UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

Sistema de Aquisição de Sinais Eletromiográficos e de Força com Software de Monitoração, Registro e Biofeedback em Ambiente C++ Builder Empregando o Microcontrolador MSP430

Orientador: Adson Ferreira da Rocha

Co-orientador: Wilson Henrique Veneziano

André Garcia Pena

Rodrigo Contini Martinelli Pereira

Projeto Final de Graduação em Engenharia Elétrica (1°/2005)

Brasília, junho de 2005

AGRADECIMENTOS

Rodrigo e André agradecem a todos aqueles que, de alguma forma, contribuíram para a conclusão deste trabalho e da graduação de ambos em Engenharia Elétrica.

Agradecem, primeiramente, ao professor Adson Ferreira da Rocha por ter aceito orientá-los em seu projeto de graduação. Mesmo à distância, fez-se presente, e, sempre que surgiam novos percalços, indicava novos rumos a serem tomados.

Agradecem, ainda, ao doutorando Wilson Henrique Veneziano, co-orientador, que prestou considerável auxílio na etapa de elaboração do relatório final. Contribuiu com artigos, livros e resumos, além de inúmeras sugestões para o texto final, após minuciosa leitura. Sempre atencioso, prontificou-se a auxiliar em tudo aquilo que fosse possível e, de fato, foi imprescindível.

O professor Jake Carvalho do Carmo também tem importante participação no projeto, já que, além de disponibilizar o acesso ao Laboratório de Biomecânica da Faculdade de Educação Física, auxiliou bastante nos momentos de hesitação.

Impossível, ainda, ignorar a contribuição involuntária de milhões de outros brasileiros, excluídos e marginalizados, desprovidos de qualquer expectativa de uma vida digna. Sua exclusão do processo produtivo, conseqüência de uma sociedade injusta e cruel, possibilitou o ingresso dos formandos em uma instituição de qualidade inquestionável e renome internacional. Pelo eterno dever de gratidão a todos aqueles que, com seu sofrimento, financiaram os sonhos de alguns poucos, prometem carregar consigo a lealdade e a honestidade, indissociáveis, tendo a pretensão de justificar o sacrifício de seus irmãos por seu êxito.

Os professores, todos, tiveram destacada importância na empreitada. Entre o melhor corpo docente do país, destacaram-se os professores Antônio José Martins Soares, Lúcio Martins da Silva, Adolfo Bauchspiess, Paulo Henrique Portela de Carvalho, Mauro Moura Severino, Ricardo Zelenovsky e Alcides Leandro da Silva, entre outros.

Finalmente, agradecem às suas respectivas famílias, sem as quais não teriam chegado a lugar algum.

Rodrigo agradece a suas avós, pelos conselhos e por fazerem com que, em seus lares, o neto se sentisse em casa: Maria Martinelli Pereira e Natalia Porcheddu Contini. A seus inúmeros tios e tias, que sempre o incentivaram: Giuseppe Contini, Mauri Martinelli Pereira, Angelo Contini, Luigi Contini, Salvatore Contini, Maurize Martinelli Pereira, Franco Contini, Mauro Martinelli Pereira, Mario Contini Junior, Maurici Martinelli Pereira, Ricardo Andrea

Contini, Deise Martinelli Pereira, Tania Christina Contini, Alessandra Stella Contini, Salvatore Loi e Stefano Pacetti. Aos seus pais, Maurício Martinelli Pereira e Antonia Grasia Contini Martinelli Pereira, que, desde cedo, procuraram conscientizá-lo sobre a relevância dos estudos e propiciaram um ambiente familiar sadio e tranqüilo. À sua irmã, Stefani Contini Martinelli Pereira, a pessoa mais importante em sua vida, pelos risos, abusos, favores, chateações... por ser, enfim, quem ela é! E os seus avôs, Mauri Pereira e Mario Contini, que, embora não tenham podido ver a graduação do neto, estariam orgulhosos e envaidecidos.

Há, ainda, agradecimentos destinados aos amigos Thiago Aguiar Soares, Alex Pires de Azevedo, Jhonathan Morais de Carvalho, Rafael Galvão de Oliveira, Edson Mintsu Hung e Virgínia Fernandes, simplesmente por serem grandes amigos e terem, também, sobrevivido (quase) ilesos ao enlouquecedor curso de Engenharia Elétrica. Além destes, Rodrigo agradece, ainda, a outros amigos, que contribuíram bastante para o êxito alcançado, simplesmente por serem amigos: João Carlos Dourado de Jesus, Thiago de Souza Dias, Túlio de Almeida Costa, Marcos Souza de Oliveira, Victor Ormitto Lopes Gurgel, Thiago de Azevedo Barbosa (diploma não é mais sua exclusividade!), Marco Aurélio do Valle Fiorese, Larissa Hanna do Monte Vieira e Nerone, entre alguns outros.

André agradece a seus pais Carlos Alberto Mundim Pena e Ethel Garcia Pena por terem sempre ajudado a traçar seu caminho, orientando-o em inúmeras etapas de sua vida. À suas irmãs Marina Garcia Pena e Simone Garcia Pena, por fazerem de sua vida um momento repleto de alegrias. À sua namorada, Fernanda Laus de Aquino, por estar sempre ao seu lado, apoiando-o em todos os momentos difíceis. Dedica este trabalho ao seu avô, José Godoy Garcia, que mesmo não estando presente neste momento, sempre foi uma fonte de inspiração pelo seu caráter e jeito simples de ser; ao seu avô Mario Mundim Penna que realiza hoje um de seus sonhos que é ver seu neto se formar; à suas avós Maria do Carmo e Maria Rochael pelo amor e carinho que sempre deram ao seu neto.

