UNIVERSIDADE FEDERAL DO ABC

Trabalho de Graduação em Engenharia Biomédica

Sistema de Aquisição de dados para uso em Miografia por Impedância Elétrica

Aluno: Victor Gonçalves Marques

Professor Orientador: Olavo Luppi Silva

São Bernardo do Campo

2017

Sumário

Resumo

Miografia por impedância elétrica (MIE) é um exame não invasivo para avaliação de músculos esqueléticos, funcionando através da injeção de uma corrente alternada de baixa amplitude e medindo a diferença de potencial entre eletrodos posicionados superficialmente para cálculo de sua impedância. Fatores como idade, nível de contração e doenças neuromusculares afetam os valores de bioimpedância dos músculos, característica esta que é dependente da frequência de estimulação utilizada. O presente projeto desenvolveu um sistema de aquisição de oito canais voltado para MIE, realizando aquisição de potenciais em até 8 canais simultâneos com precisão de 16 bits. A estimulação do músculo é gerada por uma fonte de corrente regulada por tensão, que é multiplexada para as direções longitudinal e transversal às fibras musculares.

Palavras-chave: Miografia por Impedância Elétrica, Músculo, Bioimpedância, Instrumentação Biomédica

1 Introdução

Os primeiros estudos acerca das propriedades elétricas de tecidos biológicos datam do final do século XIII, com Luigi Galvani e seus experimentos com músculos de sapos [1]. A medição adequada de tais propriedades foi no entanto realizada apenas a partir do final do século XIX, com a intenção de compreender os mecanismos de funcionamento de neurônios e tecidos musculares [2].

Os trabalhos na área de eletrofisiologia podem ser divididos em dois grupos: aqueles que medem propriedades elétricas inerentes aos tecidos e aqueles que buscam quantizar a perda de energia nos tecidos quando da aplicação de correntes elétricas externas, avaliando assim a sua integridade, densidade e número de células [2]. Como resultados do primeiro foram criados exames como eletromiografia (EMG), eletrocardiografia (ECG) e eletroencefalografia (EEG) [1, 2], que medem o comportamento elétrico de músculos esqueléticos, do coração e da atividade cerebral, respectivamente. O segundo grupo gerou técnicas como a cardiografia por impedância elétrica, que determina parâmetros cardiodinâmicos, a tomografia por impedância elétrica, voltada para a obtenção de imagens médicas, e a análise de bioimpedância (BIA), estabelecida na clínica como método de avaliação nutricional, buscando estimar as proporções de água, músculos e gordura no corpo como um todo [2, 3].

A miografia por impedância elétrica (MIE) é uma técnica pertencente ao segundo grupo, que utiliza-se de correntes de baixa amplitude e alta frequência aplicadas localmente sobre músculos esqueléticos para avaliar a bioimpedância destes [2]. É uma técnica não invasiva e acessível, capaz de gerar informação de grande relevância clínica, uma vez que, dentre outros fatores, idade, diferentes níveis de contração muscular e doenças neuromusculares podem gerar alterações na impedância dos músculos, embora os mecanismos que regem esse fenômeno ainda não estejam totalmente esclarecidos [4].

A medição em exames de MIE é normalmente realizada com quatro eletrodos posicionados na superfície do músculo de interesse (Fig. 1), sendo dois deles responsáveis pela injeção de corrente e os demais por medir a diferença de potencial entre eles, configuração conhecida como tetrapolar [2]. Essa configuração de eletrodos faz com que a gordura subcutânea e outros tecidos adjacentes influenciem mais nas medidas do que em exames onde a corrente é injetada em um ponto distante da região de medição; essa técnica é no entanto favorecida na literatura por permitir análise de músculos específicos, de sua anisotropia elétrica e por não sofrer interferência de ângulos das articulações [2].

Os eletrodos podem ser posicionados de maneira longitudinal às fibras musculares ou transversalmente a elas, sendo o estudo em diversas direções interessante devido à anisotropia elétrica do músculo, havendo comportamentos diferentes dependendo da direção em que se propaga a corrente [2, 3]. O estudo em ambas as direções simultaneamente foi, no entanto, explorado apenas em [5].

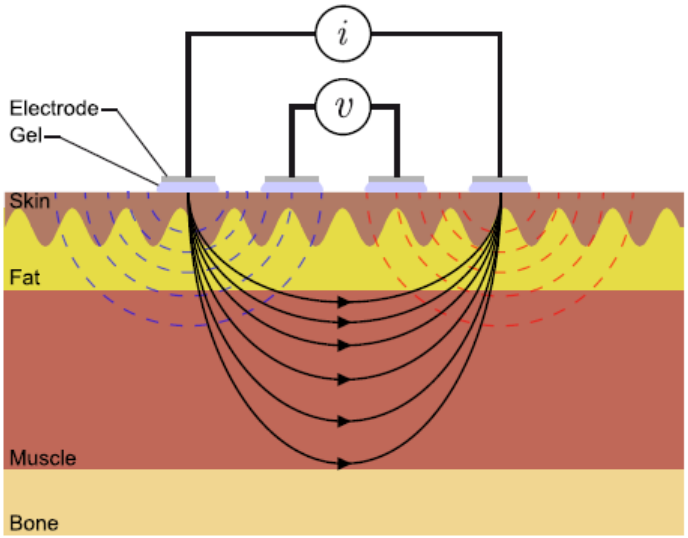
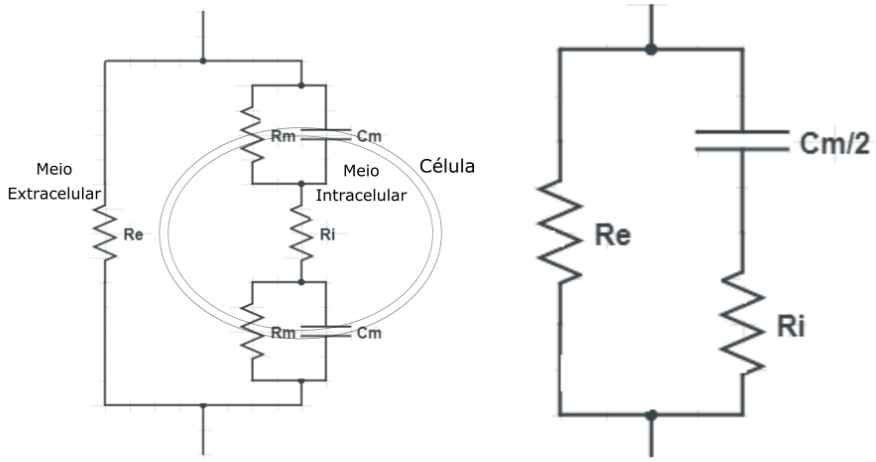


Figura 1: Posicionamento de eletrodos em MIE na configuração quadripolar. Extraído de [3]

Para a correta interpretação dos dados de MIE é necessário que seja feita uma aproximação da bioimpedância a partir de um circuito equivalente que represente o funcionamento do tecido. O comportamento impeditivo dos tecidos biológicos frente a estímulos elétricos pode ser modelado como uma associação de resistores e capacitores, que representam as diferentes partes das células e a matriz extracelular [6]. As células possuem um meio intracelular (citoplasma) envolto por uma membrana composta por uma bicamada fosfolipídica, com canais iônicos ligando os meios intra e extracelulares [7]. O comportamento da bicamada se assemelha ao de um capacitor, separando dois condutores (citoplasma e matriz extracelular) com um material dielétrico (fosfolipídeos) [6]. Já os canais de íons apresentam comportamento similar ao de um resistor, assim como o citoplasma e o meio extracelular. A resistência dos últimos é dependente da concentração de eletrólitos em solução [6]. O circuito equivalente a esse modelo pode ser visto na Fig. 2a; uma versão mais simplificada é apresentada na Fig. 2b, onde todas as resistências da célula, que estão em série, são representadas como apenas um resistor.

A impedância pode ser representada pelo número complexo na Eq. 1, onde R_{eq} representa a resistência equivalente do tecido, composta pela combinação das resistências intra e extra celulares (R_i e R_e respectivamente), e X é a reatância, influenciada apenas pela capacitância equivalente das membranas (C_m). O valor ω é a frequência angular do estímulo. O valor de Z pode ser obtido a partir da lei expandida de Ohm ($V = Z \cdot I$) quando a diferença de potencial e a corrente são conhecidas. A determinação específica dos valores de R_{eq} e C_m depende da fase, obtida pela relação trigonométrica entre resistência e reatância ($\theta = arctan(X/R)$) e que pode ser calculada a partir do atraso entre a corrente de estímulo do circuito e a medida de tensão realizada.



(a) Completo (b) Simplificado

Figura 2: Modelos elétricos de tecidos biológicos

$$Z = R_{eq} + jX = R_{eq} - \frac{j}{\omega \cdot C_m} \tag{1}$$

A equação 1 mostra a dependência que a reatância possui da frequência de estimulação. Em frequências baixas, o capacitor formado pelas membranas celulares é carregado, tornando o fluxo de corrente por dentro das células mais difícil. O comportamento capacitivo se aproxima de um curto circuito em frequências altas, fazendo com que o fluxo de corrente se distribua nos meios intra e extracelular proporcionalmente às suas respectivas resistências (Fig. 3). Esse comportamento pode ser utilizado em exames de bioimpedância de maneira a explorar a porção do tecido que seja de interesse: medições com baixas frequências dão ênfase ao comportamento impeditivo do exterior da célula, enquanto frequências mais altas analisam o tecido como um todo [8].

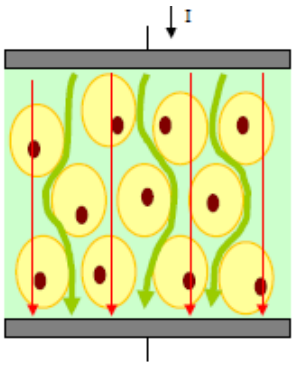


Figura 3: Caminho percorrido pela corrente no tecido em baixa (verde) e alta (vermelho) frequência. Adaptado de [6]

Estudos utilizando MIE foram realizados em geral em uma única frequência de estimulação, a 50 kHz, valor escolhido devido à maior reatividade que os tecidos apresentam neste valor e à maior disponibilidade de equipamentos comerciais de bioimpedância que funcionem nesta configuração [2]. Embora haja alguns equipamentos capazes de aná-

lise multifrequência, o volume de dados gerado torna a avaliação como um todo difícil, sendo ainda necessário desenvolver medidas resumo para obter resultados mais relevantes [2]. Ainda que estudos com diversas frequências englobem frequentemente estímulos abaixo de 10 kHz [9, 10], poucos estudos são voltados para essa faixa, sendo estes focados normalmente em frequências acima de 50 kHz. A exploração do comportamento da bioimpedância muscular a baixas frequências pode revelar aspectos relevantes de doenças que atinjam o tecido muscular, especialmente no que diz respeito à resistividade da matriz extra-celular. A forte dependência da resistência dessa porção do tecido dos eletrólitos em sua solução pode servir de indicador para alguns aspectos das patologias, como por exemplo isquemia no músculo [6].

No presente trabalho, um sistema de aquisição de MIE voltado para análises envolvendo frequências de corrente de até 10 kHz, que seja capaz de realizar medidas simultâneas nas direções longitudinal e transversal às fibras musculares, foi desenvolvido. O sistema tem como objetivo a realização de exames de MIE que busquem explorar o comportamento do tecido muscular do bíceps, com foco na matrix extra-celular, buscando assim elucidar novos aspectos da bioimpedância em músculos normais relaxados e contraídos.

2 Objetivos

Este projeto teve como objetivo a construção de um sistema de aquisição para Miografia por Impedância elétrica (MIE), operando em frequências de estímulo de até 10 kHz. O sistema foi composto por uma fonte de corrente monopolar, uma placa de aquisição de sinais com oito canais e um computador, responsável pelo controle do hardware e pelo processamento dos sinais. Toda a programação foi realizada na linguagem C, funcionando em um sistema Linux.

3 Metodologia

O presente dispositivo foi desenvolvido tendo em mente o posicionamento de eletrodos e o protocolo de aquisição proposto por Morimoto et al [5], com alterações visando endereçar a aquisição em frequências baixas para MIE, entre 5 e 10 kHz. No mencionado trabalho é utilizado um bracelete no qual os eletrodos são fixados em posições padronizadas e que é colocado sobre o bíceps braquial. Há eletrodos posicionados tanto longitudinalmente como transversalmente às fibras musculares do paciente, conforme mostra a Fig. 4.

