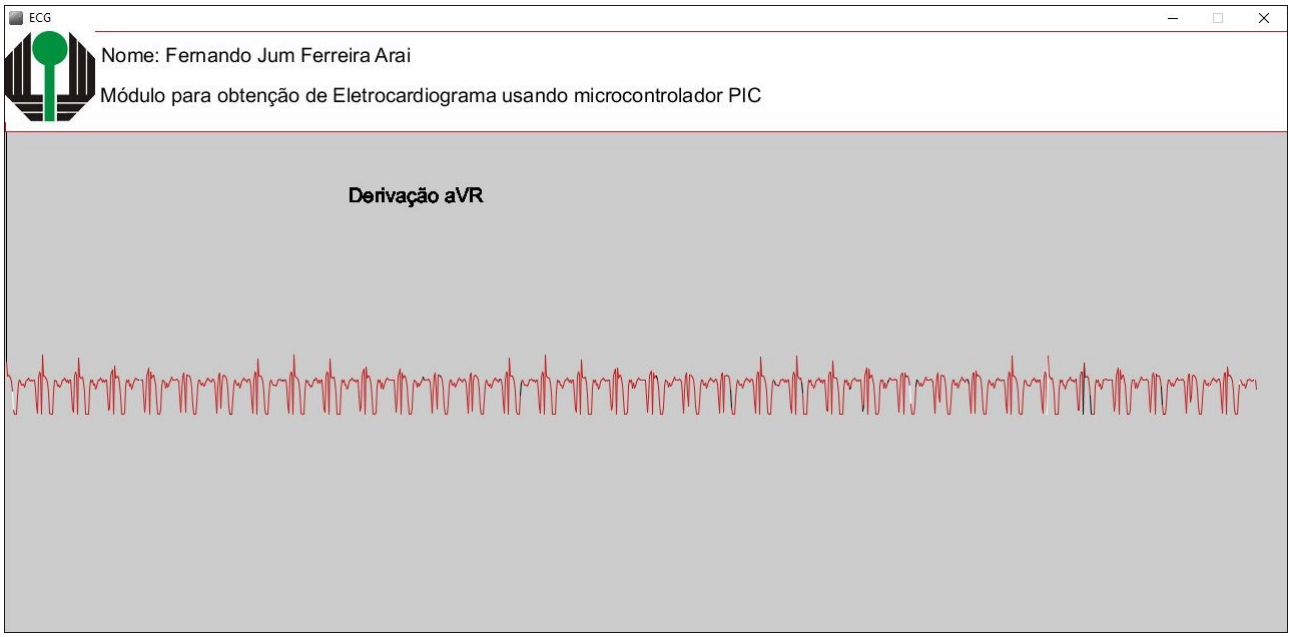
Assim como nas derivações bipolares I e II, a III foi feita com amplificadores operacionais ideais, subtraindo avF e avL. Na figura 40 podemos ver o resultado da simulação da derivação bipolar III e na figura 39 o sinal original.

4.2 Testes com módulo de obtenção de eletrocardiograma

Foram realizados testes com um módulo de obtenção de eletrocardiograma(ARAI, 2016) para observar as derivações unipolares avR, avL e avF e também as derivações bipolares I, II e III. O módulo apresenta sua saída na tela de um computador.

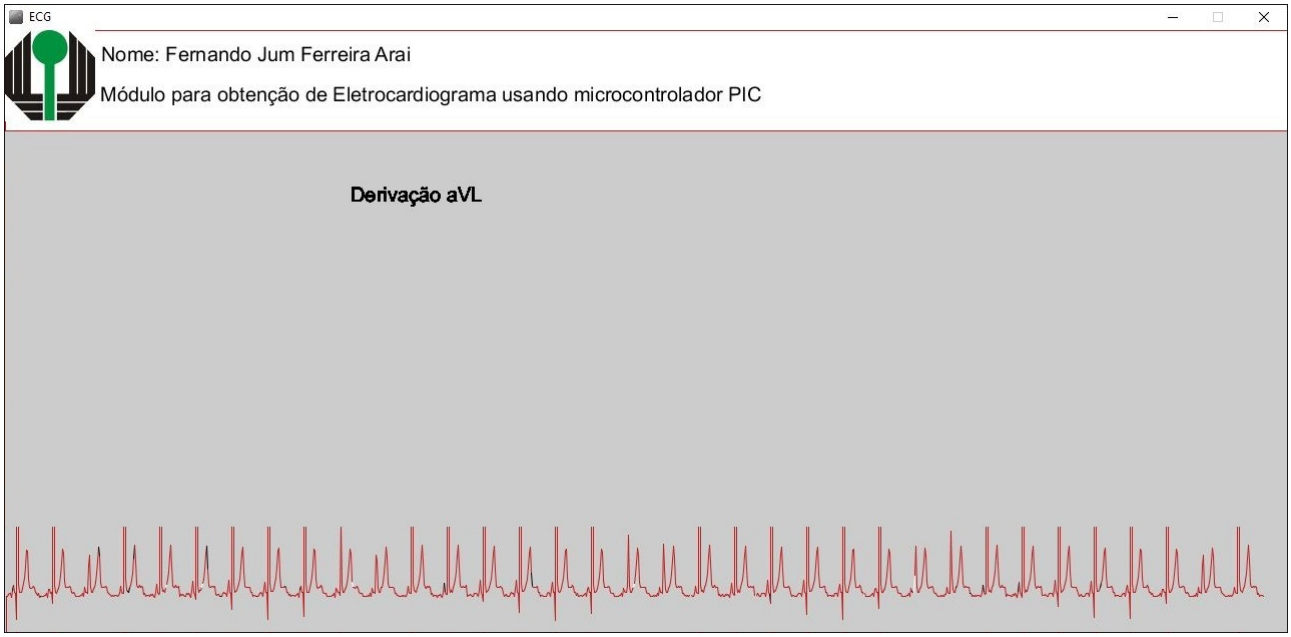
Nas figuras 41, 42 e 43 vemos o resultado das derivações avR, avL e avF no módulo de aquisição.

Figura 41 – Derivação unipolar avR do módulo de obtenção.