Agradecem, ainda, a Deus, sempre, por tudo!

ABSTRACT

SISTEMA DE AQUISIÇÃO DE SINAIS DE FORÇA E ELETROMIOGRÁFICOS COM SOFTWARE DE MONITORAÇÃO, REGISTRO E BIOFEEDBACK EM AMBIENTE C++ BUILDER EMPREGANDO O MICROCONTROLADOR MSP430

This project consisted of the implementation of a system for acquisition, register and biofeedback of force and eletromiographic signals. The central objective was the use of microcontroller MSP430 and the environment of programming C++ Builder, in order to minimize the costs of the system, conferring to it bigger portability and independence. As in previous projects of other researchers, the main application proposal was the verification of the influence of the simultaneous contraction of two antagonistic muscles on the carried through total effort in the intended movement.

The alterations implemented proposals and in this work had shown satisfactory, since, from a less onerous and more flexible technology, they had been reached resulted similar to those of previous works.

The stage of validation of the system, however, was not carried through of the initially foreseen form, due to the time and the high cost of manufacture of the circuit prihnted circuit board. It was opted, then, for the validation in group of benches, inserting themselves signals known in each canal of the analogical-digital converter of the microcontroller, and capturing them by means of implemented software. A spectral analysis of the captured signals proved that the system operates correctly, not losing given nor inserting distortions.

RESUMO

SISTEMA DE AQUISIÇÃO DE SINAIS DE FORÇA E ELETROMIOGRÁFICOS COM SOFTWARE DE MONITORAÇÃO, REGISTRO E BIOFEEDBACK EM AMBIENTE C++ BUILDER EMPREGANDO O MICROCONTROLADOR MSP430

Este projeto consistiu na implementação de um sistema para aquisição, registro e *biofeedback* de sinais de força e eletromiográficos. O objetivo central foi a utilização do microcontrolador MSP430 e do ambiente de programação *C++ Builder*, a fim de minimizar os custos do sistema, conferindo-lhe maior portabilidade e independência. Como em projetos anteriores de outros pesquisadores, a principal aplicação proposta foi a verificação da influência da contração simultânea de dois músculos antagonistas sobre o esforço total realizado no movimento pretendido.

As alterações propostas e implementadas neste trabalho revelaram-se satisfatórias, já que, a partir de uma tecnologia menos onerosa e mais flexível, alcançaram-se resultados semelhantes àqueles de trabalhos prévios.

A etapa de validação do sistema, contudo, não foi realizada da forma inicialmente prevista, devido ao tempo exíguo e ao alto custo de fabricação da placa de circuito impresso profissional. Optou-se, então, pela validação em bancada, inserindo-se sinais conhecidos em cada canal do conversor analógico-digital do microcontrolador, e capturando-os por meio do software implementado. A análise espectral dos sinais capturados comprovou que o sistema opera corretamente, não perdendo dados nem inserindo distorções.

SUMÁRIO

LISTA DE FIGURAS

Capítulo 2

Capítulo 3

Capítulo 4

Capítulo 5

LISTA DE TABELAS

Capítulo 3

Capítulo 4

Capítulo 5

Capítulo 1: INTRODUÇÃO

1.1. OBJETIVO

O objetivo deste trabalho é a implementação de um sistema capaz de adquirir, registrar, processar e monitorar em tempo real sinais de força e eletromiográficos. Deseja-se, com isso, estudar músculos como o abdutor curto do polegar, o bíceps e o quadríceps, além de seus respectivos antagonistas.

Para tanto, o software deve exibir de modo simples e direto os sinais captados na contração desses músculos. A partir da meta de força, poder-se-ia observar o nível médio quadrático (RMS) dos eletromiogramas dos músculos agonista e antagonista. Além disso, o programa deve permitir o cadastro de sujeitos e o armazenamento dos arquivos referentes aos seus experimentos de forma prática e organizada, em pastas individuais. Assim, seria possível observar e analisar, posteriormente, os resultados obtidos, levantando-se históricos individuais.

Outro aspecto considerado é o custo de implementação do sistema. Por isso, optou-se pela utilização do microcontrolador de baixo consumo MSP430 (Texas Instruments, EUA) e da plataforma *C++ Builder* (Borland, EUA). Este microcontrolador, um circuito integrado de baixo preço, supriria as necessidades de aquisição dos sinais, enquanto a plataforma escolhida permitiria a criação de um arquivo de instalação. Isso conferiria maior independência ao programa, já que o computador a recebê-lo não necessitaria da instalação do *C++ Builder*, um software comercial de elevado custo.

1.2. MOTIVAÇÃO

O Departamento de Engenharia Elétrica da Universidade de Brasília possui um extenso histórico de pesquisas e experimentos na área biomédica. A parceria com a Faculdade de Educação Física propiciou um novo patamar para estes estudos, já que despertou em vários alunos o interesse nesse campo multidisciplinar. Desta forma, vários projetos de graduação destinaram-se a estudos voltados a esta área, atingindo, sempre, resultados bastante satisfatórios.