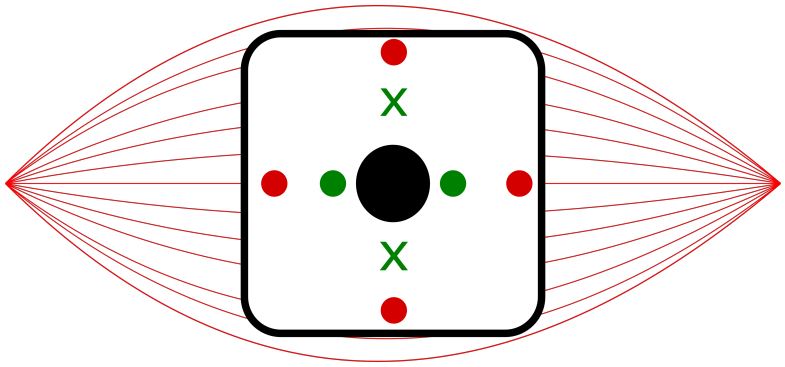


Figura 4: Posicionamento dos eletrodos sobre as fibras musculares – Vermelho: injeção de corrente; Verdes: medição de diferenças de potencial transversais (X) e longitudinais (O) às fibras musculares

Neste trabalho, pequenas alterações foram realizadas no que diz respeito ao posicionamento dos eletrodos e às frequências de estimulação e de amostragem. Ao invés dos 12 eletrodos utilizados em [5], o presente sistema utilizou oito destes, sendo quatro para injeção de corrente e quatro para medição de diferenças de potencial, conforme eletrodos em vermelho e verde na Fig. 4. Além disso, a frequência de amostragem, embora variável, foi configurada tendo em vista um limite superior de 50 kHz.

O sistema é composto por uma fonte de corrente, dois multiplexadores, uma placa de aquisição e um computador, responsável pelo controle do sistema, conforme pode ser visto na Fig. 5. A fonte de corrente é monopolar e controlada por tensão, sendo responsável por injetar correntes de baixa amplitude (menores que 10 mA) e com frequência entre 5 e 10 kHz no paciente. Os multiplexadores direcionam a corrente gerada por esta fonte para os eletrodos correspondentes. A placa de aquisição controla os multiplexadores e realiza a digitalização dos sinais, enviando-os por fim para o computador, que os processa e armazena. A Fig. 6 apresenta um diagrama de blocos do sistema; a seguir, cada um dos blocos será descrito em detalhe, com as configurações utilizadas e a descrição de experimentos relevantes que foram realizados para a melhor compreensão dos componentes de cada etapa. A seção 3.5 descreve em detalhes o protocolo experimental utilizado para a validação do equipamento.

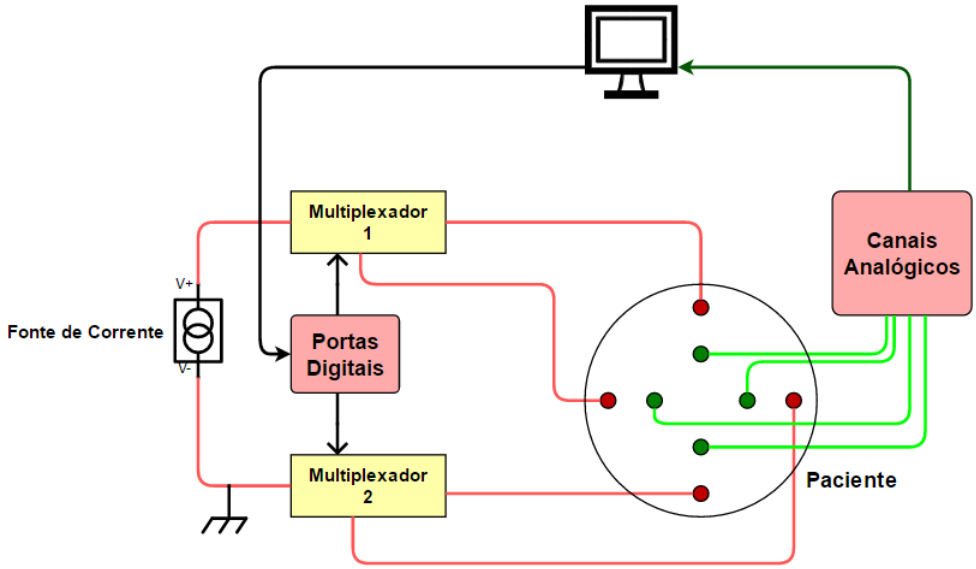


Figura 5: Esquema dos componentes do sistema. Os blocos em vermelho correspondem a partes da placa de aquisição 1608FS-Plus.

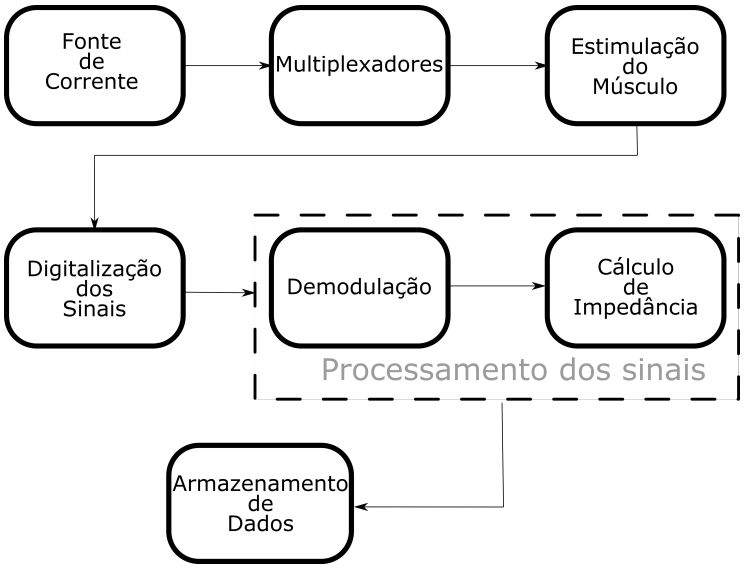


Figura 6: Digrama de blocos do sistema

3.1 Fonte de Corrente

A fonte de corrente foi desenvolvida com base no modelo de Howland modificado (Fig. 7) [11]. Os valores de resistência e de alimentação do circuito foram selecionados com o objetivo de desenvolver uma fonte que fornecesse sinais com amplitude de 5 mA para cargas de até 1500 Ω. O controle do ganho é feito pelo resistor R_{13}, sendo que a corrente na saída é igual à tensão na entrada multiplicada por $1/R_{13}$ Simulações foram desenvolvidas no software LTspice para observar o comportamento do circuito antes da montagem física. O amplificador operacional utilizado foi o LM358 [12].

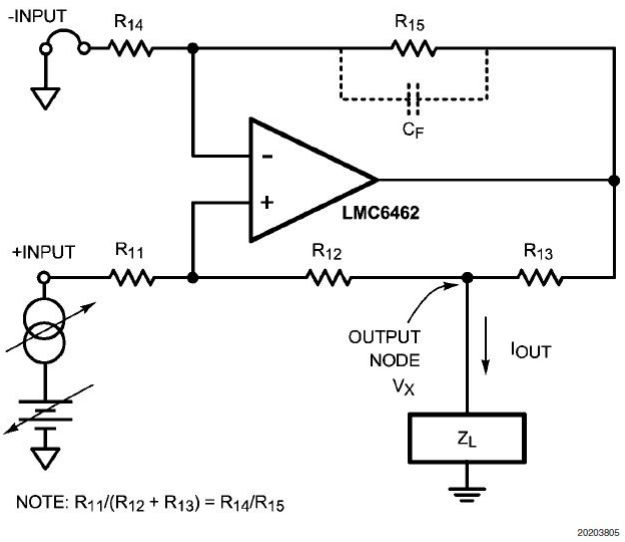


Figura 7: Modelo de fonte de corrente Howland modificada [11]

3.2 Multiplexadores e portas digitais

Os multiplexadores (MUX) utilizados devem ser de configuração *single-pole double-throw* (SPDT), o que corresponde a circuitos que recebam apenas um sinal na entrada, que pode então ser enviado a duas saídas [13]. Foi utilizado um circuito integrado (CI) de modelo CD4052B [14], que possui dois circuitos multiplexadores, com uma entrada comum e quatro saídas possíveis para cada um. Cada um desses MUX foi conectado a um dos polos da fonte de corrente, de maneira que a corrente atravessasse o músculo ora alinhada com as fibras musculares, ora transversal a elas (Fig. 8).

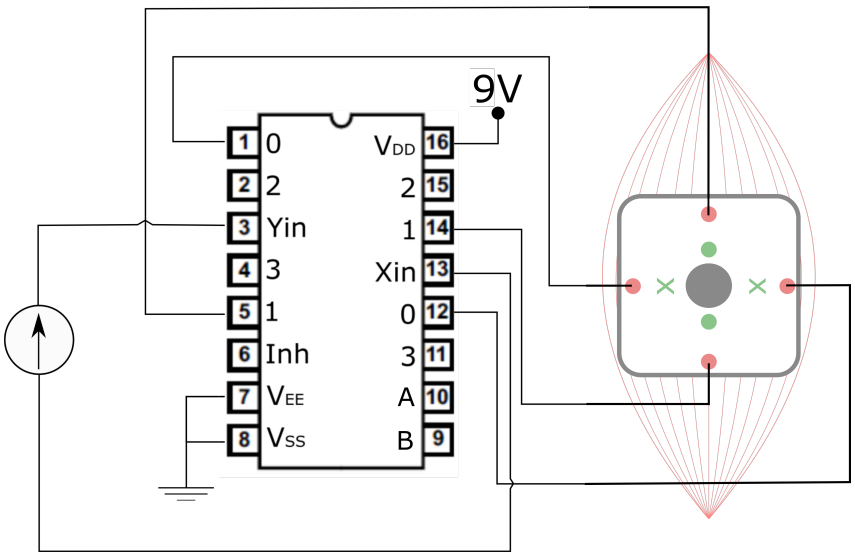


Figura 8: Multiplexador CD4052B e esquema de direcionamento da fonte de corrente. As entradas A e B são ligadas à placa de aquisição para controle do circuito integrado.

Para o controle dos multiplexadores foram utilizadas as portas digitais da placa de aquisição de modelo USB-1608FS-Plus da Measurement Computing (Fig. 9) [15]. Tal

placa de aquisição foi utilizada também nas demais etapas do projeto. As saídas digitais fornecem tensões de 5 ou 0 V, conforme nível lógico 1 ou 0, respectivamente. Duas destas portas foram utilizadas nas entradas A e B do CI para direcionamento da corrente, conforme a tabela 1. Os circuitos integrados foram alimentados com valor de 9 V, fornecidos por uma fonte de tensão contínua (DC, da sigla em inglês para corrente contínua).

Tabela 1: Tabela verdade do multiplexador.

A	B	Output
0	0	0x, 0y
0	1	1x, 1y
1	0	2x, 2y
1	1	3x, 3y

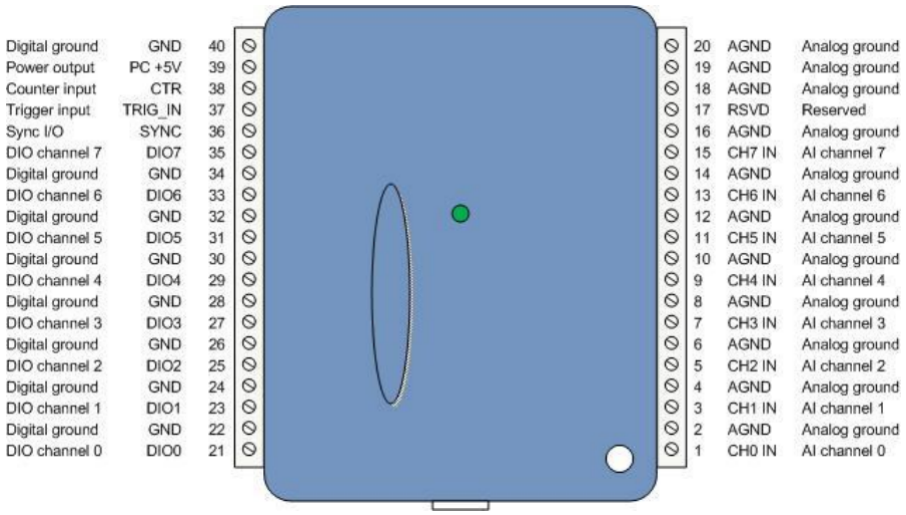


Figura 9: Placa de aquisição USB-1608FS-Plus [15]

De maneira a sincronizar o funcionamento dos MUX e das portas digitais sem a interferência de outros componentes do sistema, o seguinte teste foi proposto: um gerador de sinais de ondas quadradas é conectado na entrada do MUX, com a referência em comum com o circuito integrado. Um único multiplexador foi então utilizado para direcionar a corrente fornecida para um dos dois possíveis circuitos, ambos compostos por um LED e um resistor, através do controle da placa de aquisição, programado computacionalmente. Um resumo do esquema experimental é dado na Fig. 10.