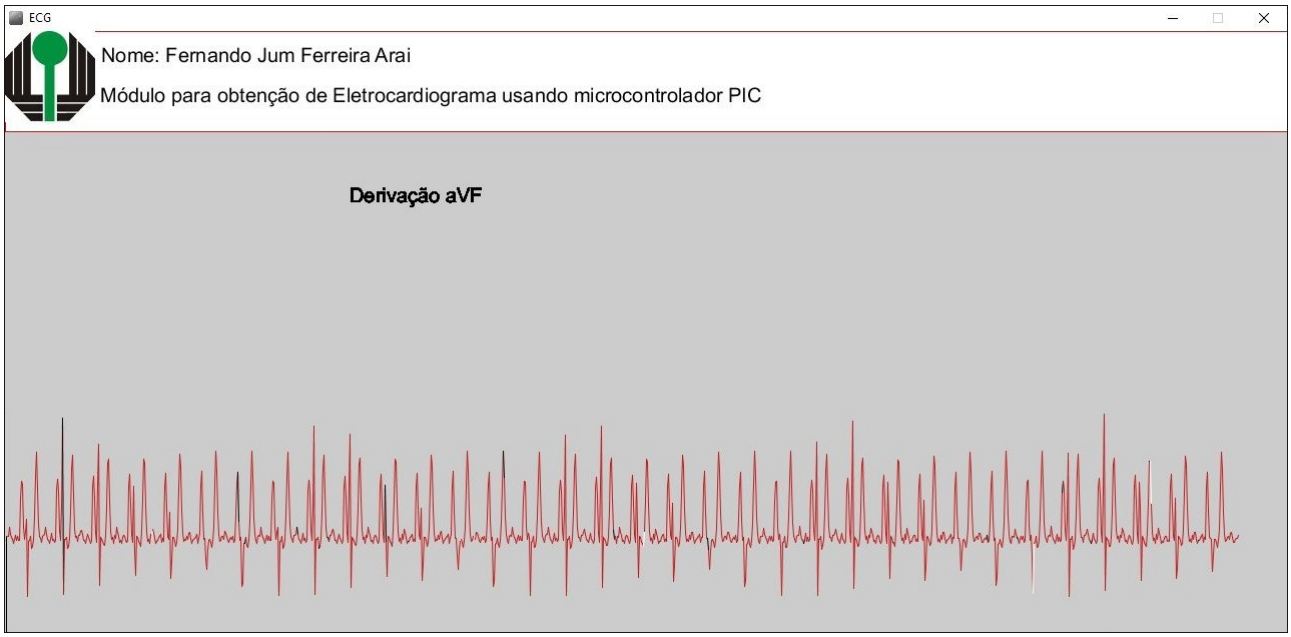
Fonte: Obtida pelo módulo de aquisição.

Figura 42 – Derivação unipolar avL do módulo de obtenção.



Fonte: Obtida pelo módulo de aquisição.

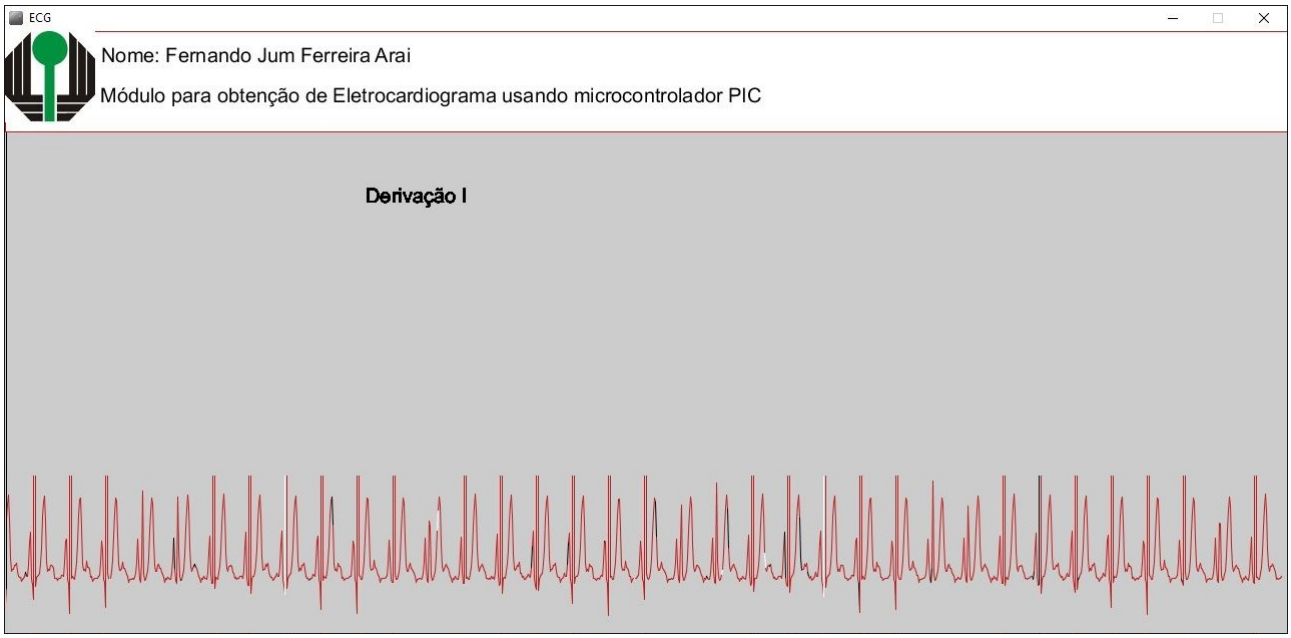
Figura 43 – Derivação unipolar avF do módulo de obtenção.



Fonte: Obtida pelo módulo de aquisição.

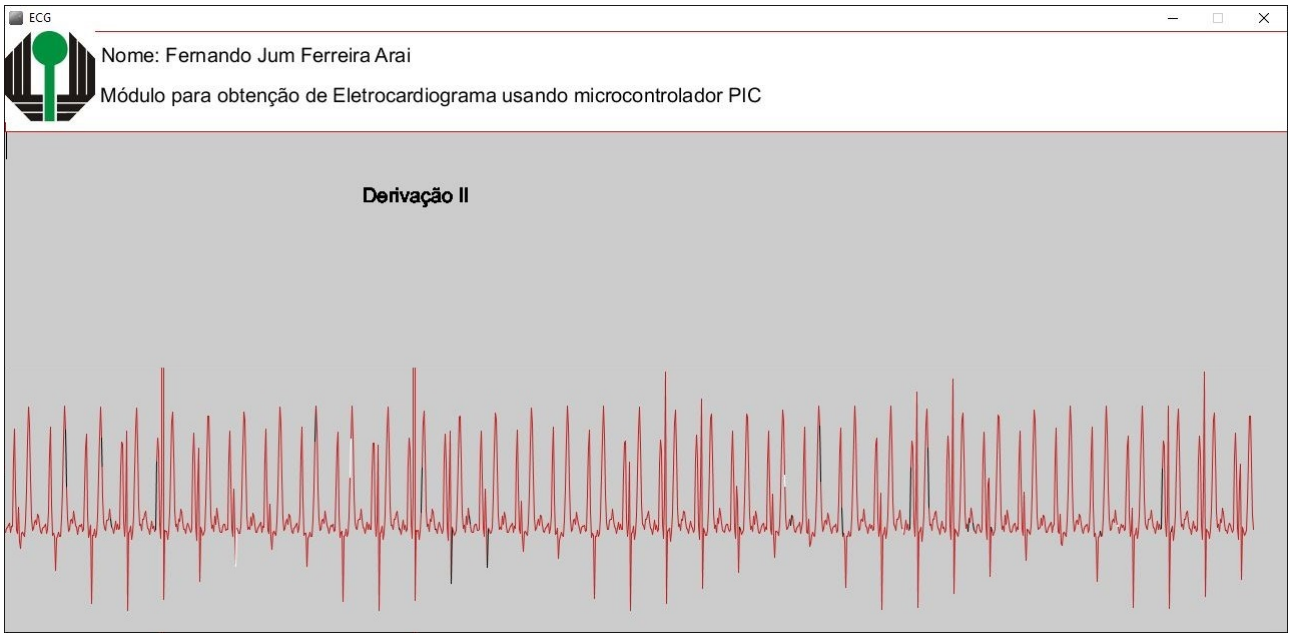
Nas figuras 44, 45 e 46 vemos o resultado das derivações avR, avL e avF no módulo de aquisição.