Dos trabalhos supracitados, vários se destinaram ao estudo do eletromiograma – sua captação, transmissão, processamento e observação. Verificou-se, então, a partir da análise de eletromiogramas, que os resultados obtidos para um mesmo nível de força diferem bastante

para diferentes indivíduos. Foram, ainda, desenvolvidas várias ferramentas para a aquisição e processamento de sinais eletromiográficos.

Uma dessas ferramentas, em particular, desenvolvida por Rodrigo Tapia Passos de Oliveira e Vitor Barata Ribeiro Blanco Barroso, sob orientação dos professores Adson Ferreira da Rocha e Jake Carvalho do Carmo, influenciou os rumos deste trabalho. O sistema desenvolvido pela dupla processava o eletromiograma obtido e retornava os resultados de forma simples e direta, sendo bastante útil para o registro e o *biofeedback* de sinais de força e eletromiográficos. Contudo, estava atrelado à plataforma *Labview* (National Instruments, EUA), o que exigia a presença deste software comercial de elevado custo no computador utilizado para as observações. Além disso, o sistema não utilizava o microcontrolador MSP430, mas uma placa de aquisição de dados da National Instruments.

O professor Adson sugeriu, então, a implementação de um sistema semelhante, mas que utilizasse o MSP430 para a aquisição e um programa desenvolvido no ambiente *C++ Builder*. Isso simplificaria o sistema, reduzindo seus custos de implementação e conferindo-lhe maior flexibilidade e independência.

1.3. ORGANIZAÇÃO DO TRABALHO

Este trabalho está dividido em sete capítulos.

- Este capítulo 1 traz uma sucinta apresentação do projeto a partir de seus objetivos e suas motivações.
- O capítulo 2, por sua vez, apresenta uma revisão bibliográfica acerca do eletromiograma, com ênfase no eletromiograma de superfície – evolução histórica, formação, aplicações e empecilhos em sua utilização.
- O capítulo 3 apresenta o software desenvolvido e poderá servir como seu manual de uso.
- Já o capítulo 4 traz o hardware utilizado para a aquisição dos sinais de força e eletromiográficos.
- O capítulo 5 apresenta os dados obtidos na etapa de validação do sistema implementado.
- O capítulo 6 traz, além das principais dificuldades encontradas no decorrer do trabalho, propostas para trabalhos correlatos futuros.

- Finalmente, o capítulo 7 conclui o trabalho a partir de uma visão geral de tudo o que foi realizado, apontando os objetivos atingidos e os não atingidos.

Capítulo 2: REVISÃO BIBLIOGRÁFICA

2.1. O ELETROMIOGRAMA

2.1.1. Histórico e evolução

O movimento intrínseco é o sinal primordial da vida animal. O homem sempre apresentou curiosidade com relação aos órgãos da locomoção do seu próprio corpo e no dos outros seres [4].

Os músculos podem contrair e produzir força. Nos organismos vivos, os movimentos são realizados por ativação muscular. Através do movimento coordenado das suas partes, os organismos podem mudar a sua posição no espaço e aplicar forças mecânicas no ambiente. Além da locomoção, os músculos são requisitados para processos de transporte dentro do corpo, como a condução de fluidos nos sistema cardiovascular e gastrintestinal ou no transporte de gases no sistema respiratório [11].

As primeiras experiências científicas das quais se tem conhecimento estão relacionadas com os músculos e suas funções. Leonardo da Vinci dedicou grande parte do seu pensamento à análise dos músculos e suas funções. O mesmo aconteceu com Andrea Versalius, considerado o pai da anatomia moderna, mas estes se preocuparam mais com a geografia dos músculos mortos e não com a sua dinâmica.

O primeiro homem a devolver a vida aos músculos foi Galvani, que, no final do século XVIII, publicou suas experiências com preparados neuromusculares e eletricidade animal. Por mais de dois séculos, os biólogos trabalharam com as revelações de Galvani de que os músculos esqueléticos se contraem ao serem estimulados com eletricidade e que, ao se contraírem, por qualquer motivo, geram uma corrente ou tensão perceptível. As descobertas de Galvani marcaram os inícios da neurofisiologia e do estudo da dinâmica da contração muscular. Porém o mundo teve que aguardar até que o francês Duchenne, em meados do século passado XIX, aplicasse a eletricidade a músculos esqueléticos intactos. O seu trabalho *Physiologie des Mouvements* apresentou a descrição dos movimentos que produzem os músculos estimulados através da pele por correntes elétricas. Contudo, o descobrimento de Galvani permaneceu como uma curiosidade científica até o século XX, quando se desenvolveram melhores métodos para captar e registrar minúsculas cargas elétricas. O mérito principal do desenvolvimento da nova técnica de captar os potenciais elétricos gerados no músculo, a eletromiografia (EMG), é atribuído aos fisiologistas ingleses Adrian e Bronk, ao

americano D. Denny-Brown e a vários escandinavos. Deve-se admitir que as primeiras técnicas não se prestavam para estudos detalhados. Durante décadas, aplicou-se a eletromiografia por razões diagnósticas e clínicas e não objetivando a cinesiologia básica [4].

Em 1960 nasce o *biofeedback*, com o cientista Basmajian, que fazia treinamento de sistema neuro-muscular (unidades motoras isoladas) utilizando *biofeedback* de sinais eletromiográficos, para com isso poder efetivar as técnicas de relaxamento geral e de tratamento de dores.