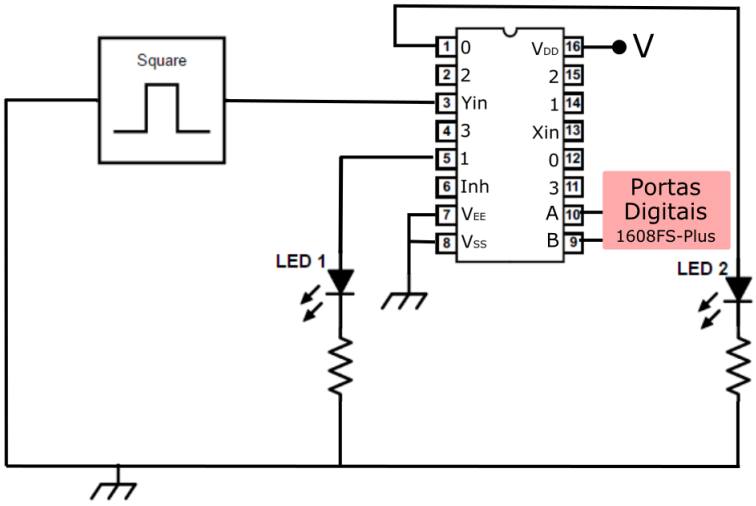


Figura 10: Esquema experimental para teste com multiplexador

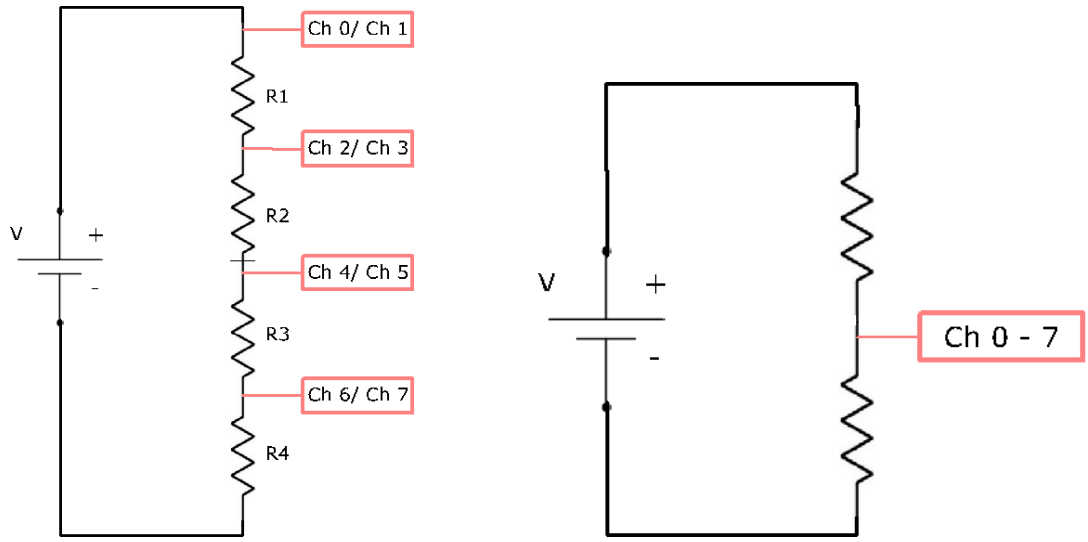
3.3 Portas Analógicas

A placa de aquisição USB-1608FS-Plus possui 8 portas analógicas *single-ended* para a conversão analógico-digital (A/D) de 16 bits dos sinais medidos por elas. Elas operam em intervalos de ± 1 V a ± 10 V, conforme decisão do usuário, com frequência de amostragem de 50 kHz, com todos os canais ativos, a 400 kHz, com apenas um canal. Todas as portas são utilizadas nesse sistema, sendo quatro delas responsáveis pela medição dos sinais longitudinalmente às fibras do músculo e as demais pela aquisição na transversal. Em cada direção, duas portas são utilizadas para medição da queda de potencial em um resistor sentinela, de maneira a obter o valor real de corrente sendo fornecido pela fonte; os demais medem a queda de potencial no músculo.

A aquisição dos sinais foi configurada programaticamente através de um arquivo de configurações fornecido pelo usuário. Nesse arquivo estão contidos todos os parâmetros que sejam relevantes para realização dos protocolos experimentais, como frequência da fonte de corrente, frequência de aquisição, tempo ou número de ciclos para cada chaveamento da fonte de corrente, etc.

É importante garantir que os canais de medição realizem a digitalização e transferência dos sinais para o computador conforme especificações sinalizadas programaticamente para a placa e que a qualidade destes sinais seja avaliada, garantindo que a relação sinal ruído (SNR) seja grande o suficiente. Para realização destas tarefas, dois testes foram propostos: o primeiro consiste na utilização dos canais de aquisição para medir a queda de potencial sobre resistores, com frequências variáveis sendo utilizadas na fonte de alimentação (Fig. 11a). O segundo consiste na medição do SNR dos canais e foi feito com a utilização de um divisor de tensão com dois resistores, sendo todos os canais conectados simultaneamente entre eles (Fig. 11b). São realizadas então medições, ora com a fonte de tensão ligada (sinal) e ora desligada (ruído), para todos os intervalos de medição da placa; os valores

SNR são então calculado a partir de valores médios encontrados. Os sinais foram avaliados por meio dos softwares Matlab 2013 e Microsoft Excel 2010.



(a) Aquisição analógica (b) Relação SNR

Figura 11: Arranjo experimental para testes com os canais analógicos

3.4 Processamento e armazenamento dos sinais

O sistema é controlado por um computador funcionando com um sistema operacional `Linux`, sendo toda a programação da placa de aquisição, que controla os demais componentes, realizada na linguagem C. O fabricante da placa disponibiliza *drivers* e bibliotecas específicas para programação do equipamento neste sistema operacional.

Após a digitalização dos sinais, é necessário extrair dos potenciais medidos os valores de corrente injetada e de impedância do músculo. Para a realização desses cálculos, a amplitude, frequência e fase dos sinais devem ser estimadas; uma vez que a fonte de corrente emite sinais senoidais, os valores medidos também terão a mesma configuração, alterada devido à impedância dos tecidos e dos resistores sentinela, sendo possível então ajustar aos dados uma curva da forma

$$\vec{r}(t_N) = Asin(\omega_0 t_N + \phi) + C \tag{2}$$

onde $\vec{r}(t_N)$ é um vetor contendo os parâmetros de amplitude (A), fase (ϕ) e *offset* (C). A frequência angular ω_0 é conhecida e igual à da fonte de corrente. Esse ajuste é realizado utilizando uma abordagem baseada no método dos quadrados mínimos totais [16]. Uma descrição mais detalhada da dedução do algoritmo para esta finalidade pode ser encontrada no Apêndice A.

A obtenção de parâmetros é realizada para segmentos pequenos do sinal, sincronizados com a duração da aplicação de corrente em cada direção. Após essa etapa, é possível

calcular os valores desejados para cada segmento, tendo como base lei de Ohm (Eq. 1). Para estimar a corrente injetada (I), basta substituir os valores estimados de tensão e a resistência sentinela (R) conhecida; para calcular a impedância do tecido (Z), basta substituir os valores para corrente real injetada e a diferença de potencial medida (V). Vale ressaltar que a tensão e a corrente possuem formas senoidais, sendo possível resolver essa equação com o uso de fasores, caso a frequência se mantenha constante.

Ao fim destes cálculos, os sinais medidos e os valores estimados para as tensões, impedâncias e correntes devem ser armazenados de acordo com a decisão do usuário. Isso foi realizado utilizando diferentes arquivos texto: o primeiro armazena os valores de tensão medidos na placa; o segundo, os valores obtidos na demodulação; o terceiro armazena os valores de impedância do tecido, com uma escala de tempo baseada na frequência com que foram calculados.

3.5 Procedimento Experimental

O sistema foi testado em um voluntário, com a utilização da cinta para fixação de quatro eletrodos para medição de sinais e quatro para injeção de corrente, conforme a Fig. 4. O posicionamento dos eletrodos no voluntário seguiu o protocolo descrito em [5], obedecendo os seguintes passos:

1. Marcar uma linha entre a protuberância acromial e o tendão distal do bíceps braquial;
2. Pedir para que o voluntário realize uma flexão de cotovelo para evidenciar o músculo bíceps braquial;
3. Marcar a posição onde o perímetro do músculo é maior;
4. Marcar o ponto de intersecção entre a linha de maior perímetro do músculo e a linha longitudinal do item 1;
5. Após marcar o ponto de intersecção, centralizar a abertura da cinta sobre ele.

As aquisições de sinal foram realizadas com frequências de estimulação de 1, 5 e 10 kHz, com frequência de amostragem em 25 kHz e 10 chaveamentos da fonte de corrente para cada direção. O número de períodos da corrente senoidal para cada chaveamento foi determinado de modo a totalizar 10 segundos de exame.

As medições foram divididas em dois grupos, conforme atividade muscular: repouso e esforço isométrico. As coletas foram separadas com intervalos para descanso. A coleta em repouso foi realizada com o braço apoiado sobre uma mesa, formando um ângulo de 90 graus na articulação. No exame com esforço isométrico, o voluntário realizou a flexão de antebraço (rosca direta) até o esforço máximo, com auxílio de uma outra pessoa.

4 Resultados

Os resultados são aqui expostos em duas etapas: inicialmente, os dados obtidos com testes realizados para cada componente isoladamente são apresentados. Em seguida, o sistema final é descrito. Em ambos os casos, tanto os circuitos utilizados quanto a programação realizada são apresentados. Por fim, os resultados do procedimento experimental são apresentados.

O desenvolvimento das rotinas para controle da placa de aquisição foram escritas em C com o uso da interface gráfica Geany, disponível para o sistema operacional utilizado (`Linux Ubuntu 16.04 LTS`, com kernel da versão `4.4.0-47-generic`). A programação foi realizada com base em um programa teste disponibilizado pelo fabricante juntamente com os *drivers* e a biblioteca da placa, adaptando o algoritmo para as necessidades do projeto. Todas as funções específicas para esta placa de aquisição possuem o sufixo `_USB1608FS_Plus`, que será omitido para facilitar a leitura ao longo do relatório.

4.1 Testes de componentes individuais

4.1.1 Fonte de corrente

O circuito da Fig. 12 foi desenvolvido após diversos testes com o LTspice para selecionar o melhor conjunto de componentes que e valores adequados para a alimentação do amplificador operacional. Foram utilizados quatro resistores de valor nominal 100 $k\Omega$,com precisão de 5%, além de um resistor de 1 $k\Omega$ de mesma precisão para definir o ganho e um *trimpot* (R11) de resistência máxima 100 $k\Omega$ para balanceamento das proporções adequadas entre os resistores (ver Fig. 7). O valor do *trimpot* foi ajustado utilizando uma ponte de Wheatstone [17]. A alimentação do amplificador LM358 foi definida como ± 9 V.

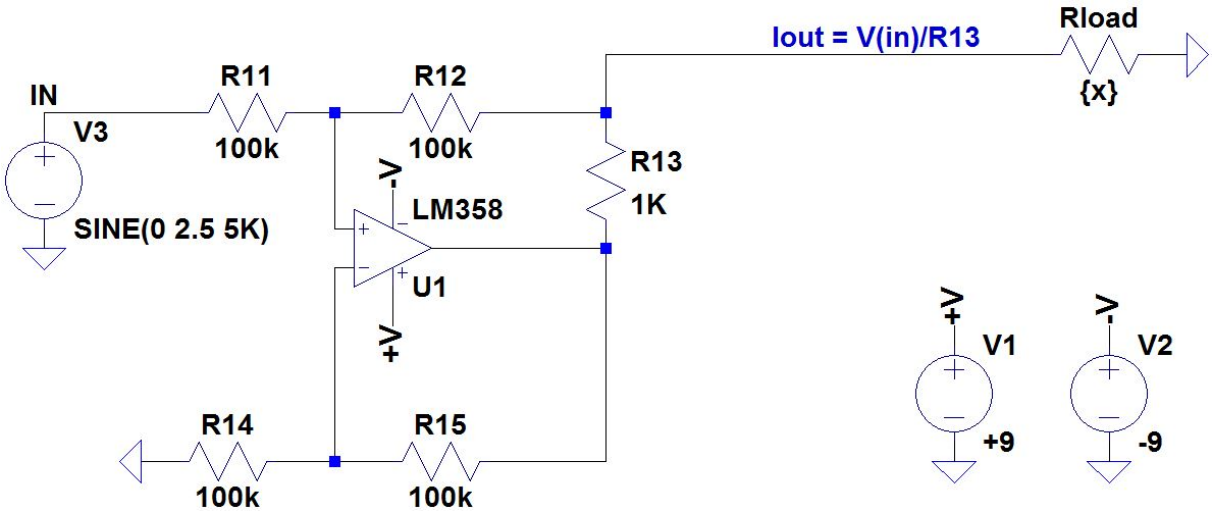


Figura 12: Fonte de corrente desenvolvida

O sinal na fonte de tensão foi obtido com um gerador de sinais da B&K Precision, modelo 4054, sendo o amplificador operacional alimentado com uma fonte de tensão DC de modelo EEL-8006, da Edutec. As medições foram realizadas com um osciloscópio de modelo 2190D, da B&K Precision. A fonte foi testada com resistores de carga com valores entre 46.7 e 3159 Ω, com sinal de entrada senoidal de 1, 2.5 e 5 V pico a pico, resultando nos valores mostrados no gráfico da Fig. 13. As medidas marcadas com um asterisco apareceram saturadas, com as amplitudes limitadas devido à capacidade do amplificador operacional dentro da alimentação fornecida a ele. Os resultados da simulação em LTspice são apresentados na mesma figura.