Figura 44 – Derivação bipolar I do módulo de obtenção.



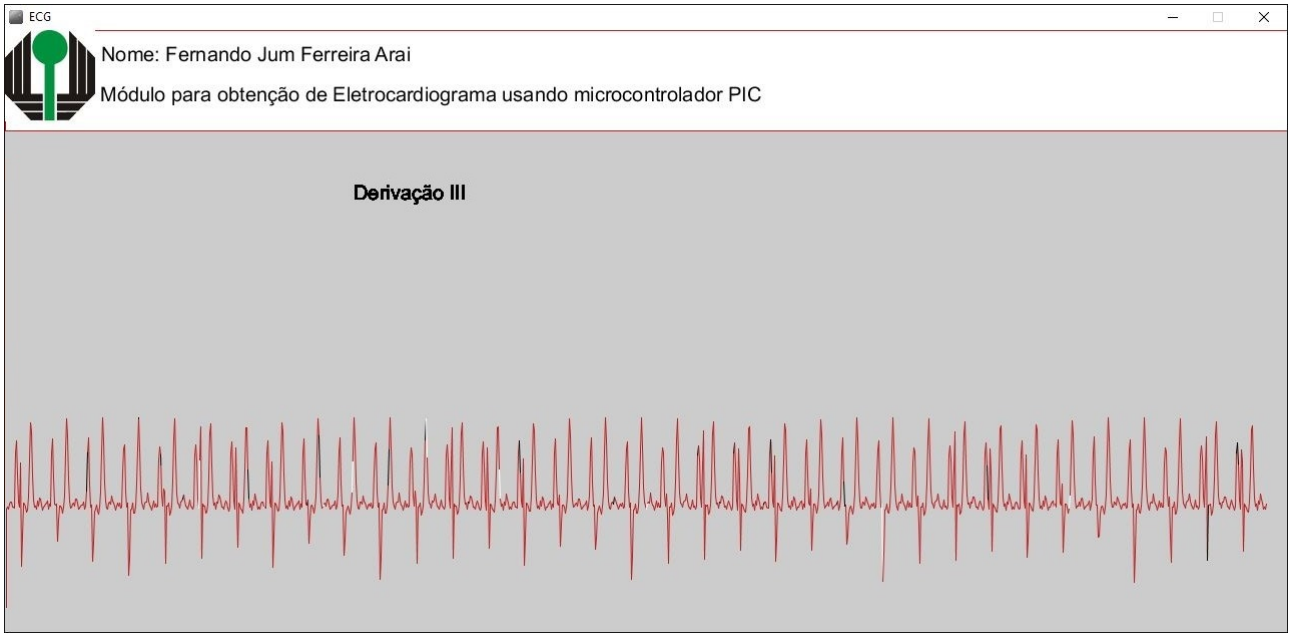
Fonte: Obtida pelo módulo de aquisição.

Figura 45 – Derivação bipolar II do módulo de obtenção.



Fonte: Obtida pelo módulo de aquisição.

Figura 46 – Derivação bipolar III do módulo de obtenção.



Fonte: Obtida pelo módulo de aquisição.

5 Discussão Final

A proposta deste trabalho foi criar um módulo que simulasse os sinais elétricos do coração a fim de dispensar o uso de seres vivos em testes e calibrações. O método utilizado foi a filtragem do sinal modulado através de PWM. Tais objetivos foram atingidos.

Entre as dificuldades iniciais, uma delas foi encontrar um banco de dados com amostras de ECG. Outra foi uma tentativa de criar um PWM manualmente no microcontrolador, com o objetivo de utilizar frequência de PWM maior, entretanto, o uso da frequência escolhida no trabalho mostrou-se mais do que suficiente para atender as necessidades do projeto.

Não foi possível obter detalhes maiores das componentes de ruídos inclusos naturalmente no ECG, como ruído muscular, ruído de 60 Hz da rede elétrica e composições espectrais de artefatos de movimento. Estes ruídos são importantes pois sem eles não é possível saber se os filtros do eletrocardiógrafo em teste estão funcionando de maneira adequada.

Para trabalhos futuros, além da adição dos ruídos para um sinal mais fidedigno, é sugerido o uso de um microcontrolador com mais portas PWM para reproduzir as derivações pré-cordiais, o uso de uma memória externa para armazenar ECGs com cardiopatias além de amostras saudáveis. Também sugere-se o uso de um potenciômetro para controle da frequência dos batimentos cardíacos e uma interface para o usuário, onde ele pode escolher qual o tipo de sinal ele deseja reproduzir, além de fornecer alguns dados em tempo real. Essa interface poderia ser feita tanto com display de LCD quanto uma tela *touch*.

Referências

ARAI, F. J. F. *Módulo de Obtenção de Eletrocardiograma usando Microcontrolador PIC*. Monografia (Graduação em Engenharia Elétrica) — Departamento de Engenharia Elétrica, Centro de Tecnologia e Urbanismo, Universidade Estadual de Londrina, Londrina, 2016. 20, 28

ARDUINO. *Arduino - analogWrite*. 2016. Disponível em: <https://www.arduino.cc/en/Reference/AnalogWrite>. Acesso em: 2016-02-12. 18

ARDUINO. *Arduino UNO & Genuino UNO - Overview*. 2016. Disponível em: <https://www.arduino.cc/en/Main/ArduinoBoardUno>. Acesso em: 2016-02-12. 15

ARDUINO. *What is Arduino?* 2016. Disponível em: <https://www.arduino.cc/en/Guide/Introduction#>. Acesso em: 2016-02-12. 15

CANER, C.; ENGIN, M.; ENGIN, E. Z. The programmable ecg simulator. *Journal of Medical Systems*, v. 32, p. 355–359, 2008. 1

CHANG, J.-R. C.; TAIA, C.-C. Accurate programmable electrocardiogram generator using a dynamical model implemented on a microcontroller. *Review of Scientific Instruments*, v. 77, 2006. 1