2.1.2. **Princípio de formação do eletromiograma**

O corpo humano possui, em sua constituição, milhares de músculos. Estes, divididos em vários grupos, possuem tamanho e constituição variados, permitindo que realizem tarefas variadas nos sistemas locomotor, reprodutor, urinário, entre outros. A Figura 2.1 exibe os maiores músculos superficiais do corpo humano, vários deles objeto de profundos estudos eletromiográficos [20].

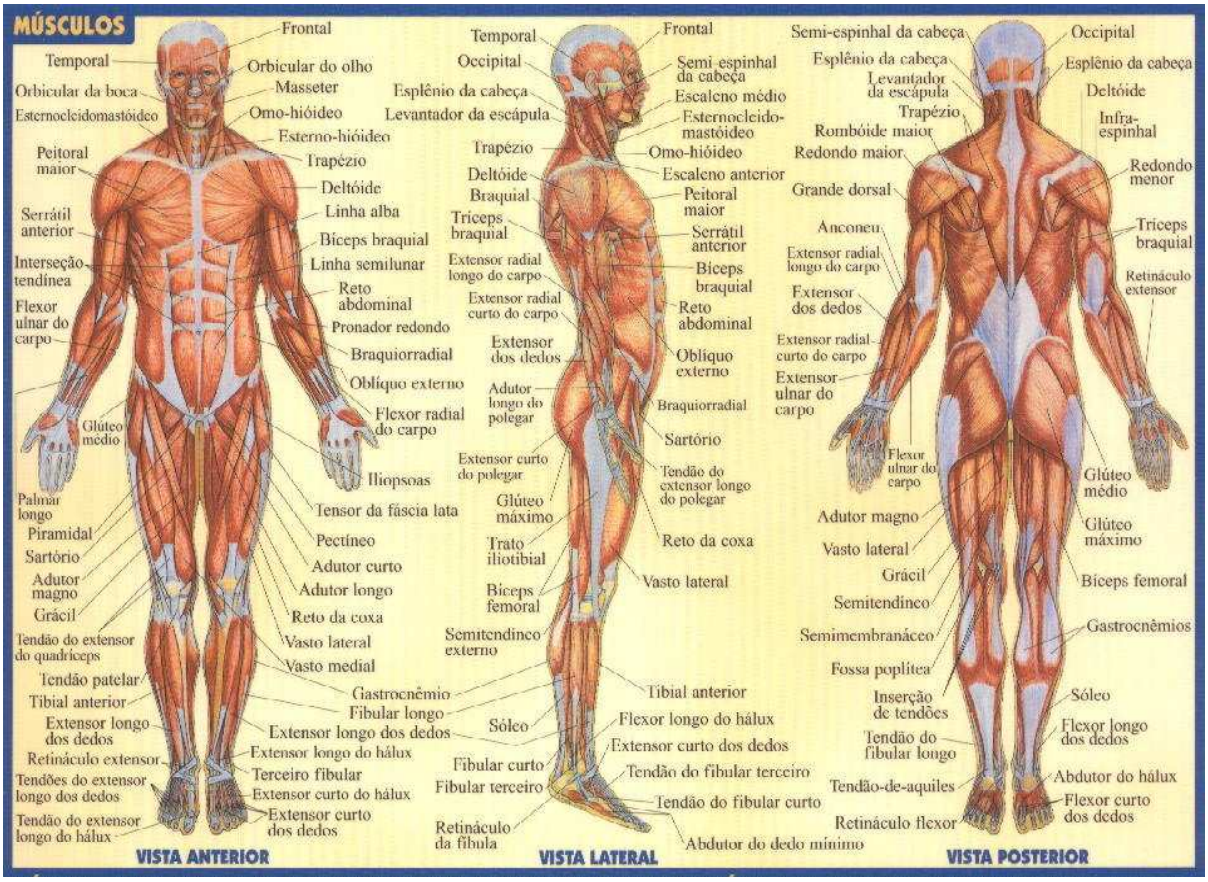


Figura 2.1 – Músculos superficiais do corpo humano [9].

A contração muscular e a conseqüente produção de força são provocadas pela mudança relativa de posição de várias moléculas ou filamentos no interior do arranjo muscular. Tal mudança consiste do deslizamento de filamentos, o que é provocado por um fenômeno elétrico conhecido como potencial de ação. O potencial de ação resulta da mudança no potencial de membrana que existe entre o interior e o exterior da célula muscular. Denomina-se eletromiografia exatamente o registro dos padrões de potenciais de ação. De uma forma mais técnica, diz-se que, ao simplesmente registrar-se o sinal muscular, efetiva-se um eletromiograma. A eletromiografia é um registro especial, no qual o fenômeno elétrico está relacionado à contração muscular.

Os sinais elétricos que chegam ao músculo a partir dos nervos geram o fenômeno da contração muscular. Este, exibido na Figura 2.2, ocorre da seguinte maneira: no estágio A, o potencial de ação que se propaga pelo filamento nervoso atinge a junção neuro-muscular. Já no estágio B, quando o pico de onda neural atinge a junção, libera os neuro-transmissores responsáveis pela geração do sinal no músculo. Na fase C, os neuro-transmissores geram um potencial no músculo. Finalmente, em D, o músculo se contrai [19].