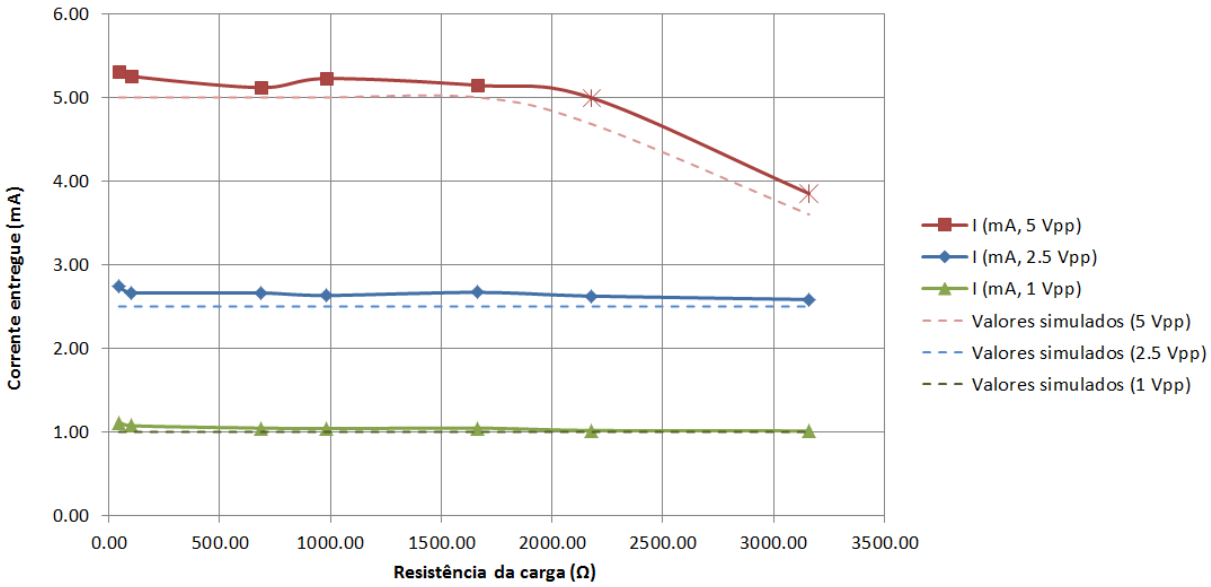


Figura 13: *Output* da fonte de corrente e da simulação em LTspice

4.1.2 Multiplexadores e portas digitais

Para o circuito de teste dos multiplexadores, resistores de valor nominal de 100 Ω foram utilizados em série com LEDs vermelhos. A onda quadrada foi fornecida por meio do mesmo gerador de sinais utilizado anteriormente, a uma frequência de 1 Hz, de modo que fosse possível notar a olho nu o acendimento das luzes. A alimentação do CI dos MUX foi fornecida pela saída de tensão 5 V da placa de aquisição.

Para controlar o acendimento dos LEDs, um número binário de dois bits foi gerado, utilizando as duas primeiras portas digitais. Para a programação destas portas, foi necessário o uso de duas funções da biblioteca da placa: a primeira delas, `usbDTristateW` é responsável por configurar o `tristate`[1] da placa, ou seja, quais das portas serão utilizadas para entrada ou para saída de sinais digitais. A segunda, `usbDLatchW`, é responsável por

[1]Em eletrônica digital, portas lógicas com saídas tristate permitem a geração de valores 0 (*low*), 1 (*high*) ou Z (alta impedância na saída).

configurar o `latch`[2] , ou seja, quais das portas de `output` estarão no estado `high` ou `low`, correspondentes a 5 ou 0 volts respectivamente.

Na rotina desenvolvida, um valor booleano alternado foi enviado à porta A do MUX, enquanto a porta B foi mantida constantemente no valor lógico baixo. Com esse procedimento, foi possível observar o acendimento dos LEDs em cada trilha na ordem desejada.

4.1.3 Aquisição analógica

Em seguida, a aquisição analógica foi testada com a medição de um número finito e pré determinado de amostras para cada canal. Tal modo de operação foi selecionado por ser utilizado também na rotina final, devido ao uso de diversos parâmetros de aquisição que dispensam a necessidade de medições contínuas. O divisor de tensão representado na Fig. 11a foi montado quatro resistores, a saber, 979, 99.0, 99.3 e 99.1 Ω e alimentação DC de aproximadamente 9 V, sendo dois canais de medição conectados a cada nó, exceto naquele que conecta o último resistor à referência das portas. Medições em cada ponto foram realizadas com multímetro para servir com base para comparação.

Para o controle deste tipo de medição, as seguintes funções da placa foram utilizadas: `usbAInScanStart, usbAInScanStop, usbAInScanClearFIFO, usbAInScanConfig, usbAInScanConfigR, usbAInScanRead`.

As funções `usbAInScanStart` e `usbAInScanStop` controlam o início e fim da aquisição de sinais. Como parâmetros para o início, são pedidos uma referência à placa, o número de amostras desejado, a frequência de aquisição, os canais utilizados e uma variável contendo opções da medição, que incluem o modo de transferência de dados para o computador e a configuração do `pacer` interno da placa. Antes de iniciar a aquisição é também necessário configurar os intervalos de tensão (*ranges*) nos quais cada canal realizará a medição; isso é feito através da função `usbAInScanConfig`. Por fim, é importante garantir que a memória interna do dispositivo esteja vazia para a entrada de novos dados; isso é realizado através da função `usbAInScanClearFIFO`. As funções `usbAInScanConfigR` e `usbAInScanRead` são utilizadas para, respectivamente, recuperar as configurações utilizadas na aquisição e para obter o número de bits transferidos, verificando o sucesso da medição.

Os dados foram medidos com todas as configurações possíveis para o *range* da placa, com frequência de amostragem em 200 Hz. A diferença de potencial média de cada canal foi calculada e é apresentada na tabela 2, bem como os valores medidos com o multímetro. O desvio padrão das medidas foi maior nos canais 0 e 1 com o *range* em ±10 V, sendo da ordem de 1.2 mV; os menores desvios foram observados nos canais 5 (*range* ± 2 V) e 7 (*range*±1 V), ambos da ordem de 0.072 mV. Todos os demais desvios padrão se encontram na faixa entre estes dois valores.

[2]Latch é um circuito com dois estados estáveis e que pode ser utilizado para armazenar informações sobre os estados de portas digitais

Tabela 2: Resultados do teste de aquisição analógica. Todas as medidas estão em volts

Canal	Range				Multímetro (V)	Valor nominal (V)
	± 10 V	± 5 V	± 2 V	± 1 V		
0	9.1233 ± 0.0012	-	-	-	9.1000 ± 0.0455	9.1000
1	9.1228 ± 0.0012	-	-	-	9.1000 ± 0.0455	9.1000
2	2.1259 ± 0.0002	2.1252 ± 0.0001	-	-	2.1230 ± 0.0106	2.1203
3	2.1253 ± 0.0002	2.1250 ± 0.0001	-	-	2.1230 ± 0.0106	2.1203
4	1.4192 ± 0.0002	1.4183 ± 0.0001	1.4177 ± 0.0001	-	1.4160 ± 0.0071	1.4145
5	1.4176 ± 0.0003	1.4171 ± 0.0001	1.4167 ± 0.0001	-	1.4160 ± 0.0071	1.4145
6	0.7075 ± 0.0002	0.7076 ± 0.0001	0.7072 ± 0.0001	0.7072 ± 0.0001	0.7070 ± 0.0035	0.7065
7	0.7081 ± 0.0002	0.7075 ± 0.0001	0.7071 ± 0.0001	0.7071 ± 0.0001	0.7070 ± 0.0035	0.7065

4.1.4 Relação sinal ruído das portas analógicas

Resistores de valores $98.9 \pm 0,8$ e $99.0 \pm 0,8\ \Omega$, respectivamente, foram utilizados para montar o circuito da Fig. 11b. A fonte de alimentação DC foi configurada para o valor nominal máximo de cada *range* (i.e. 10, 5, 2 e 1 V) e 100 amostras foram obtidas com frequência de amostragem em 200 Hz. Em seguida, o mesmo foi feito com a fonte desligada. Médias e desvios padrão amostrais dos sinais foram calculados; a relação sinal ruído foi extraída dividindo os valores medidos com a fonte ligada por aqueles medidos com a fonte desligada. Os resultados são mostrados na tabela 3 em decibels.

Tabela 3: Resultados do teste da relação sinal ruído

Canais	SNR por Range (dB)				Média por canal (dB)
	± 10 V	± 5 V	± 2 V	± 1 V	
0	66.52 ± 0.54	62.89 ± 0.68	56.08 ± 0.41	50.15 ± 0.42	85.91 ± 0.52
1	76.64 ± 3.19	83.61 ± 4.68	63.66 ± 0.98	57.14 ± 0.96	70.26 ± 2.45
2	75.03 ± 1.49	75.72 ± 1.93	103.46 ± 104.33	90.85 ± 42.27	86.26 ± 37.51
3	80.91 ± 3.1	85.83 ± 6.45	70.11 ± 2.35	63.29 ± 1.97	75.04 ± 3.47
4	71.67 ± 1.98	72.03 ± 2.28	64.97 ± 1.08	59.19 ± 1.15	66.97 ± 1.62
5	93.88 ± 12.98	73.51 ± 1.43	65.53 ± 1.19	59.35 ± 1.13	73.07 ± 4.18
6	109.66 ± 76.35	92.85 ± 13.09	77.94 ± 5.16	67.93 ± 3.40	87.09 ± 24.5
7	81.27 ± 5.69	89.63 ± 9.72	78.06 ± 4.84	76.26 ± 8.26	81.30 ± 7.13
Média por range (dB)	81.95 ± 13.17	79.51 ± 5.03	72.47 ± 15.04	65.52 ± 7.45	74.86 ± 10.17

4.2 Software de controle do sistema

O funcionamento do *software* de controle do sistema é descrito a seguir. A rotina completa pode ser encontrada no Apêndice C, bem como as respectivas subfunções no Apêndice D. Uma descrição em pseudocódigo é dada abaixo (Algoritmo 1):

Algoritmo 1: Rotina Main

```
1  Importação de bibliotecas;
2  Definição de constantes;
3  início
       Entrada: Arquivo de configuração .cfg;
4      Declaração de variáveis;
5      Inicialização da placa;
6      se Placa não encontrada então
7          Erro;
8      fim
9      Configura tristate
10     Lê arquivo de configuração;
11     se Arquivo não encontrado ou inválido então
12         Erro;
13     fim
14     Determinação das demais configurações com base nos parâmetros fornecidos;
15     Mostra parâmetros da aquisição;
16     Configura placa para aquisição;
17     Início da aquisição:
18     enquanto m < número de chaveamentos faça
19         Configura portas digitais;
20         Coleta amostras correspondentes a uma aplicação;
21         Converte valores para volts;
22         Salva valores em uma matriz geral;
23         Muda valor da porta digital;
24     fim
25     Calcula fasores e impedância;
       Saída: Arquivos .txt com os dados
26     retorna 0;
27 fim
```

Após inicialização das variáveis necessárias, é feita uma verificação da presença da placa. Caso não haja erros, é exibido um *prompt* ao usuário para que selecione um arquivo de extensão `.cfg`. Esse formato é o padrão da biblioteca *LibConfig* para leitura de arquivos de configuração. Nele são contidas informações numéricas ou textuais, associadas a um nome, como uma variável. Para o presente projeto, os parâmetros descritos abaixo são pedidos ao usuário para configurar a aquisição; um exemplo da organização deste tipo de arquivo é dado em seguida:

- Nome: um vetor de caracteres contendo um nome para associação com esse arquivo de configuração. Esse nome é também utilizado para identificar os arquivos salvos no final da execução.
- Número de canais: o usuário tem a opção de utilizar 8 canais, conforme previsto na metodologia, ou apenas 4, caso deseje testar diferentes protocolos.
- *Range*: o usuário seleciona o intervalo de tensões que se espera medir, de acordo com um código numérico. Os valores 0, 1, 3 e 5 correspondem respectivamente a intervalos de ±10, ±5, ±2 e ±1 V.
- Tempo de aquisição: determina o tempo total para o exame realizado.
- Número de chaveamentos da fonte de corrente para cada direção: determina quantas vezes o sinal da fonte de corrente será aplicado nas direções longitudinal e transversal.
- Frequência da fonte de corrente.
- Número de períodos por chaveamento: determina quantos pedíodos da fonte de corrente senoidal serão injetados a cada aplicação, em cada direção.
- Frequência de amostragem: indica a frequência de digitalização dos dados pela placa de aquisição.
- Período de amostragem: intervalo de tempo entre duas amostras.

Organização de arquivo .cfg

```
name = "T_humano_A";

N_chan = 8; // Número de canais, 4 ou 8
range = 5; // Range (0,1,3,5)
acq_time = 10.0;// Tempo de aquisição
cicles_app = 0;// Numero de ciclos da fonte para cada aplicação em cada direção
times_direction = 20; // numero de chaveamentos da fonte para cada direção
f_source = 1000.0; //frequência da fonte
f_sampling = 25.e3;// frequência de amostragem
T_s = 0; // período de amostragem
```

Alguns dos parâmetros acima são excludentes e sobredefinem a aquisição, podendo gerar contradições. É o caso da frequência de amostragem em relação ao período de aquisição e do tempo em relação aos números de períodos da fonte e de chaveamentos. Para evitar contradições, há a possibilidade de escolher o valor 0 para os argumentos que não se deseje calcular. O *software* lê essas informações e decide quais parâmetros devem ser calculados com base na tabela 4 abaixo, onde T significa que o parâmetro foi informado, F significa que não foi informado (i.e. é zero ou `NULL`). Erros são retornados caso informações insuficientes tenham sido fornecidas. No caso da frequência de amostragem e seu período, a preferência sempre é dada à frequência, com cálculo do período caso ambos tenham sido informados, para evitar incongruências.