COUCH II, L. W. *Digital and Analog Communication Systems*. 6. ed. [S.l.]: Prentice Hall, 2002. 8

DRAGHICIU, N.; CRETIU, P. Ecg simulator. *Journal of Electrical and Electronics Engineering*, v. 6, p. 33–36, 2013. 1

GUYTON, A. C.; HALL, J. E. *Tratado de Fisiologia Médica Quarta Edição*. [S.l.]: Saunders Elsevier, 2006. 3, 4

KERNIGHAN, B. W.; RITCHIE, D. M. *C - A Linguagem de Programação*. [S.l.]: Campus, 1986. 22

MALVINO, A. P. *Eletrônica Vol. 2 Quarta Edição*. [S.l.]: McGraw Hill, 1995. 9

MATHWORKS. *Interpolation - increase sampling rate by integer factor - MATLAB interp*. 2016. Disponível em: <http://www.mathworks.com/help/signal/ref/interp.html>. Acesso em: 2016-02-15. 16

OPPENHEIM, A. V.; SCHAFER, R. W.; BUCK, J. R. *Discrete-Time Signal Processing Second Edition*. [S.l.]: Prentice Hall, 1999. 13, 14, 15

PORTAL-EDUCAÇÃO. *O Eletrocardiograma*. 2013. Disponível em: <http://www.portaleducacao.com.br/enfermagem/artigos/30351/o-eletrocardiograma>. Acesso em: 2016-02-09. 1

UNICAMP/UNIFESP. *GEMA - Gerenciamento de Manutençao de Equipamentos Médico-Hospitalares*. 2016. [CD-ROM]. 3, 5, 6, 7, 8

WEI, Y.-C. et al. A three-lead, programmable, and microcontroller-based electrocardiogram generator with frequency domain characteristics of heart rate variability. *Review of Scientific Instruments*, v. 82, 2012. 1

6 Apendice

6.1 Termos da série de Fourier de sinal PWM

6.1.1 Termo a_0

$$a_0 = \frac{1}{T}\int_0^T f(t)dt = \frac{1}{T}\int_0^{T_{on}} V_p dt + \frac{1}{T}\int_{T_{on}}^T 0dt = \frac{T_{on}}{T}V_p = DV_p \tag{6.1}$$

6.1.2 Termo a_n

$$a_n = \frac{2}{T}\int_0^T f(t)cos\left(\frac{2\pi nt}{T}\right)dt = \frac{2}{T}\int_0^{T_{on}} V_p cos\left(\frac{2\pi nt}{T}\right)dt + \frac{2}{T}\int_{T_{on}}^T 0cos\left(\frac{2\pi nt}{T}\right)dt$$

$$a_n = \frac{2}{T}\left[V_p\frac{T}{2\pi n}sin\left(\frac{2\pi nt}{T}\right)\right]_0^{T_{on}} = \frac{V_p}{\pi n}sin\left(\frac{2\pi nT_{on}}{T}\right) = \frac{V_p}{\pi n}sin(2\pi nD) \tag{6.2}$$

6.1.3 Termo b_n

$$a_n = \frac{2}{T}\int_0^T f(t)sin\left(\frac{2\pi nt}{T}\right)dt = \frac{2}{T}\int_0^{T_{on}} V_p sin\left(\frac{2\pi nt}{T}\right)dt + \frac{2}{T}\int_{T_{on}}^T 0sin\left(\frac{2\pi nt}{T}\right)dt$$

$$a_n = \frac{-2}{T}\left[V_p\frac{T}{2\pi n}cos\left(\frac{2\pi nt}{T}\right)\right]_0^{T_{on}} = \frac{-V_p}{\pi n}cos\left(\frac{2\pi nT_{on}}{T}\right) + \frac{V_p}{\pi n} = \frac{V_p}{\pi n}(1 - cos(2\pi nD)) \tag{6.3}$$