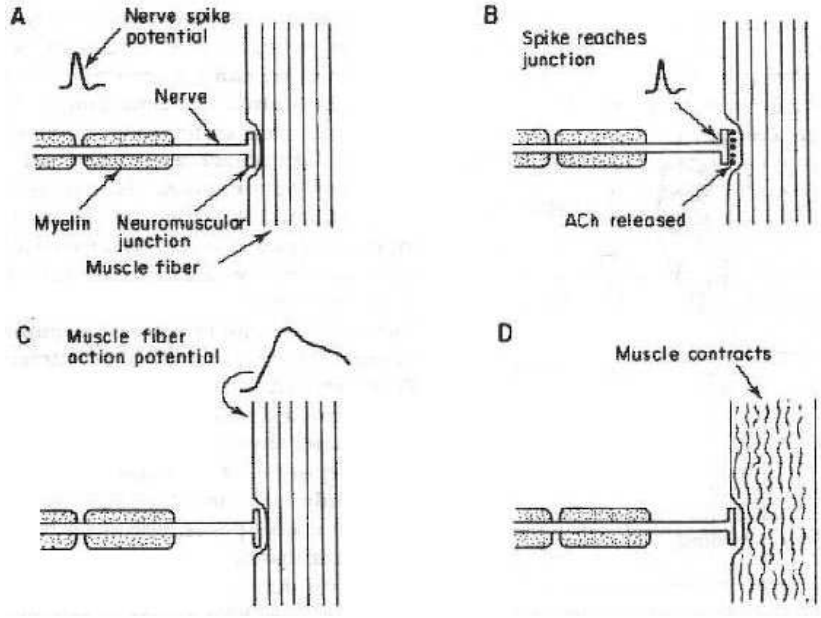


Figura 2.2 – Estágios da contração muscular.

A Figura 2.3 exibe o esquemático de um músculo esquelético do fêmur, detalhando desde o músculo completo e seu posicionamento na perna até os sarcômeros, a base para o desenvolvimento da contração muscular.

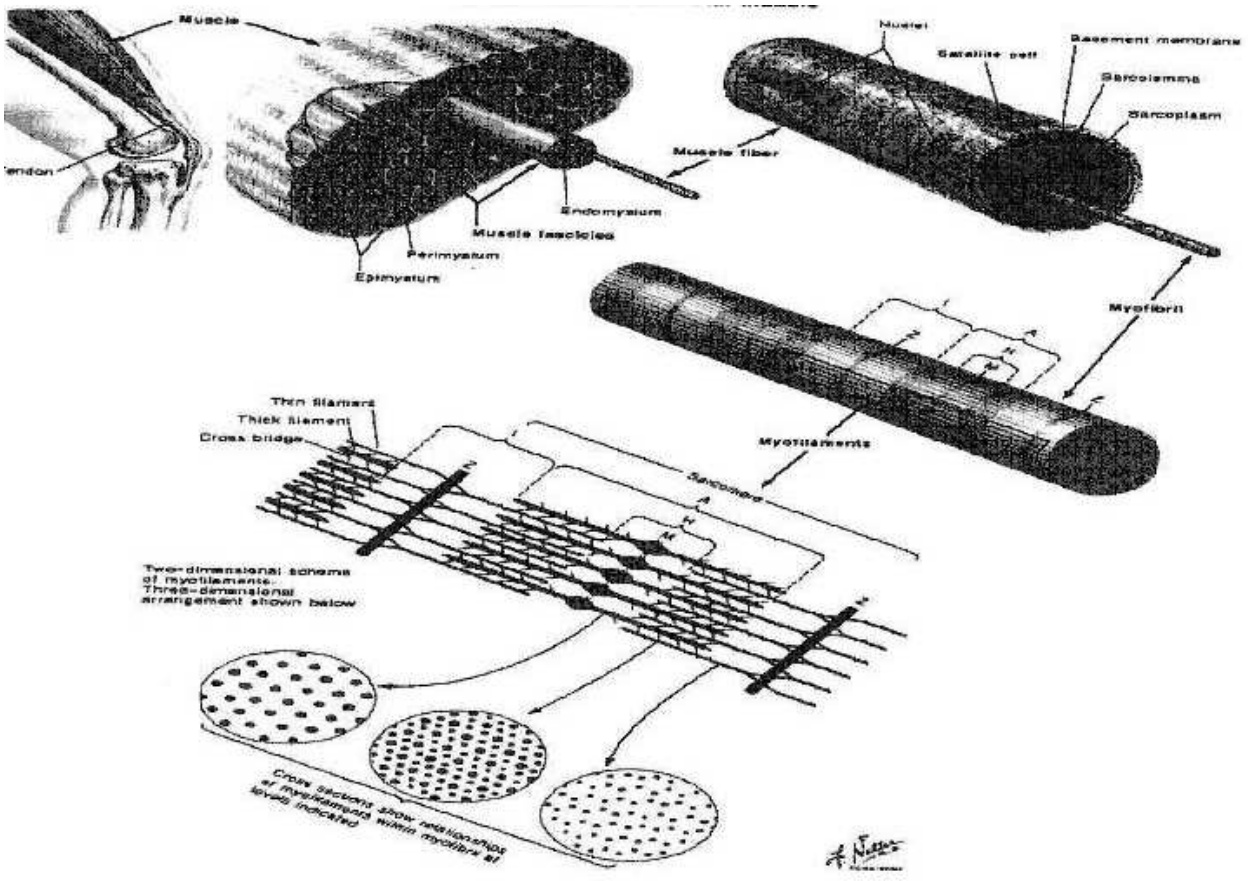


Figura 2.3 – A composição do músculo, das células musculares, das miofibras e dos miofilamentos.