Tabela 4: Processo de decisão do software para diferentes parâmetros informados

Tempo	Número de chaveamentos	Número de períodos da fonte	Informações Calculadas
T	T	T	Número de chaveamentos
T	T	F	Número de períodos da fonte
T	F	T	Número de chaveamentos
T	F	F	Erro
F	T	T	Tempo
F	T	F	Erro
F	F	T	Erro
F	F	F	Erro

Com as configurações feitas, um sumário é apresentado ao usuário, que pode verificar se algum parâmetro foi alterado. Caso aceite, a aquisição de dados é iniciada. As medições são realizadas em ciclos, com a direção da corrente sendo alterada a cada ciclo por meio de comandos das portas digitais aos MUX. Duas matrizes são utilizadas para o armazenamento dos dados: a primeira lê os valores correspondentes a uma aplicação diretamente da placa, enquanto a segunda armazena os dados de tensão de todas as aplicações, após os números terem sido convertidos para valores em volts com uso de uma função específica para isso na biblioteca da placa. A conversão para volts é realizada em cada ciclo, sendo então armazenada na matriz.

Para facilitar as manipulações matriciais subsequentes, a biblioteca `GSL` da *GNU Scientific Library* foi utilizada. Nela é definido um novo formato que permite acesso rápido a elementos, linhas ou colunas de matrizes, além de possuir funções com algumas operações básicas de álgebra linear. A matriz que armazena todos os dados das medições em volts é definida no formato `gsl_matrix`.

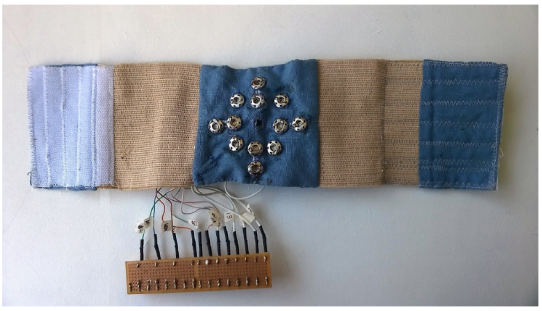
Uma vez em posse dos dados, é possível realizar a demodulação. Isso foi implementado em uma biblioteca separada criada para o projeto, onde também há outras funções gerais

para manipulação e armazenamento de dados, que são apresentadas no apêndice D. Como resultados desta etapa, uma matriz contendo valores de amplitude, fase e *offset* para cada canal e para todos os chaveamentos é retornada. É gerada também uma nova escala de tempo, cujo período é dado por $T = T_F \cdot N_{ciclos}$, onde T_F é o período da fonte de corrente e N_{ciclos} é o número de ciclos (períodos) da fonte por aplicação. Isso foi feito pois apenas um valor de impedância é calculado por chaveamento da fonte.

Por fim, o usuário pode armazenar os dados. Três tipos de arquivo são gerados quando a escolha for afirmativa: o primeiro armazena diretamente a matriz de tensões, com um vetor de valores temporais baseados na frequência de amostragem. O segundo guarda os sinais demodulados, com a mesma escala de tempo, e o terceiro armazena os dados de impedância na nova escala de tempo.

4.3 Procedimento Experimental

A Fig. 14 mostra a cinta para fixação de eletrodos e o posicionamento destes sobre o braço do voluntário. Foram utilizados resistores sentinela de 30 Ω e precisão de 1% para obtenção da corrente real sendo injetada. O *range* utilizado na placa de aquisição para a medição dos sinais foi de ±10 V. Não foi possível realizar a medição em frequências de amostragem maiores que 25 kHz, o que ocasionou um erro por parte da placa de aquisição.



(a) Cinta para posicionamento de eletrodos

(b) Eletrodos posicionados sobre o bíceps com auxílio da cinta

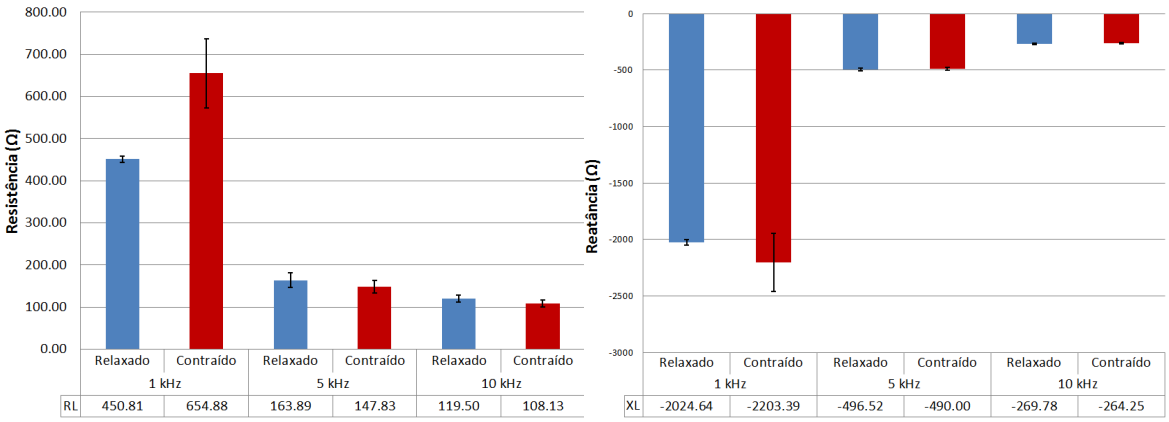
Figura 14: Posicionamento de eletrodos no arranjo experimental

Os sinais foram demodulados e os valores de resistência e reatância foram calculados para as direções longitudinal e transversal, com base nas diferenças de potencial e nas correntes injetadas nas respectivas direções. O gerador de sinais foi configurado para alimentar a fonte de corrente com tensão de 2.5 V pico a pico, o que resulta no valor nominal de 2.5 mA para a injeção de corrente. Os valores reais das correntes alternadas injetadas nas direções transversal e longitudinal, para os músculos relaxados e contraídos, são apresentados na tabela 5, com o respectivos desvios de seu valor nominal.

Tabela 5: Corrente real injetada no voluntário e desvio do valor nominal

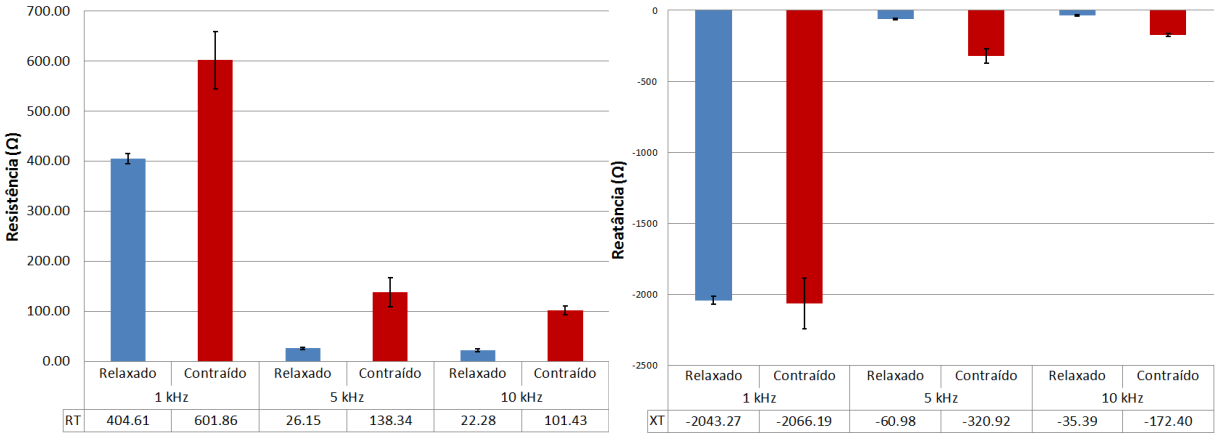
	Frequência (kHz)	Corrente injetada (mA) Músculo Relaxado	Corrente injetada (mA) Músculo Contraído	Desvio do valor nominal (2.5 mA)
Longitudinal	**1**	1.2582 ± 0.0071	0.9714 ± 0.0076	55.41%
	5	2.4824 ± 0.0247	2.4824 ± 0.0247	0.70%
	10	2.4912 ± 0.0173	2.5044 ± 0.0162	0.09%
Transversal	**1**	2.0564 ± 0.0036	2.0526 ± 0.004	17.82%
	5	2.517 ± 0.0179	2.517 ± 0.0179	0.68%
	10	2.5306 ± 0.0109	2.5112 ± 0.0192	0.84%

Os valores obtidos para resistência (R) e reatância (X) são apresentados na Fig. 15 para a direção longitudinal e na Fig. 16 para a direção transversal:



(a) Resistência (b) Reatância

Figura 15: Impedâncias longitudinais no músculo a 1, 5 e 10 kHz



(a) Resistência (b) Reatância

Figura 16: Impedâncias transversais no músculo a 1, 5 e 10 kHz

5 Discussão

Neste trabalho, um sistema de aquisição para miografia por impedância elétrica foi elaborado, com desenvolvimento de uma fonte de corrente e de circuito multiplexador, além de uso de uma placa de aquisição comercial e processamento do sinal medido de acordo com os padrões para esse tipo de exame. O sistema foi capaz de aplicar corrente na direção longitudinal e transversal às fibras do músculo bíceps braquial, com uso de uma braçadeira para fixação de eletrodos.

A fonte de corrente desenvolvida foi capaz de entregar correntes com precisão média de 1% para cargas de até 1664 Ω, com um desvio do valor nominal da fonte de 1.04%. Acima dessa faixa de resistência, os erros foram crescentes, especialmente para o valor de corrente de 5 mA. Quando utilizada no sistema final, a corrente entregue apresentou um desvio grande apenas no caso de estimulação a 1 kHz. Isso pode ter sido causado pela combinação da resistência interna do multiplexador, que pode atingir valores de até 530 Ω com a alimentação utilizada [14], com a impedância do músculo nesta faixa de frequência, que atingiu valores de resistência de até cerca de 655 Ω e de reatância de até aproximadamente -2200 Ω. Não foi observada diferença significativa entre as correntes aplicadas com o músculo relaxado e contraído, mesmo com a diferença de impedância observada, exceto no caso de estimulação a 1 kHz. As diferenças entre as correntes aplicadas longitudinalmente e transversalmente não excederam o valor de 0.04 mA para estímulos nas frequências de 5 e 10 kHz.

Esses resultados para a fonte de corrente não são ideais, uma vez que o cálculo da impedância depende diretamente da corrente real injetada, que deve ser exata e dentro de faixas específicas. No entanto, o uso de resistores sentinela permitiu a obtenção do valor verdadeiro da corrente entrando no músculo, com base na queda de potencial sobre este componente, permitindo o uso satisfatório do sistema. O uso de um *trimpot* para o balanceamento entre todas as resistências não é recomendado, devido a possíveis imprecisões do seu ajuste, e idealmente resistores de precisão superior a 1% devem ser utilizados [11]. A proporção das resistências do circuito da fonte de corrente Howland e o valor do resistor de ganho são de grande importância para seu funcionamento correto [11], sendo possíveis causas para a menor acurácia da fonte desenvolvida. A substituição dos componentes mencionados em versões futuras da fonte desenvolvida podem garantir melhores resultados.

De maneira geral, o *hardware* desenvolvido foi suficiente para a aquisição de dados sem ruídos significativos que impedissem a avaliação dos dados obtidos. Com exceção do canal 2 funcionando no *range* de ± 2 V, todos os canais apresentaram SNR superior a 30 dB, o que é desejável em instrumentos de medição. A SNR ruim neste canal pode ter sido causada por erro na execução do experimento, uma vez que os demais dados medidos nesta configuração apresentaram qualidade similar à dos outros canais. A frequência de

amostragem se mostrou mais limitada do que o esperado, uma vez que não foi possível realizar aquisições mais rápidas que 25 kS/s. Uma possível causa para isso é a presença apenas de portas USB versão 1.1 no computador utilizado, o que pode limitar a velocidade do sistema [15].

A evolução natural do sistema leva à necessidade de utilização de uma placa de circuito impresso em substituição ao uso da *protoboard* para melhor organização dos componentes e otimização da relação sinal-ruído. Desenvolvimentos futuros do sistema também devem ser voltados para a portabilidade deste, tornando possível o seu uso de maneira versátil em um ambiente hospitalar real. Devem ser então utilizados um circuito gerador de sinais em conjunto com a fonte de corrente e um microcontrolador para coordenar os diversos elementos do sistema, incluindo a placa de aquisição, e possivelmente armazenar os dados em memória *flash* ou similar. A alimentação dos componentes deve ser feita também de maneira independente de uma fonte DC, com uso da rede elétrica ou baterias.

O *software* para o sistema de aquisição desenvolvido foi capaz de realizar o controle da placa de conversão A/D e o multiplexador de maneira coordenada e precisa, além de armazenar os dados e processá-los em *offline*. Foi possível obter valores de resistência e reatância para o músculo no intervalo entre os eletrodos com base no processo de demodulação implementado. A execução do programa é adaptável ao usuário e se utiliza de um mecanismo simples e condensado para configurar o sistema, que permite acesso futuro às configurações utilizadas nas medições. A aquisição funciona na faixa de 100 Hz até 25 kHz, realizando a transferência de dados para o computador sem perda de informação.