6.2 Código MATLAB^{TM} para obter razões cíclicas

```
clear all
clc
% cd E:\EEUEL\EE2015\TCC\Amostras %Pasta contendo arquivo com amostras;
load('s0504_rem') %Carrega o arquivo com as amostras de ECG;

%Normalização da amplitude das amostras:
ecg_o = val/1000;
clear val
%Freq. de corte do filtro passa-baixas:
freqFiltro = 194;
bpm = 50;
freqcard = bpm/60;
%Freq. do PWM:
fc = 31250/64;
```

```
15 %Número de amostras por período de PWM:
16 nS = 256;
17 %Período de amostragem:
18 Ts = 1/(nS*fc);
19 fs = 1/Ts;
20
21 tempo = Ts:Ts:1/freqcard;
22 ecg = zeros(length(ecg_o(:,1)),length(tempo));
23
24 for lin=1:length(ecg(:,1))
25     %Interpolação do sinal;
26     aux1 = interp(ecg_o(lin,:),fs/1000);
27     %Seleção do primeiro batimento;
28     ecg(lin,:) = aux1(1:length(tempo));
29     %Norm. de offset;
30     ecg(lin,:) = ecg(lin,:) + abs(min(ecg_o(lin,:)));
31     %Norm. de pico;
32     ecg(lin,:) = ecg(lin,:) / max(ecg(lin,:));
33 end
34
35 %Normalização de amplitudes de transição:
36 c6 = .06*[ones(1,4.841e4) zeros(1,(150e3-4.841e4))];
37 c7 = .021*[ones(1,5.156e4) zeros(1,(150e3-5.156e4))];
38 c9 = .0116*[ones(1,7.084e4) zeros(1,(150e3-7.084e4))];
39 c12 = .02*[zeros(1,1.314e5) ones(1,(150e3-1.314e5))];
40
41 ecg(6,:) = ecg(6,:) - c6;
42 ecg(7,:) = ecg(7,:) + c7;
43 ecg(9,:) = ecg(9,:) + c9;
44 ecg(12,:) = ecg(12,:) - c12;
45
46 clear aux1 ecg_o
47
48 %Onda triangular da portadora:
49 triang = 1+sawtooth(2*pi*fc*tempo,0.5);
50 saida = zeros(15,length(tempo));
51
52 %Comparação com triang. para gerar PWM:
53 for col=1:length(ecg(:,1))
54     for i=1:length(ecg(1,:))
55         if triang(i) <= ecg(col,i)
56             saida(col,i) = 1;
57         else
58             saida(col,i) = 0;
59         end
60     end
61 end
62 for col=1:length(ecg(:,1));
63     for lin=1:nInterp:length(triang)
64         if saida(col,lin) ~= 1
65             saida(col,lin) = 1;
66         end
67     end
68 end
69
70 contagem = 0;
```

```
auxcont = 1;
dutycycle_i = 0;
dutycycle_ii = 0;
dutycycle_iii = 0;
dutycycle_avr = 0;
dutycycle_avl = 0;
dutycycle_avf = 0;
dutycycle_v1 = 0;
dutycycle_v2 = 0;
dutycycle_v3 = 0;
dutycycle_v4 = 0;
dutycycle_v5 = 0;
dutycycle_v6 = 0;
dutycycle_vx = 0;
dutycycle_vy = 0;
dutycycle_vz = 0;

%Cálculo de razão cíclica para as derivações:
for i=1:length(ecg(1,:))
    if auxcont == 1
        if saida(1,i) == 1
            contagem = contagem + 1;
        else
            dutycycle_i(end+1) = contagem;
            contagem = 0;
            auxcont = 0;
        end
    else
        if saida(1,i) == 0
            contagem = 1;
        else
            contagem = contagem + 1;
            auxcont = 1;
        end
    end
end
for i=1:length(ecg(1,:))
    if auxcont == 1
        if saida(2,i) == 1
            contagem = contagem + 1;
        else
            dutycycle_ii(end+1) = contagem;
            contagem = 0;
            auxcont = 0;
        end
    else
        if saida(2,i) == 0
            contagem = 1;
        else
            contagem = contagem + 1;
            auxcont = 1;
        end
    end
end
for i=1:length(ecg(1,:))
    if auxcont == 1
```

```
127         if saida(3,i) == 1
128             contagem = contagem + 1;
129         else
130             dutycycle_iii(end+1) = contagem;
131             contagem = 0;
132             auxcont = 0;
133         end
134     else
135         if saida(3,i) == 0
136             contagem = 1;
137         else
138             contagem = contagem + 1;
139             auxcont = 1;
140         end
141     end
142 end
143
144 for i=1:length(ecg(1,:))
145     if auxcont == 1
146         if saida(4,i) == 1
147             contagem = contagem + 1;
148         else
149             dutycycle_avr(end+1) = contagem;
150             contagem = 0;
151             auxcont = 0;
152         end
153     else
154         if saida(4,i) == 0
155             contagem = 1;
156         else
157             contagem = contagem + 1;
158             auxcont = 1;
159         end
160     end
161 end
162 for i=1:length(ecg(1,:))
163     if auxcont == 1
164         if saida(5,i) == 1
165             contagem = contagem + 1;
166         else
167             dutycycle_avl(end+1) = contagem;
168             contagem = 0;
169             auxcont = 0;
170         end
171     else
172         if saida(5,i) == 0
173             contagem = 1;
174         else
175             contagem = contagem + 1;
176             auxcont = 1;
177         end
178     end
179 end
180 for i=1:length(ecg(1,:))
181     if auxcont == 1
182         if saida(6,i) == 1
```

```
183                 contagem = contagem + 1;
184             else
185                 dutycycle_avf(end+1) = contagem;
186                 contagem = 0;
187                 auxcont = 0;
188             end
189         else
190             if saida(6,i) == 0
191                 contagem = 1;
192             else
193                 contagem = contagem + 1;
194                 auxcont = 1;
195             end
196         end
197 end
198 for i=1:length(ecg(1,:))
199         if auxcont == 1
200             if saida(7,i) == 1
201                 contagem = contagem + 1;
202             else
203                 dutycycle_v1(end+1) = contagem;
204                 contagem = 0;
205                 auxcont = 0;
206             end
207         else
208             if saida(7,i) == 0
209                 contagem = 1;
210             else
211                 contagem = contagem + 1;
212                 auxcont = 1;
213             end
214         end
215 end
216 for i=1:length(ecg(1,:))
217         if auxcont == 1
218             if saida(8,i) == 1
219                 contagem = contagem + 1;
220             else
221                 dutycycle_v2(end+1) = contagem;
222                 contagem = 0;
223                 auxcont = 0;
224             end
225         else
226             if saida(8,i) == 0
227                 contagem = 1;
228             else
229                 contagem = contagem + 1;
230                 auxcont = 1;
231             end
232         end
233 end
234 for i=1:length(ecg(1,:))
235         if auxcont == 1