Os sinais elétricos que chegam ao músculo a partir dos nervos geram o potencial de ação que viaja através de invaginações da membrana plasmática muscular, também chamada de túbulos t. A organização gerada a partir das inúmeras invaginações em cada célula permite que o potencial elétrico se desloque até a profundeza do músculo quase que instantaneamente. Os potenciais então liberam íons de cálcio, que são os responsáveis pela facilitação da contração muscular.

2.1.3. **Tipos de eletromiograma**

A eletromiografia é um dos métodos clássicos utilizados para registrar a atividade de um determinado músculo. A eletromiografia pode ser dividida em dois tipos, seguindo a classificação de Correia et. al. [12]:

Eletromiografia de profundidade: os eletrodos são colocados no interior do músculo, em contato direto com as fibras musculares. Este tipo de registro não é representativo quando o objetivo é estudar a atividade global de um músculo, é pouco utilizado por ser um método invasivo.

Eletromiografia de superfície: os eletrodos são colocados sob a pele, captando a soma da atividade elétrica de todas as fibras musculares ativas. Caracteriza-se por ser um método não-invasivo e de fácil execução, este método é largamente utilizado em áreas como o estudo cinesiológico e neurofisiologico dos músculos superficiais.

O método invasivo, que requer a utilização de agulhas e/ou microeletrodos, é o método rotineiramente empregado na prática clínica, mas causa dor e desconforto ao paciente. O outro método é conhecido como eletromiografia de superfície, pelo fato de utilizar eletrodos metálicos na superfície da pele e gel salino condutor de eletricidade. Ao contrário do outro método, não é de uso restrito por médicos, sendo amplamente empregado por fisioterapeutas e, mais recentemente, por profissionais da área desportiva [20].

A este projeto, interessa o método da eletromiografia de superfície, devido ao seu caráter não-invasivo e suas conseqüentes simplicidade e segurança, conforme relatado no parágrafo anterior.

Uma considerável limitação desse tipo de eletromiografia é a ocorrência do "cross-talk", a interferência da energia de um músculo sobre o campo de gravação de outro músculo. Esse tipo de ruído é tratado em seção posterior.

Outras desvantagens seriam a limitação do número de canais de aquisição e a complexidade do sistema neuromuscular. Devido a isso, apenas poucos pontos de captação de sinal podem ser monitorados. Diretamente associada a isso está a insuficiência de publicações que descrevam o posicionamento dos eletrodos na aquisição dos sinais eletromiográficos. De fato, não há, ainda, padronização nesse sentido [20].

2.1.4. Aplicações do eletromiograma e pesquisas recentes desenvolvidas

Atualmente as principais aplicações do sinal de EMG são:

- determinação do tempo de ativação do músculo, ou seja, o tempo entre o começo e o fim da excitação muscular;

- estimação da força produzida pelo músculo;

- obtenção de um índice da fadiga do músculo a partir da análise do espectro de freqüência do sinal eletromiográfico;

- aperfeiçoamento de atletas;

- auxílio na atividade de análise postural e ergonômica;

- num futuro próximo, avaliação e tratamento de doenças motoras e neurológicas.

2.2. CAPTAÇÃO DE SINAIS BIOELÉTRICOS

2.2.1. Tipos de eletrodos

Para captação do sinal emanado do músculo utilizam-se eletrodos. Estes devem ser selecionados e colocados de maneira apropriada e em um ponto correto do músculo.

No que diz respeito ao formato, há várias opções de eletrodos: circular, oval, quadrado, retangular ou em forma de pinos. O tamanho do eletrodo é o tamanho da área condutora, podendo variar de 1 mm^2 a alguns cm^2. Salienta-se a importância de que seja grande o suficiente para captar um número razoável de unidades motoras, porém pequeno o bastante para evitar interferência do tipo "cross-talk" de outros músculos.

Os eletrodos podem ser de diversos tipos de material: Ag/AgCl, AgCl, Ag, Au e outros. O material deve propiciar um bom contato entre a pele e o eletrodo e um comportamento estável no tempo (devido às reações químicas na interface). Comumente, emprega-se na superfície do eletrodo uma camada de gel salino condutor para reduzir a impedância entre a pele e o eletrodo [20].

O projeto do eletrodo é o mais crítico dentre os aparatos que serão usados para obter o sinal. A fidelidade do sinal eletromiográfico detectado pelo eletrodo influencia todo tratamento do sinal subseqüente. É muito difícil (quase impossível) melhorar a fidelidade e a razão sinal-ruído do sinal após sua captação. Portanto, o tipo de eletrodo e as características do amplificador desempenham um papel crucial na obtenção de um sinal livre de ruído ("*noise-free*") [20].