A rotina para controle dos exames de MIE foi desenvolvida tendo em mente um usuário familiarizado com o procedimento de multiplexação da fonte de corrente e com conceitos de frequência e período de ondas senoidais. Além disso, deve estar acostumado com a execução de programas em *prompt* de comando. Esse não é necessariamente o perfil de operadores em potencial do sistema, que podem ser profissionais da saúde sem formação em engenharia ou cursos similares. Dessa forma, seria interessante fornecer opções de configurações mais simples, padronizadas para exames de MIE. A opção de modificar tais configurações seria da mesma forma fornecida a usuários que dominem os conceitos supracitados. A utilização de uma interface gráfica também tornaria o sistema mais amigável, além de servir ao propósito de verificação de medidas antes de salvar os dados medidos, com uso de janelas contendo gráficos, permitindo o descarte de medidas excessivamente ruidosas imediatamente.

O *software* foi pensado dentro do conceito de uma arquitetura monolítica, composta por uma única rotina principal e módulos separados que são executados dentro de uma sequência determinada quando requeridos. Tal modelo foi escolhido devido ao caráter individual do desenvolvimento da rotina, realizada apenas pelo autor. A utilização de arquiteturas mais complexas pode no entanto gerar grandes melhoras nos atributos do *software*, como sua disponibilidade, performance, modificabilidade, escalabilidade e usa-

bilidade, entre outros [18], e deve ser prioridade em desenvolvimentos futuros. Uma arquitetura com estrutura de cliente-servidor já permitiria, por exemplo, processamento, armazenamento e exibição de dados em tempo real.

6 Conclusão

Foi possível desenvolver um sistema de aquisição voltado para uso em miografia por impedância elétrica, versátil e adaptável ao usuário ao mesmo tempo em que padroniza as medições e garante resultados precisos. Embora haja muita margem para melhoras, tanto no âmbito de *hardware* quanto de *software*, o sistema foi capaz de executar a tarefa para o qual foi planejado.

O presente trabalho pode servir de base para futuros estudos em MIE a baixas frequências, podendo revelar características de patologias que se manifestem de maneira mais pronunciada na matriz extracelular dos tecidos musculares. Desenvolvimentos futuros do sistema podem contribuir ainda mais para a melhor compreensão das características de bioimpedância dos tecidos musculares, através de uma interface mais robusta e sinais mais precisos.

Referências

[1] O. G. Martinsen and S. Grimnes, *Bioimpedance and bioelectricity basics*. Academic press, 2011.

[2] S. B. Rutkove, "Electrical impedance myography: background, current state, and future directions," *Muscle & nerve*, vol. 40, no. 6, pp. 936–946, 2009.

[3] B. Sanchez and S. B. Rutkove, "Electrical impedance myography and its applications in neuromuscular disorders," *Neurotherapeutics*, pp. 1–12, 2016.

[4] J. Li, M. Jafarpoor, M. Bouxsein, and S. B. Rutkove, "Distinguishing neuromuscular disorders based on the passive electrical material properties of muscle," *Muscle & nerve*, vol. 51, no. 1, pp. 49–55, 2015.

[5] L. Y. Morimoto, T. B. R. Santos, H. Tanaka, and O. L. Silva, "Electrical impedance myography (EIM) measurement standardization for biceps brachii muscle," *XXV Congresso Brasileiro de Engenharia Biomédica*, 2016.

[6] A. Ivorra, "Bioimpedance monitoring for physicians: an overview," *Centre Nacional de Microelectrònica Biomedical Applications Group*, pp. 1–35, 2003.

[7] B. Alberts, D. Bray, K. Hopkin, A. Johnson, J. Lewis, M. Raff, K. Roberts, and P. Walter, *Fundamentos da biologia celular*. Artmed Editora, 2006.

[8] R. Bayford, "Bioimpedance tomography (electrical impedance tomography)," *Annu. Rev. Biomed. Eng.*, vol. 8, pp. 63–91, 2006.

[9] L. Li, H. Shin, X. Li, S. Li, and P. Zhou, "Localized electrical impedance myography of the biceps brachii muscle during different levels of isometric contraction and fatigue," *Sensors*, vol. 16, no. 4, 2016.

[10] C. T.-S. Ching, Y.-C. Chen, L.-H. Lu, P. F. Hsieh, C.-S. Hsiao, T.-P. Sun, H.-L. Shieh, and K.-M. Chang, "Characterization of the muscle electrical properties in low back pain patients by electrical impedance myography," *PloS one*, vol. 8, no. 4, p. e61639, 2013.

[11] R. A. Pease, "A comprehensive study of the howland current pump," *National Semiconductor. January*, vol. 29, 2008.

[12] Texas Instruments, *LMx58-N Low-Power, Dual-Operational Amplifiers*, 12 2014.

[13] Sparkfun, "Switch basics." `https://learn.sparkfun.com/tutorials/switch-basics/poles-and-throws-open-and-closed`. [Online; acesso 03/Mar/2017].

[14] Texas Instruments, *CD405xB CMOS Single 8-Channel Analog Multiplexer/Demultiplexer With Logic-Level Conversion*, 8 2015.

[15] Measurement Computing, *USB-1608FS-Plus User's Guide*, 2014.

[16] J. Q. Zhang, Z. Xinmin, H. Xiao, and S. Jinwei, "Sinewave fit algorithm based on total least-squares method with application to adc effective bits measurement," *IEEE transactions on Instrumentation and Measurement*, vol. 46, no. 4, pp. 1026–1030, 1997.

[17] J. G. Webster, "Medical instrumentation-application and design.," *Journal of Clinical Engineering*, vol. 3, no. 3, p. 306, 1978.

[18] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice (2nd Edition)*. Addison-Wesley Professional, 2003.

Apêndices

A Demodulação senoidal

Considerando-se os dados digitalizados $s(t_N)$ e o valor estimado $r(t_N)$ são compostos por N amostras, tem-se que o erro entre os mesmos é dado por

$$e(t_N) = s(t_N) - r(t_N) \tag{A.1}$$

onde t_N é o enésimo instante de tempo onde o sinal senoidal foi amostrado. O vetor $s(t_N)$ é um vetor de números, já o vetor $r(t_N)$ contém os parâmetros A (amplitude), θ(fase) e C (offset) que devem ser estimados para que $r(t)$ se aproxime ao máximo de $s(t)$ para todo t. Portanto r(t) pode ser escrito como:

$$\vec{r}(t_N) = A \cdot sin(\omega_0 t_N + \theta) + C \tag{A.2}$$

$$t_N = N\frac{1}{f_s} \tag{A.3}$$

$$\omega_0 = 2\pi f_0 \tag{A.4}$$

onde f_s é a frequência de amostragem e f_0 é a frequência angular. Assim, reescrevendo $\vec{r}(t_N)$ para todos os N termos, a partir da relação trigonométrica, $sin(a \pm b) = sin(a) \cdot cos(b) \pm cos(a) \cdot sin(b)$, tem-se:

$$\vec{r}(t_N) = A(sin(\omega_0 t_N) \cdot cos(\theta)) + A(cos(\omega_0 t_N) \cdot sin(\theta)) + C \tag{A.5}$$

reescrevendo A.5 em sua forma matricial, tem-se:

$$\vec{r}(t_N) = [sin(\omega_0 t_N)t_N \quad cos(\theta)t_N \quad 1]_{Nx3} \begin{bmatrix} Acos(\theta) \\ Asin(\theta) \\ C \end{bmatrix}_{3x1} = [E]\vec{p} \tag{A.6}$$

onde, a matriz $[E] \in \mathbb{R}_{Nx3}$ é denominada como matriz de demodulação e o vetor $\vec{p}$ de parâmetros a serem estimados para obtenção da amplitude e fase do sinal. A eq. A.1 é válida para todos os instantes de tempo e o erro quadrático de $e(t_N)$ é dado por:

$$||e(t_N)||^2 = \vec{s}(t_N) - \vec{r}(t_N)^T \vec{s}(t_N) - \vec{r}(t_N) \tag{A.7}$$

Substituindo A.6 em A.7, temos:

$$||e(t_N)||^2 = (\vec{s}(t_N) - [E]\vec{p})^T(\vec{s}(t_N) - [E]\vec{p}) \tag{A.8}$$

Para estimar os parâmetros em $\vec{p}$, deve-se minimizar o erro quadrático através do cálculo da primeira derivada da função em relação ao vetor de parâmetros, resultando em:

$$[E]\vec{p} = \vec{s}(t_N) \tag{A.9}$$

sendo este um sistema linear de m equações e n incógnitas. A solução de mínimos quadrados de norma mínima para $\vec{p}$ será:

$$\vec{p} = [E]^+ \cdot \vec{s}(t_N) \tag{A.10}$$

onde $[E]^+$ é a pseudoinversa, também conhecida como inversa generalizada de Monroe-Penrose. Portanto, é possível encontrar os parâmetros de $\vec{p}$ por meio da multiplicação da pseudoinversa $[E]^+$ pelo vetor de dados digitalizados $\vec{s}$.

B Estruturas criadas para o *software*

Duas estruturas foram criadas para a comunicação de dados entre as diversas funções da rotina criada. A primeira, denominada `fitSine`, é responsável por armazenar dados resultantes das demodulações:

```
typedef struct {
        gsl_vector *amplitude;// Amplitude do seno modulado para
           cada canal
        gsl_vector *phase_rad;// Fase
        gsl_vector *offset;// offset
        gsl_vector *alpha;//
        gsl_vector *beta;//
        gsl_vector *omegat;//
        gsl_matrix *y;//
}fitSine;
```

A segunda estrutura (`Sys_Results`) armazena resultados referentes ao sistema como um todo, em quatro matrizes: a primeira contém os dados de tensão obtidos diretamente da placa de aquisição; a segunda armazena os resultados da demodulação por cada canal; a terceira armazena os fasores calculados com base nos parâmetros obtidos da demodulação; a quarta armazena as impedâncias calculadas com base nos fasores.

```
typedef struct {
        gsl_matrix *data;//
        gsl_matrix *demod_data;
        gsl_matrix *phasor_data;
        gsl_matrix *impedance_data;
}Sys_Results;
```

C Rotina *Main*

```
#include <stdlib.h>
#include <time.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <ctype.h>
#include <math.h>
#include <libconfig.h>
#include "../pmd.h"
#include "../usb-1608FS-Plus.h"
#include "Library/mylib.h"

#define MAX_COUNT      (0xffff)
#define FALSE 0
#define TRUE 1

int main(int argc, char **argv){

        /*Declaracao de Variaveis*/

        // Variaveis do sistema

        libusb_device_handle *udev = NULL; //ponteiro para o
           dispositivo
        int ret; //parametro para saida de funcoes que avaliam o
           bom funcionamento da placa
        float table_AIN[NGAINS_USB1608FS_PLUS][NCHAN_USB1608FS_PLUS
           ][2]; //Tabela de calibracao (ranges x canais x 2[
           inclinacao e offset ])

        //Outras variaveis
        config_t cfg_params;
        char cfg_file[100];

        double f_source, f_sampling,acq_time,T_s; //Frequencia da
           fonte, frequencia de amostragem, tempo de aquisicao,
           periodo de aquisicao