236             if saida(9,i) == 1
237                 contagem = contagem + 1;
238             else
```

```
239             dutycycle_v3(end+1) = contagem;
240             contagem = 0;
241             auxcont = 0;
242         end
243     else
244         if saida(9,i) == 0
245             contagem = 1;
246         else
247             contagem = contagem + 1;
248             auxcont = 1;
249         end
250     end
251 end
252
253 for i=1:length(ecg(1,:))
254     if auxcont == 1
255         if saida(10,i) == 1
256             contagem = contagem + 1;
257         else
258             dutycycle_v4(end+1) = contagem;
259             contagem = 0;
260             auxcont = 0;
261         end
262     else
263         if saida(10,i) == 0
264             contagem = 1;
265         else
266             contagem = contagem + 1;
267             auxcont = 1;
268         end
269     end
270 end
271 for i=1:length(ecg(1,:))
272     if auxcont == 1
273         if saida(11,i) == 1
274             contagem = contagem + 1;
275         else
276             dutycycle_v5(end+1) = contagem;
277             contagem = 0;
278             auxcont = 0;
279         end
280     else
281         if saida(11,i) == 0
282             contagem = 1;
283         else
284             contagem = contagem + 1;
285             auxcont = 1;
286         end
287     end
288 end
289 for i=1:length(ecg(1,:))
290     if auxcont == 1
291         if saida(12,i) == 1
292             contagem = contagem + 1;
293         else
294             dutycycle_v6(end+1) = contagem;
```

```
295                 contagem = 0;
296                 auxcont = 0;
297             end
298         else
299             if saida(12,i) == 0
300                 contagem = 1;
301             else
302                 contagem = contagem + 1;
303                 auxcont = 1;
304             end
305         end
306 end
307 for i=1:length(ecg(1,:))
308         if auxcont == 1
309             if saida(13,i) == 1
310                 contagem = contagem + 1;
311             else
312                 dutycycle_vx(end+1) = contagem;
313                 contagem = 0;
314                 auxcont = 0;
315             end
316         else
317             if saida(13,i) == 0
318                 contagem = 1;
319             else
320                 contagem = contagem + 1;
321                 auxcont = 1;
322             end
323         end
324 end
325 for i=1:length(ecg(1,:))
326         if auxcont == 1
327             if saida(14,i) == 1
328                 contagem = contagem + 1;
329             else
330                 dutycycle_vy(end+1) = contagem;
331                 contagem = 0;
332                 auxcont = 0;
333             end
334         else
335             if saida(14,i) == 0
336                 contagem = 1;
337             else
338                 contagem = contagem + 1;
339                 auxcont = 1;
340             end
341         end
342 end
343 for i=1:length(ecg(1,:))
344         if auxcont == 1
345             if saida(15,i) == 1
346                 contagem = contagem + 1;
347             else
348                 dutycycle_vz(end+1) = contagem;
349                 contagem = 0;
350                 auxcont = 0;
```

```
351         end
352     else
353         if saida(15,i) == 0
354             contagem = 1;
355         else
356             contagem = contagem + 1;
357             auxcont = 1;
358         end
359     end
360 end
361
362 %Normalização dos termos de baixa razão cíclica:
363 dutycycle_i = dutycycle_i(dutycycle_i ~= 0);
364 dutycycle_ii = dutycycle_ii(dutycycle_ii ~= 0);
365 dutycycle_iii = dutycycle_iii(dutycycle_iii ~= 0);
366 dutycycle_avr = dutycycle_avr(dutycycle_avr ~= 0);
367 dutycycle_avl = dutycycle_avl(dutycycle_avl ~= 0);
368 dutycycle_avf = dutycycle_avf(dutycycle_avf ~= 0);
369 dutycycle_v1 = dutycycle_v1(dutycycle_v1 ~= 0);
370 dutycycle_v2 = dutycycle_v2(dutycycle_v2 ~= 0);
371 dutycycle_v3 = dutycycle_v3(dutycycle_v3 ~= 0);
372 dutycycle_v4 = dutycycle_v4(dutycycle_v4 ~= 0);
373 dutycycle_v5 = dutycycle_v5(dutycycle_v5 ~= 0);
374 dutycycle_v6 = dutycycle_v6(dutycycle_v6 ~= 0);
375 dutycycle_vx = dutycycle_vx(dutycycle_vx ~= 0);
376 dutycycle_vy = dutycycle_vy(dutycycle_vy ~= 0);
377 dutycycle_vz = dutycycle_vz(dutycycle_vz ~= 0);
378
379 %Concatenação de todas as derivações em uma única matriz:
380 dutyCycle = [dutycycle_i;dutycycle_ii;dutycycle_iii;dutycycle_avr;dutycycle_avl;
381 dutycycle_avf;dutycycle_v1;dutycycle_v2;dutycycle_v3;dutycycle_v4;dutycycle_v5;
382 dutycycle_v6;dutycycle_vx;dutycycle_vy;dutycycle_vz];
383 clear dutycycle_i dutycycle_ii dutycycle_iii dutycycle_avr dutycycle_avl
384 dutycycle_avf dutycycle_v1 dutycycle_v2 dutycycle_v3 dutycycle_v4 dutycycle_v5
385 dutycycle_v6 dutycycle_vx dutycycle_vy dutycycle_vz;
386
387 %Protótipo de filtro passa-baixas:
388 %(R=10e3; C=82e-9);
389 [B,A] = butter(4,Ts*freqFiltro);
390 Y = zeros(15,length(ecg(1,:)));
391
392 %Normalização dos sinais filtrados:
393 for i=1:15
394     Y(i,:) = (1/2)+(3/2)*filter(B,A,saida(i,:));
395 end
396
397 %Exportação das razões cíclicas:
398 DC = int16(dutyCycle);
399 dlmwrite('DutyCycleMatrix.txt',DC,'newline','pc')
400
401 %"Plot" dos sinais:
402 figure(1)
403 subplot(311)
404 plot(tempo,ecg(4,:))
405 grid;
406 title('Derivação avR original')
```

```
407
408 subplot(312)
409 plot(tempo,ecg(5,:))
410 grid;
411 title('Derivação avL original')
412
413 subplot(313)
414 plot(tempo,ecg(6,:))
415 grid;
416 title('Derivação avF original')
417
418 figure(2)
419 subplot(611)
420 plot(tempo,ecg(7,:))
421 grid;
422 title('Derivação v1 original')
423
424 subplot(612)
425 plot(tempo,ecg(8,:))
426 grid;
427 title('Derivação v2 original')
428
429 subplot(613)
430 plot(tempo,ecg(9,:))
431 grid;
432 title('Derivação v3 original')
433
434 subplot(614)
435 plot(tempo,ecg(10,:))
436 grid;
437 title('Derivação v4 original')
438
439 subplot(615)
440 plot(tempo,ecg(11,:))
441 grid;
442 title('Derivação v5 original')
443
444 subplot(616)
445 plot(tempo,ecg(12,:))
446 grid;
447 title('Derivação v6 original')
```