Utilizam-se, então, dois tipos principais de eletrodos: superfície (eletrodos ativos e passivos) e fio. Eletrodos de superfície ativos são construídos em amplificadores para melhorar a impedância (não é necessário o uso de gel, além de aumentar a razão sinal-ruído). Eletrodos de superfície passivos detectam o sinal de EMG sem pré-amplificá-lo, sendo importante reduzir a resistência da pele (requer o uso do gel condutor e preparação da pele). Esses eletrodos possuem uma razão sinal-ruído menor. As vantagens do uso de eletrodos superficiais são: não serem dolorosos em sua aplicação e uso, serem fáceis de usar e serem ótimos em medidas contendo movimentos. A desvantagem é que possuem uma grande área de coleta podendo acarretar "*cross talk*".

A configuração dos eletrodos de superfície, utilizados neste projeto, pode ser:

Monopolar: onde um eletrodo é colocado sobre o feixe muscular de interesse e o outro eletrodo (chamado de referência) é colocado num ponto não afetado pela atividade do feixe

muscular de interesse, mede-se então a diferença de potencial entre estes dois pontos. Este tipo de configuração capta maior quantidade de ruído do que outras configurações.

Bipolar: consiste em colocar dois eletrodos sobre a região que se deseja estudar e o terceiro eletrodo chamado terra é colocado num local não afetado pela atividade da região de interesse. Mede-se agora a diferença de potencial elétrico entre os dois eletrodos que estão sobre a região de interesse, tomando-se como referência o eletrodo terra. Desta forma é possível a utilização de amplificadores diferenciais de alto ganho, o que em última análise melhoram significativamente a relação sinal-ruído, uma vez que os ruídos presentes nos cabos que levam o sinal dos eletrodos ao condicionador são, então, subtraídos pelo amplificador diferencial

A afixação dos eletrodos deve ser precisa e, para isso, são utilizados atlas de eletromiografia. A Figura 2.4 exibe um exemplo de marcações para a localização de eletrodos de superfície.

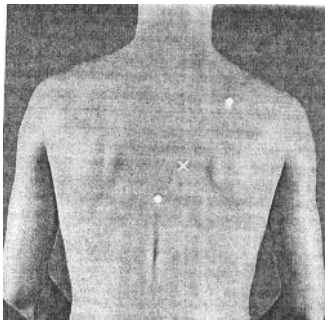


Figura 2.4 – Exemplo de marcações para a localização de eletrodos de superfície [16].

Além disso, com o intuito de melhorar o contato entre a pele e o eletrodo, aquela deve ser preparada. Essa simples medida, que consiste na raspagem dos pêlos da região, e a limpeza da superfície com álcool ou sabão neutro, diminui o ruído e o risco de desbalanceamento de impedância entre os eletrodos.

Ainda sobre o posicionamento do eletrodo no músculo observado, aquele deve ser colocado entre a zona de inervação e a de inserção do tendão junto ao músculo, ao longo da linha média longitudinal, conforme ilustra a Figura 2.5.

A Figura 2.6, por sua vez, traz um eletromiograma detectado do músculo anterior da tíbia durante uma contração isométrica a 50% da força máxima voluntária e seu espectro de freqüência.

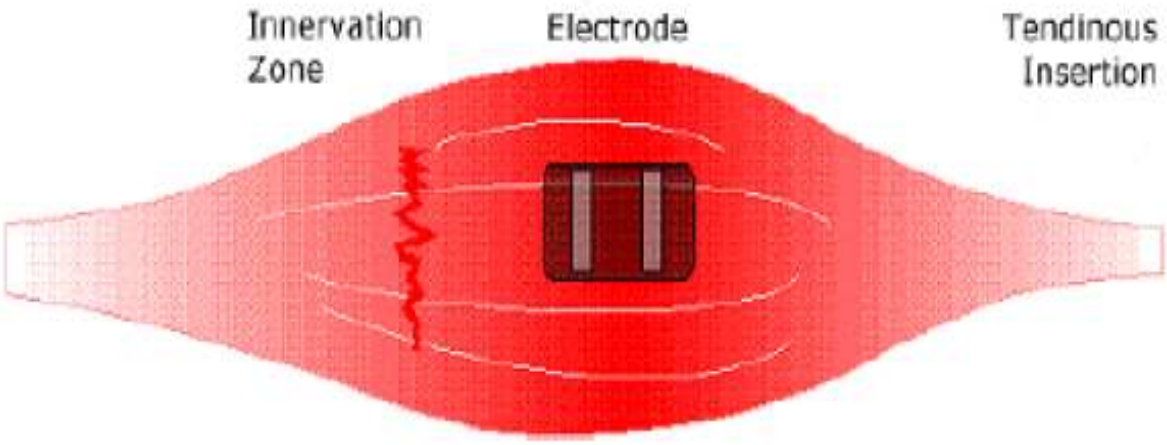


Figura 2.5 – Demonstração da melhor posição para colocação do eletrodo comercial da Delsys [15].