```

```
34          int samples_cicle, N_samples,n_loops,range, cicles_time,
               times_direction,N_chan; /* # amostras por ciclo, #
               amostras total, numero de aquisicoes, #ciclos por
               direcao, #numero de aplicacoes por direcao, Numero de
               canais */
35          int col = 0; //controle de loops
36          uint8_t ranges[8],MUX_door = TRUE, options; //armazena
               todos os intervalos de medicao, porta onde esta o MUX,
               opcoes de aquisicao
37          uint8_t channels = 0xff; // Canais a serem usados
38
39          gsl_matrix *Data_Out,*electrode_pairing;
40
41          Sys_Results Output;
42
43          /*Inicializacao da placa*/
44          udev = NULL;
45          ret = libusb_init(NULL);
46          if(ret<0){
47                  perror("Falha ao iniciar a libusb!");
48                  exit(1);
49          }
50          if((udev = usb_device_find_USB_MCC(USB1608FS_PLUS_PID,NULL)
               )){
51                  printf("Dispositivo encontrado!\n");
52          }else{
53                  printf("Falha, placa USB 1608FS-Plus nao encontrada
                       !\n");
54                  return 0;
55          }
56
57          usbBuildGainTable_USB1608FS_Plus(udev, table_AIN); //obtem
               parametros de calibracao
58
59          usbDTristateW_USB1608FS_Plus(udev,0xfe); /*Configura quais
               portas digitais sao entradas ou saidas. 0 = saida, 1 =
               entrada Ler como binario; Configuracao do MUX*/
60
61          printf("\n-----------------MIOGRAFIA POR IMPEDANCIA
               ELETRICA----------------- \n\n ");
62
```

```
63
64          /*Parametros da aquisicao USAR ARQUIVO DE CONFIGURACAO*/
65
66
67          config_init (&cfg_params);
68          printf("Enter .cfg file name/path:\n");
69          scanf("%123s",cfg_file);
70          strcat(cfg_file,".cfg");
71
72          /* Read the file. If there is an error, report it and exit.
                */
73          if(! config_read_file(&cfg_params, cfg_file))
74          {
75          fprintf(stderr, "%s:%d - %s\n", config_error_file(&
                cfg_params),
76                                  config_error_line(&cfg_params),
                                       config_error_text(&cfg_params));
77          config_destroy(&cfg_params);
78          return(EXIT_FAILURE);
79          }
80
81          /*Decision Tree para determinar tipo de configuracao*/
82
83          if(config_lookup_float(&cfg_params,"f_sampling",&f_sampling
                )==0.0){
84                //Configuracao usando periodo
85                config_lookup_float(&cfg_params,"T_s",&T_s);
86                f_sampling = 1/T_s;
87
88          }else{
89                //Configuracao usando frequencia de amostragem
90                T_s = 1/f_sampling;
91          }
92
93          config_lookup_float(&cfg_params,"f_source",&f_source); //
                frequencia da fonte
94
95          if(config_lookup_float(&cfg_params,"acq_time",&acq_time) !=
                0){
96
97                N_samples = acq_time*f_sampling;
```

```
98                  config_lookup_int(&cfg_params,"cicles_time",&
                        cicles_time); //Da erro se colocado direto no if
99
100                 if(cicles_time == 0){
101                         //Configuracao com times_direction e
                                acq_time
102                         config_lookup_int(&cfg_params,"
                                times_direction",&times_direction);
103                         samples_cicle = (int)round(N_samples/
                                times_direction);
104                         cicles_time = round(N_samples/(
                                samples_cicle*2));
105                 }else{
106                         //Configuracao com cicles_time e acq_time
107                         samples_cicle = round(N_samples/(
                                cicles_time*2));
108                         times_direction = round(N_samples/
                                samples_cicle);
109                 }
110         }else{
111                 //Configuracao com cicles_time e times_direction
112                 config_lookup_int(&cfg_params,"cicles_time",&
                        cicles_time);
113                 config_lookup_int(&cfg_params,"times_direction",&
                        times_direction);
114                 N_samples = times_direction*2*(cicles_time*
                        f_sampling/f_source);
115                 samples_cicle = round(N_samples/(cicles_time*2));
116                 acq_time = N_samples*T_s;
117         }
118         /*colocar verificacoes para erros*/
119
120
121         printf("\n\nSumario da Aquisicao \n----------------------\n
                \n");
122
123         printf("Frequencia de Amostragem [Hz]: %0.2f \n",f_sampling
                );
124         printf("Periodo de Amostragem [ms]: %0.2f \n",T_s*1000);
125         printf("Frequencia da Fonte [Hz]: %0.2f \n",f_source);
126         printf("Numero total de amostras: %i \n",N_samples);
```

```
127         printf("Ciclos da fonte para cada aplicacao em cada direcao
                : %i \n",cicles_time);
128         printf("Numeros de aplicacoes em cada direcao: %i \n",
                times_direction);
129         printf("Numero de amostras coletadas em cada aplicacao: %i
                \n",samples_cicle);
130         printf("Tempo estimado de aquisicao [s]: %0.2f \n",acq_time
                );
131
132         /*Preparacao da aquisicao*/
133
134         n_loops = N_samples/samples_cicle; //Numero de repeticoes
                das medicoes
135         config_lookup_int(&cfg_params,"range",&range);
136         config_lookup_int(&cfg_params,"N_chan",&N_chan);
137         for(int i =0;i<N_chan;i++){ranges[i] = range;}
138
139         usbAInScanStop_USB1608FS_Plus(udev); //Para qualquer scan
                que esteja ocorrendo
140         usbAInScanClearFIFO_USB1608FS_Plus(udev);// Limpa o
                endpoint do FIFO (ie. limpa a fila?)
141         usbAInScanConfig_USB1608FS_Plus(udev, ranges);// Configura
                os ranges de cada canal
142         sleep(1);
143
144         uint16_t sdataIn[N_chan*samples_cicle]; //Reserva espaco
                para os dados de entrada
145         float sdataOut[N_chan*N_samples]; // Reserva espaco para os
                 dados de saida
146         Data_Out = gsl_matrix_alloc(N_samples,N_chan); // Reserva
                espaco para os dados de saida em GSL
147
148         //Configura o modo de transferencia dos dados para o pc
149         if (f_sampling < 100.) {
150           options = (IMMEDIATE_TRANSFER_MODE | INTERNAL_PACER_ON);
                //problema nessa opcao
151         } else {
152           options = (BLOCK_TRANSFER_MODE | INTERNAL_PACER_ON);
153         }
154
155         /*Aquisicao*/
```

```
156         printf("\nInicio da Aquisicao\n");
157         for(int m=0;m<n_loops;m++){
158                 //1) Manda informacao para a fonte com MUX
159                 usbDLatchW_USB1608FS_Plus(udev, MUX_door);
160                 //2) Coleta dados pelo tempo do frame
161                 usbAInScanStart_USB1608FS_Plus(udev, samples_cicle,
                        f_sampling, channels, options);
162                 ret = usbAInScanRead_USB1608FS_Plus(udev,
                       samples_cicle, N_chan, sdataIn, options);
163
164                 //3) Salva na matriz depois de converter pra volts
                       (Ideal seria fazer isso so no fim) ou
                       multithread!
165                                 // Outra opcao seria so copiar a
                                       tabela para a tabela maior,e
                                       depois converter. Deve gastar
                                       menos tempo
166                 for(int i = 0;i<samples_cicle*N_chan;i++){
167                         sdataOut[m*samples_cicle*N_chan+i] =
                                volts_USB1608FS_Plus(

                                rint(sdataIn[i]*table_AIN[range][(
                                uint8_t)col][0] + table_AIN[range][(
                                uint8_t)col][1]),range); /*Ajusta
                                valores de acordo com calibracao e
                                converte para Volts*/
168                         col++;
169                         if(col==N_chan){col = 0;}
170                 }
171                 //4) Muda direcao
172                 MUX_door = !MUX_door;
173         }
174         printf("\nFim da Aquisicao\n");
175
176         //matriz com pareamento dos eletrodos
177         // Duas primeiras linhas: DDP musculo (L,T)
178         // Duas ultimas linhas: DDP sentinela (L,T)
179
180         electrode_pairing = gsl_matrix_alloc(4,2);
181
```

```
182         gsl_matrix_set(electrode_pairing,0,0,1); gsl_matrix_set(
               electrode_pairing,0,1,0); // [1 0]
183         gsl_matrix_set(electrode_pairing,1,0,2); gsl_matrix_set(
               electrode_pairing,1,1,3); // [2 3]
184         gsl_matrix_set(electrode_pairing,2,0,4); gsl_matrix_set(
               electrode_pairing,2,1,5); // [4 5]
185         gsl_matrix_set(electrode_pairing,3,0,6); gsl_matrix_set(
               electrode_pairing,3,1,7); // [6 7]
186
187         /*Espaco para demodulacao*/
188         pointer2gsl_matrix(Data_Out,sdataOut,N_samples,N_chan);
189
190         Output = calculateImpedance(Data_Out,samples_cicle,
               electrode_pairing,f_source,f_sampling);
191
192         /*Salvar em arquivos .txt*/
193         //ver se usuario quer salvar
194         saveFile_gsl(cfg_params,Data_Out, 0);
195         saveFile_gsl(cfg_params,Output.impedance_data, 1);
196         saveFile_gsl(cfg_params,Output.phasor_data, 2);
197
198         config_destroy (&cfg_params);
199
200         return 0;
201 }
```

D Subfunções

D.1 Converter para vetor GSL

```
1  int pointer2gsl_array(gsl_vector * gsl_vector, float* M){
2          int m = gsl_vector->size;
3
4          for(int i=0;i<m;i++){
5                  gsl_vector_set(gsl_vector,i,(double)M[i]);
6          }
7          return 1;
8  }
```

D.2 Converter para matriz GSL

```
1  int pointer2gsl_matrix(gsl_matrix * gsl_M, float* M, int m, int n){
2          double element;
3          for(int i = 0;i<m;i++){
```

```
                for(int j = 0;j<n;j++){
                        element = (double)M[i*n+j];
                        gsl_matrix_set(gsl_M,i,j,element);
                }
        }
        return 1;
}
```

D.3 Cálculo de fasores e impedância

```
Sys_Results  calculateImpedance(gsl_matrix* Data, int samples_cicle
   ,gsl_matrix * electrode_pairing, float f0, float fs) {

        /*This function gets the data and calculates the amplitude,
            phase and offset from all channels
         * It then obtains differential values for the electrode
            pairs specified in electrode_pairing and
         * stores resulting amplitude and phase in the phasor
            matrix. Impedance calculation can be performed
         * on the resulting matrix
         *
         * Matrix Shapes:
         * - Data (number of samples, number of channels)
         * - electrode_pairing (number of channels/2, 2)
         * - phasor_matrix (number of cycles, number of channels
            /2*2)*/

        int N_cicles = (Data->size1)/samples_cicle;
        int N_channels = Data->size2;
        int a,b; //codigos dos eletrodos
        gsl_matrix * ddp_matrix; //guarda diferenca entre os canais
        gsl_vector * ddp_a, *ddp_b; //vetores auxiliares para
           calcular as diferencas
        gsl_matrix * temp_demod_matrix, *demod_matrix; //utilizadas
            para guardar valores durante a demodulacao
        gsl_vector * temp_demod_vector; //utilizado para guardar
           valores durante a demodulacao
        gsl_matrix* phasor_matrix; //guarda valores dos fasores de
           corrente e ddp
        gsl_matrix* impedance_matrix; // guarda valores calculados
           de impedancia, em forma retangular
        float temp_A, temp_P;
```

```
23
24          gsl_complex comp_1, comp_2, R_sent;
25
26          fitSine partial_results; // utilizado durante as
               demodulacoes
27          Sys_Results Output; // structure para retornar todas as
               matrizes
28
29          //inicializa
30          ddp_matrix = gsl_matrix_alloc(Data->size1,N_channels/2);
31          ddp_a = gsl_vector_alloc(Data->size1);
32          ddp_b = gsl_vector_alloc(Data->size1);
33
34          temp_demod_matrix = gsl_matrix_alloc(samples_cicle,
               N_channels/2);
35          temp_demod_vector = gsl_vector_alloc(N_channels/2);
36
37          demod_matrix = gsl_matrix_alloc(N_cicles,3*N_channels/2);
38
39          /*comp_1 = gsl_complex_polar(0,0);
40          comp_2 = gsl_complex_polar(0,0);*/
41          R_sent = gsl_complex_polar(30,0);
42          phasor_matrix = gsl_matrix_alloc(N_cicles,N_channels);
43
44          impedance_matrix = gsl_matrix_alloc(N_cicles/2,4); //L L T
               T *******
45
46          //Diferenca dos canais
47          for(int i = 0;i<N_channels/2;i++){
48                  gsl_matrix_get_col(ddp_a,Data,gsl_matrix_get(
                       electrode_pairing,i,0));
49                  gsl_matrix_get_col(ddp_b,Data,gsl_matrix_get(
                       electrode_pairing,i,1));
50
51                  //a-b
52                  gsl_vector_sub(ddp_a,ddp_b);
53
54                  gsl_matrix_set_col(ddp_matrix,i,ddp_a);
55          }
56
57          //Demod
```

```
58          for(int N=0;N<N_cicles;N++){
59                  //Copy portion of matrix
60                  for(int i = 0;i<samples_cicle;i++){
61                          gsl_matrix_get_row(temp_demod_vector,
                               ddp_matrix,N*samples_cicle+i);
62                          gsl_matrix_set_row(temp_demod_matrix,i,
                               temp_demod_vector);
63                  }
64
65                  partial_results = sineRegression_lms(
                       temp_demod_matrix,f0,fs);
66                          for(int k = 0;k<N_channels/3;k++){
67                                  gsl_matrix_set(demod_matrix,N,3*k,
                                       gsl_vector_get(partial_results.
                                       amplitude,k));
68                                  gsl_matrix_set(demod_matrix,N,3*k
                                       +1,gsl_vector_get(
                                       partial_results.phase_rad,k));
69                                  gsl_matrix_set(demod_matrix,N,3*k
                                       +2,gsl_vector_get(
                                       partial_results.offset,k));
70                          }
71                          //Fasor
72                          //VL,VT,IL,IT
73                          for(int k = 0;k<N_channels/2;k++){
74                                  if(k<1){
75                                          gsl_matrix_set(
                                               phasor_matrix,N,2*k,
                                               gsl_vector_get(
                                               partial_results.
                                               amplitude,k));
76                                          gsl_matrix_set(
                                               phasor_matrix,N,2*k+1,
                                               gsl_vector_get(
                                               partial_results.
                                               phase_rad,k));
77                                  }else{
78                                          gsl_matrix_set(
                                               phasor_matrix,N,2*k,
                                               gsl_vector_get(
                                               partial_results.
```

```
                                                amplitude,k)/30);
79                                          gsl_matrix_set(
                                                phasor_matrix,N,2*k+1,
                                                gsl_vector_get(
                                                partial_results.
                                                phase_rad,k));
80                                  }
81                          }
82          }
83
84          //Impedancia
85          int j = 0;
86          for(int i =0;i<N_cicles;i++){
87                  if(i%2==0){
88                          temp_A = gsl_matrix_get(phasor_matrix,j,2)/
                                gsl_matrix_get(phasor_matrix,j,6);
89                          temp_P = gsl_matrix_get(phasor_matrix,j,3)-
                                gsl_matrix_get(phasor_matrix,j,7);//
                                phase VL -phase IL
90
91                          //Resistencia
92                          gsl_matrix_set(impedance_matrix,j,2,temp_A)
                                ;
93                          // Reatancia
94                          gsl_matrix_set(impedance_matrix,j,3,temp_P)
                                ;
95                  }else{
96                          temp_A = gsl_matrix_get(phasor_matrix,j,0)/
                                gsl_matrix_get(phasor_matrix,j,4);
97                          temp_P = gsl_matrix_get(phasor_matrix,j,1)-
                                gsl_matrix_get(phasor_matrix,j,5);//
                                phase VL -phase IL
98
99                          //Resistencia
100                         gsl_matrix_set(impedance_matrix,j,0,temp_A)
                                ;
101                         // Reatancia
102                         gsl_matrix_set(impedance_matrix,j,1,temp_P)
                                ;
103                         j++;
104                 }
```

```
105         }
106
107         // Organiza resultado final
108         Output.data = Data;//
109         Output.demod_data = demod_matrix;
110         Output.phasor_data = phasor_matrix;
111         Output.impedance_data = impedance_matrix;
112
113         return Output;
114 }
```