6.3 Código em linguagem C utilizado no Arduino^{TM}

```
1  //SIMULADOR DE ECG - VERSÃO 0.9
2  unsigned int index = 0;
3
4  //Derivações unipolares
5  byte avR[586] = {112,112,112,112,112,112,112,114,114,112,114,112,112,112,114,113,
6  114,114,113,112,112,114,114,114,114,114,114,114,114,114,114,114,114,116,114,115,116,
7  115,116,116,115,114,116,116,114,116,116,116,116,116,116,116,116,116,116,116,116,116,
8  116,116,116,116,116,116,116,116,116,116,116,116,116,116,116,117,118,118,118,116,117,
9  118,118,117,116,116,117,118,118,117,116,117,117,116,116,116,116,116,116,116,116,116,
10 116,116,116,116,116,116,116,116,116,116,116,116,116,116,116,116,116,116,114,116,114,
```

```
116,115,116,116,116,116,116,116,116,114,115,116,116,116,116,116,116,116,116,
116,115,116,116,116,115,116,116,116,116,115,114,114,115,116,117,116,116,116,118,
116,116,118,116,116,116,116,116,116,116,116,116,116,116,116,116,115,116,117,116,116,
117,116,116,116,116,116,117,116,116,115,116,116,115,114,114,114,114,112,110,110,
110,109,108,108,108,108,108,108,110,109,107,106,105,104,105,103,102,102,102,101,
100,102,99,100,100,100,98,99,100,99,98,98,100,100,101,103,104,107,108,110,112,
112,113,115,116,116,117,118,117,118,118,118,118,118,118,118,118,118,118,118,118,120,
120,120,120,119,120,120,120,120,119,120,120,118,118,120,120,119,118,120,120,118,
118,118,119,121,123,124,121,119,113,105,96,85,69,57,43,25,10,4,6,22,42,63,73,79,
90,100,110,122,127,128,128,127,128,128,125,123,122,120,120,119,118,118,118,118,
118,117,118,118,118,118,118,118,116,117,116,118,118,118,120,118,118,119,118,117,
118,118,116,117,118,118,118,118,118,118,118,118,118,118,118,118,118,117,116,116,117,
118,119,119,118,118,117,117,118,118,118,117,116,116,117,118,118,118,116,116,116,
116,116,116,116,116,117,116,114,114,114,116,115,114,114,114,114,114,113,112,112,
112,112,111,110,110,111,110,110,110,109,108,107,105,106,104,104,104,104,102,102,
100,99,97,98,96,96,95,93,91,89,88,87,86,86,84,83,82,80,79,78,77,76,74,72,72,72,
70,70,70,68,67,68,68,68,69,70,70,70,70,72,72,74,77,78,78,82,82,84,87,89,91,93,
94,97,98,99,100,102,104,104,106,107,107,108,108,110,110,112,112,111,114,113,114,
114,112,114,114,114,114,114,114,114,116,116,116,116,116,116,116,116,115,116,116,
116,116,117,116,116,116,116,115,115,114,116,116,115,115,116,116,116,116,116,115,
114,116,115,116,116,116,116,116,116,116,116,114,114,114,114,114,114,114,114,113,
113,114,115,114,112,112,112,113,114,115,114,112,114,114,114,114,114,114,114,114,
115,114,115,115,114,115,115,114,115,114,114,114};

byte avL[586] = {79,68,69,70,70,70,70,70,70,70,70,70,70,68,68,68,68,70,70,70,70,
70,70,70,70,68,70,70,68,70,70,69,68,68,69,68,68,68,68,70,70,70,70,69,68,68,68,68,
70,70,68,68,69,68,70,70,70,70,68,70,68,68,68,70,70,68,70,68,70,68,68,68,68,68,68,
68,68,68,68,70,70,68,68,68,68,70,68,68,68,69,68,68,68,70,68,69,68,68,68,68,68,68,
70,68,68,68,69,68,68,68,68,68,68,68,68,68,68,68,68,70,68,68,68,68,68,70,70,68,68,
70,70,70,69,68,68,68,68,69,68,68,68,68,70,68,68,69,70,70,70,70,69,68,70,68,70,68,
68,70,68,68,70,70,70,68,70,70,70,70,70,68,68,70,68,68,68,68,68,68,70,69,70,70,68,
70,70,70,70,69,70,70,69,68,66,68,68,68,68,68,68,68,68,68,68,68,68,68,70,70,70,70,
70,71,71,72,72,71,72,70,72,72,72,72,72,72,73,74,74,76,76,76,76,74,74,74,72,70,70,
70,70,70,70,70,70,70,70,70,70,70,70,70,70,70,70,70,70,70,70,70,68,70,68,70,68,70,70,
70,70,70,70,70,70,70,68,70,70,70,68,68,65,62,58,57,56,56,59,65,71,82,94,105,116,
123,126,128,128,124,121,122,122,121,114,107,98,86,81,72,72,73,72,74,74,74,74,74,
74,74,74,74,74,72,72,70,71,71,70,70,70,70,70,70,72,70,70,70,70,70,70,70,70,70,72,
72,70,70,70,71,71,70,70,70,70,70,70,70,70,70,70,70,70,71,72,72,70,70,70,70,70,72,
72,72,72,72,72,72,72,73,72,72,72,72,72,72,72,72,72,72,73,72,74,74,74,74,74,74,74,
74,74,74,74,75,74,74,74,76,76,76,76,76,76,76,76,76,76,76,78,78,78,80,78,78,78,80,
79,80,80,82,82,82,82,82,82,83,84,84,84,84,86,86,88,88,86,88,88,88,88,90,90,90,90,
90,90,91,92,92,92,92,90,91,92,90,91,90,90,88,88,88,87,85,86,84,82,82,81,80,80,79,
78,78,76,76,75,76,74,74,74,74,74,72,74,73,72,74,72,72,72,72,73,72,72,72,72,72,72,
72,72,72,72,72,72,72,72,72,72,72,72,72,72,72,72,72,72,72,72,72,72,72,72,72,72,72,
72,72,72,72,74,74,72,74,72,72,72,72,72,72,72,72,74,72,72,72,72,72,72,72,73,73,72,
72,72,74,73,74,73,72,72,72,74,72,72,72,72,72,72,74,72,72,72,72,72,72,72,72,72,72,
72,72,72};

byte avF[586] = {39,48,47,46,48,48,48,44,45,44,44,46,46,48,48,48,48,46,46,46,44,
46,44,44,46,46,46,44,44,44,44,44,44,44,44,44,44,44,44,42,42,42,42,42,44,44,43,42,42,
42,42,42,42,40,41,40,40,40,40,40,40,42,42,44,42,40,40,42,41,42,42,40,40,42,42,40,
42,40,40,40,40,40,40,40,41,40,40,40,40,42,42,42,42,42,42,42,42,42,42,42,42,42,44,
42,42,42,44,42,42,42,42,44,42,44,46,44,45,44,44,44,42,44,44,44,42,42,42,42,42,42,
40,40,42,42,44,42,44,44,44,44,44,44,44,45,42,42,44,44,42,42,40,40,42,42,40,42,40,40,
40,40,40,42,42,40,41,42,40,39,39,40,40,42,44,44,42,40,42,41,40,42,42,42,42,40,42,
42,41,40,40,42,42,42,54,56,60,60,63,66,66,66,66,66,67,66,66,66,66,66,66,68,68,70,
```