D.4 Demodulação

```
1  fitSine sineRegression_lms(gsl_matrix* Data, float f0,float
       samp_freq){
2          /*Use GSL for all*/
3
4          float Ts = 1/samp_freq;
5          gsl_vector *time, *omegat; //time vector, angle vector(?)
6          gsl_vector *Sines,*Cosines;
7          gsl_vector *alpha,*beta,*C;
8          gsl_matrix *mat_j, *vec_p, *mat_jI;
9
10         gsl_vector *amplitude,*phase_rad;
11         gsl_vector *phi_alpha,*phi_beta;
12
13         gsl_matrix *gsl_Y;//, *ycos,*ysin;
14
15         float temp_val_1 = 0,temp_val_2 = 0;
16         gsl_vector *temp_vector_1,* temp_vector_2;
17
18         int Nchan,N; //#elements
19         fitSine results;
20
21         //Initialization
22         N = Data->size1; //size of data
23         Nchan = Data->size2;
24
25         time = gsl_vector_alloc(N); //alloc memory for arrays
26         omegat = gsl_vector_alloc(N);
27
28         mat_j = gsl_matrix_alloc(N,3);
```

```
29          mat_jI = gsl_matrix_alloc(3,N);
30
31          vec_p = gsl_matrix_alloc(3,Nchan);
32
33          alpha = gsl_vector_alloc(Nchan);
34          beta = gsl_vector_alloc(Nchan);
35          C =  gsl_vector_alloc(Nchan);
36
37          amplitude = gsl_vector_alloc(Nchan);
38          phase_rad = gsl_vector_alloc(Nchan);
39          phi_alpha = gsl_vector_alloc(Nchan);
40          phi_beta = gsl_vector_alloc(Nchan);
41
42          gsl_Y = gsl_matrix_alloc(N,Nchan);
43
44          //temporary
45          temp_vector_1 = gsl_vector_alloc(Nchan);
46          temp_vector_2 = gsl_vector_alloc(Nchan); //livrar vetores
               dps em todas as funcoes
47
48          //---------------------//
49          for(int i = 0; i<N;i++){gsl_vector_set(time,i,(double)i*Ts)
               ;} //creates time vector
50
51          for(int i = 0; i<N;i++){gsl_vector_set(omegat,i,2*PI*f0*i*
               Ts);}//create omega vector;
52
53          /*Compute mat_J and inverse*/
54          for(int i=0;i<N;i++){
55                  temp_val_1 = gsl_vector_get(omegat,i);
56                  gsl_matrix_set(mat_j,i,0,(float)sin(temp_val_1));
                       //first column  = sines
57                  gsl_matrix_set(mat_j,i,1,(float)cos(temp_val_1));
                       // second columns = cosines
58                  gsl_matrix_set(mat_j,i,2,1.0);
                                        // third column = 1
59          }
60
61          pinv(mat_j,mat_jI);
62
```

```
63          gsl_productMatrix(mat_jI,Data,vec_p);//nessa matriz estao
               contidos alfa, beta e C para cada canal que houver em
               Data
64
65          //separa vec_p
66          gsl_matrix_get_row(alpha,vec_p, 0);
67          gsl_matrix_get_row(beta,vec_p, 1);
68          gsl_matrix_get_row(C,vec_p, 2);
69
70          //amplitude e fase
71          for(int i = 0;i<Nchan;i++){
72                  temp_val_1 = sqrt(pow(gsl_vector_get(alpha,i),2)+
                       pow(gsl_vector_get(beta,i),2));
73                  gsl_vector_set(amplitude,i,temp_val_1);
74
75                  temp_val_2 = atan2(gsl_vector_get(beta,i),
                       gsl_vector_get(alpha,i));
76                  gsl_vector_set(phase_rad,i,temp_val_2);
77
78          }
79          //Phi alfa e phi beta
80          phi_alpha = alpha;
81          gsl_vector_div(phi_alpha,amplitude);
82
83          phi_beta = beta;
84          gsl_vector_div(phi_beta,amplitude);
85
86          for(int i = 0;i<Nchan;i++){
87                  temp_val_1 = acos(gsl_vector_get(phi_alpha,i));
88                  gsl_vector_set(phi_alpha,i,temp_val_1);
89
90                  temp_val_2 = asin(gsl_vector_get(phi_beta,i));
91                  gsl_vector_set(phi_beta,i,temp_val_2);
92          }
93
94          //Y
95          for(int i = 0;i<N;i++){
96                  temp_vector_1 = alpha;
97                  temp_vector_2 = beta;
98
```

```
99                  gsl_vector_scale(temp_vector_1,sin(2*PI*f0*
                       gsl_vector_get(time,i)));
100                 gsl_vector_scale(temp_vector_2,cos(2*PI*f0*
                       gsl_vector_get(time,i)));
101
102                 gsl_vector_add(temp_vector_1,temp_vector_2);
103                 gsl_vector_add(temp_vector_1,C);
104                 gsl_matrix_set_row(gsl_Y,i,temp_vector_1);
105         }
106
107         // Mandar para os resultados
108         results.amplitude = amplitude;
109         results.phase_rad = phase_rad;
110         results.offset = C;
111         results.alpha = alpha;
112         results.beta = beta;
113         results.omegat = omegat;
114         results.y = gsl_Y;
115
116         return results;
117 }
```

D.5 Cálculo de pseudoinversa

```
1  int pinv(gsl_matrix *usr_matrix,gsl_matrix *P_inv){
2
3          /*This function calculates the pseudo inverse matrix based
              on the Moore-Penrose method
4           * using SVD;
5           * The number of rows of the matrix must be >= the number
               of columns, so that there are*/
6          int Nrow = usr_matrix->size1, Ncol = usr_matrix->size2;
7          gsl_matrix *U,*Ut,*V,*Sigma_plus,*Sigma,*VS; // SVD's U, U
              transposed and V matrix, pinv from SVD's Sigma
8          gsl_vector *Sigma_array,*work;
9          float _temp_value;
10
11         /*Initialize necessary Matrices*/
12         //U
13         U = gsl_matrix_alloc(Nrow,Ncol);
14         gsl_matrix_memcpy(U,usr_matrix);
15         //Ut
```

```
16          Ut = gsl_matrix_alloc(Ncol,Nrow);
17          //Sigma_array e Sigma
18          Sigma_array = gsl_vector_alloc(Ncol);
19          Sigma = gsl_matrix_calloc(Ncol,Ncol);
20          //Sigma_plus
21          Sigma_plus = gsl_matrix_calloc(Ncol,Ncol);
22          //V e Vt
23          V = gsl_matrix_alloc(Ncol,Ncol);
24          //Workspace
25          work = gsl_vector_alloc(Ncol);
26          VS = gsl_matrix_alloc(Ncol,Ncol);
27
28          /*thin SVD*/
29          gsl_linalg_SV_decomp(U,V,Sigma_array,work);
30          /* U = m x n
31           * E = n x n
32           * V = n x n*/
33
34          /*make Sigma^+*/
35          for(int i = 0;i<Ncol;i++){
36                  _temp_value = gsl_vector_get(Sigma_array,i);
37                  if(_temp_value<=QUASI_ZERO){
38                          break;
39                  }else{
40                          gsl_matrix_set(Sigma_plus,i,i,1/_temp_value
                               );
41                  }
42                  gsl_matrix_set(Sigma,i,i,_temp_value);
43          }
44
45          /*Transpose U*/
46          gsl_matrix_transpose_memcpy(Ut,U);
47          /* get Pseudo inverse
48           * A_plus = V(n*n) * Sigma_plus(n*n) * Ut(n*m)*/
49
50          gsl_productMatrix(V,Sigma_plus,VS);
51          gsl_productMatrix(VS,Ut,P_inv);
52          //gsl2pointer(P_inv,A);
53          return 1;
54  }
```

D.6 Cálculo de matriz GSL transposta

```
int transposeMatrix(float **usr_matrix, int Nrow,int Ncol, float **
    T_matrix){

        for(int row = 0;row<Nrow;row++){
                for(int col = 0;col<Ncol;col++){
                        *((float *)T_matrix + (col * Nrow) + row) =
                            *((float *)usr_matrix + (row * Ncol) +
                           col);
                }
        }
        return 1;
}
```

D.7 Cálculo de produto de matriz GSL

```
int gsl_productMatrix(gsl_matrix *A, gsl_matrix *B, gsl_matrix * AB
    ){
        double x;
        if(A->size2!=B->size1){
                printf("Number of columns in A is different from
                   number of rows in B!\n");
                return -1;
        }
        for(int i=0;i<A->size1;i++){ //percorre linhas de A e AB
                for(int j=0;j<B->size2;j++){//percorre colunas de
                   AB
                        for(int k=0;k<B->size1;k++){//percorre
                           linhas de B
                                x = gsl_matrix_get(AB,i,j)+
                                   gsl_matrix_get(A,i,k)*
                                   gsl_matrix_get(B,k,j);
                                gsl_matrix_set(AB,i,j,x);
                        }
                }
        }
        return 1;
}
```

D.8 Armazenamento de arquivos

```
int saveFile_gsl(config_t configuration, gsl_matrix * Data, int
    fileType){
        FILE *data_file;
```

```
3         char *fname_data, fname_impedance[50], fname_phasor[50];
4                         config_lookup_string(&configuration,"name"
                             ,&fname_data);
5                 strcpy(fname_impedance,fname_data);
6                 strcpy(fname_phasor,fname_data);
7
8         switch(fileType){
9
10                case 0:
11
12                /*Apenas arquivo com dados base*/
13                strcat(fname_data,"_data.txt");
14                data_file = fopen(fname_data,"w");
15
16                for(int i=0;i<Data->size2;i++){
17                        fprintf(data_file,"Channel_%d; ",i );
18                }
19                fprintf(data_file,"\n");
20                for(int row=0;row<Data->size1;row++){
21                        for(int col=0;col<Data->size2;col++){
22                                fprintf(data_file," %lf ;",
                                    gsl_matrix_get(Data,row,col));
23                        }
24                        fprintf(data_file,"\n");
25                }
26
27                fclose(data_file);
28                break;
29
30                case 1:
31                /*Apenas arquivo com dados de impedancia*/
32                strcat(fname_impedance,"_impedance.txt");
33                data_file = fopen(fname_impedance,"w");
34
35                //Nao generalizado
36                fprintf(data_file,"RL; ");
37                fprintf(data_file,"XL; ");
38                fprintf(data_file,"RT; ");
39                fprintf(data_file,"XT; ");
40
41                fprintf(data_file,"\n");
```

```
42                  for(int row=0;row<Data->size1;row++){
43                          for(int col=0;col<Data->size2;col++){
44                                  fprintf(data_file," %lf ;",
                                       gsl_matrix_get(Data,row,col));
45                          }
46                          fprintf(data_file,"\n");
47                  }
48
49                  fclose(data_file);
50                  break;
51
52                  case 2:
53                  /*Apenas arquivo com dados de fasor*/
54                  strcat(fname_phasor,"_phasors.txt");
55                  data_file = fopen(fname_phasor,"w");
56
57                  fprintf(data_file,"VL_A; ");
58                  fprintf(data_file,"VL_Ph; ");
59                  fprintf(data_file,"VT_A; ");
60                  fprintf(data_file,"VT_Ph; ");
61                  fprintf(data_file,"IL_A; ");
62                  fprintf(data_file,"IL_Ph; ");
63                  fprintf(data_file,"IT_A; ");
64                  fprintf(data_file,"IT_Ph; ");
65
66                  fprintf(data_file,"\n");
67                  for(int row=0;row<Data->size1;row++){
68                          for(int col=0;col<Data->size2;col++){
69                                  fprintf(data_file," %lf ;",
                                       gsl_matrix_get(Data,row,col));
70                          }
71                          fprintf(data_file,"\n");
72                  }
73
74                  fclose(data_file);
75                  break;
76          }
77          return 0;
78  }
```