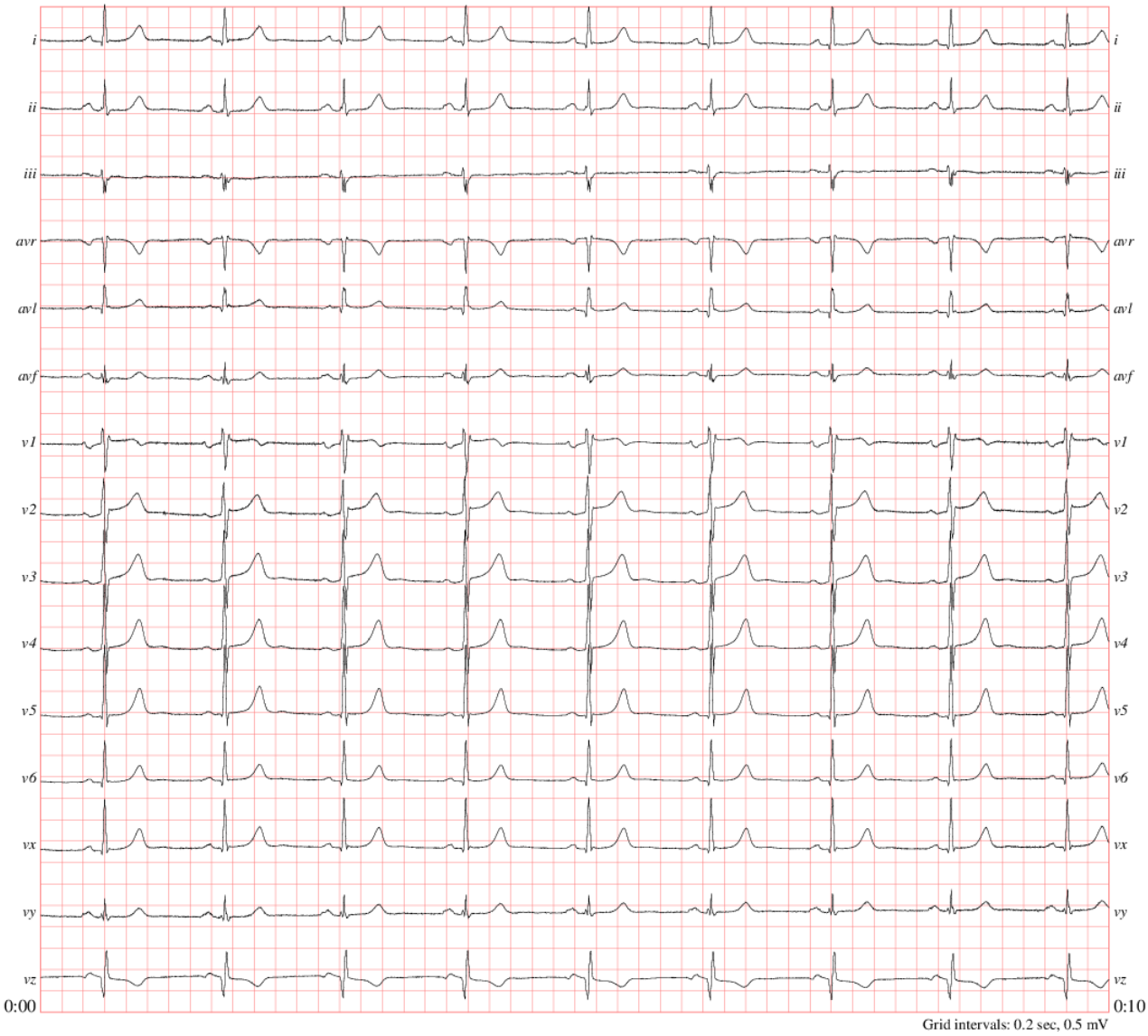
```
68,68,70,70,70,72,72,72,72,72,71,74,76,74,70,70,67,63,62,63,62,62,57,56,56,56,56,
54,53,50,48,48,47,46,46,46,44,44,44,44,44,44,42,44,42,40,40,42,42,42,42,42,40,40,
40,40,40,42,44,42,42,42,42,44,42,44,47,55,65,71,70,69,68,64,55,49,35,19,13,14,20,
40,70,99,120,125,91,53,14,12,18,26,35,30,29,18,14,16,16,14,14,19,22,22,24,26,30,
32,36,39,40,40,40,40,40,42,44,42,42,42,44,44,44,42,42,42,40,40,39,40,42,42,42,42,
42,42,42,42,44,44,42,40,42,42,42,44,42,45,44,42,42,40,40,40,42,42,42,40,40,40,40,
38,38,38,36,38,38,38,40,42,40,38,39,38,38,38,38,38,38,40,38,38,40,40,38,38,40,38,
40,42,42,42,40,42,42,42,42,42,42,42,44,44,46,48,46,48,46,46,47,50,51,52,54,55,52,
54,56,56,58,62,64,66,66,66,67,68,66,66,69,75,74,75,76,77,78,79,80,80,82,82,84,82,
82,82,83,82,82,80,80,78,78,78,76,76,74,74,73,72,70,68,67,66,63,62,60,60,59,58,56,
56,56,55,52,54,52,52,50,48,46,48,47,45,46,45,44,46,44,44,44,44,45,43,44,43,43,44,
42,42,42,42,41,40,40,42,42,40,40,41,42,42,42,42,44,44,42,44,41,42,42,42,42,42,40,
40,40,40,40,42,42,40,40,40,40,40,40,42,44,45,44,42,42,44,44,44,44,42,44,44,44,44,
44,45,43,42,43,44,44,44,42,44,45,44,42,43,42,42,42,42,42,42,42,42,42,44,44,46};

//Derivações bipolares
//I = avL - avR
//II = avF - avR
//III = avF - avL

void setPwmFrequency(int pin, int divisor) {
/**
 * Divides a given PWM pin frequency by a divisor.
 *
 * The resulting frequency is equal to the base frequency divided by
 * the given divisor:
 *   - Base frequencies:
 *      o The base frequency for pins 3, 9, 10, and 11 is 31250 Hz.
 *      o The base frequency for pins 5 and 6 is 62500 Hz.
 *   - Divisors:
 *      o The divisors available on pins 5, 6, 9 and 10 are: 1, 8, 64,
 *        256, and 1024.
 *      o The divisors available on pins 3 and 11 are: 1, 8, 32, 64,
 *        128, 256, and 1024.
**/
  byte mode;
  if(pin == 5 || pin == 6 || pin == 9 || pin == 10) {
    switch(divisor) {
      case 1: mode = 0x01; break;
      case 8: mode = 0x02; break;
      case 64: mode = 0x03; break;
      case 256: mode = 0x04; break;
      case 1024: mode = 0x05; break;
      default: return;
    }
    if(pin == 5 || pin == 6) {
      TCCR0B = TCCR0B & 0b11111000 | mode;
    } else {
      TCCR1B = TCCR1B & 0b11111000 | mode;
    }
  } else if(pin == 3 || pin == 11) {
    switch(divisor) {
      case 1: mode = 0x01; break;
      case 8: mode = 0x02; break;
      case 32: mode = 0x03; break;
      case 64: mode = 0x04; break;
```

```
123         case 128: mode = 0x05; break;
124         case 256: mode = 0x06; break;
125         case 1024: mode = 0x7; break;
126         default: return;
127       }
128       TCCR2B = TCCR2B & 0b11111000 | mode;
129     }
130 }
131
132 void setup()
133 {
134   pinMode(11, OUTPUT); //timer2;
135   pinMode(10, OUTPUT); // timer1;
136   pinMode(9, OUTPUT); //timer1;
137   //pinMode(8, OUTPUT);
138   //pinMode(7, OUTPUT);
139   pinMode(6, OUTPUT); //timer0;
140   pinMode(5, OUTPUT); //timer0;
141   //pinMode(4, OUTPUT);
142   pinMode(3, OUTPUT); //timer2;
143
144   setPwmFrequency(11,64);
145   setPwmFrequency(10,64);
146   setPwmFrequency(9,64);
147
148 }
149
150 void loop()
151 {
152   for(index=0;index<586;index++)
153   {
154     analogWrite(11,avR[index]);
155     delay(1);
156
157     analogWrite(10,avL[index]);
158     delay(1);
159
160     analogWrite(9,avF[index]);
161     delay(1);
162   }
163 }
```

7 Anexo

7.1 Gráfico do sinal de ECG original utilizado

Figura 47 – Sinal de ECG obtido da paciente mencionada na seção 3.2.



Fonte: Página do PTB[1].

[1] <http://www.physionet.org/cgi-bin/atm/ATM> DATABASE:*PTB Diagnostic ECG Database (ptbdb)*, Record: *patient267/s0504_re*