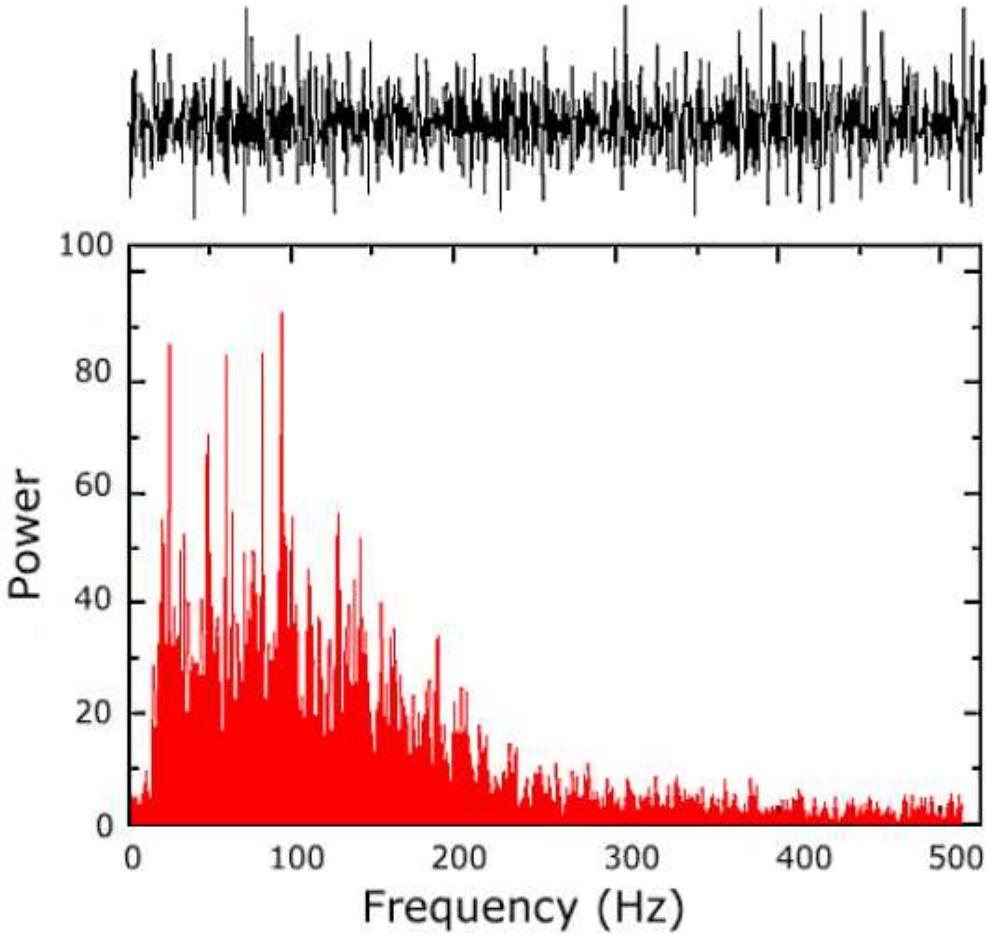


Figura 2.6 – Espectro de freqüência de um sinal de EMG detectado do músculo anterior da tíbia durante uma contração isométrica a 50% da força máxima voluntária [15].

2.2.2. Principais interferências na aquisição

Um sinal elétrico não é distorcido apenas pelo canal, podendo ser contaminado ao longo do caminho por sinais indesejados, e de comportamento aleatório e imprevisível, reunidos no amplo termo ruído. Ou seja, do ponto de vista funcional, ruídos e artefatos são definidos como qualquer interferência contida no sinal eletromiográfico. Este pode ocorrer por causas externas ou internas. O ruído externo inclui a interferência de sinais transmitidos em canais adjacentes, ruído gerado por interruptores defeituosos de equipamentos elétricos, luzes fluorescentes ou ruído natural proveniente de relâmpagos e tempestades elétricas, por exemplo. Com os devidos cuidados, o ruído externo pode ser minimizado. Já o ruído interno resulta basicamente da movimentação térmica de elétrons em condutores. Cuidados adequados podem reduzir os efeitos do ruído interno, mas nunca eliminá-los. O ruído é um dos fatores básicos que limitam a taxa de transmissão [5].

Entre os inúmeros tipos de ruído, um dos mais significativos é aquele inerente aos componentes eletrônicos dos equipamentos de detecção e gravação de sinais. De fato, todos os equipamentos eletrônicos geram ruído elétrico. Esse ruído tem componentes em uma larga faixa de freqüências – entre zero e alguns milhares de hertz. Esse ruído, embora não possa ser eliminado, pode ser reduzido utilizando-se componentes eletrônicos de alta qualidade, circuitos inteligentes e técnicas de construção adequadas [15].

Outro significativo ruído é aquele gerado pelo ambiente de trabalho. Ele origina-se das diferentes fontes de radiação eletromagnéticas, como transmissores de rádio e televisão, cabos elétricos, bulbos de luz, lâmpadas fluorescentes, motores, etc. De fato, qualquer equipamento eletromagnético gera tais ondas e pode contribuir para o ruído. O sinal de ruído dominante do ambiente provém do sinal de 60 Hz da rede de energia. O ruído ambiente, em geral, pode ter amplitudes que variam entre uma e três vezes a ordem de magnitude do sinal eletromiográfico. Por ter amplitudes tão significativas, este é um ruído também bastante prejudicial.

Como supracitado, a interferência da rede de instalação elétrica, na freqüência 60 Hz, é uma considerável fonte de ruído. Eletrodos mal conectados permitem que seus efeitos sejam maximizados, comprometendo consideravelmente a qualidade dos sinais observados. O comprometimento da qualidade desses sinais pode acarretar sérias perdas de informação para determinados estudos. A utilização de cabos elétricos mais curtos e blindados, além de certos cuidados na realização dos testes, como guardar certa distância de equipamentos elétricos, como lâmpadas e computadores, pode reduzir consideravelmente a interferência desse tipo de

ruído. Alguns eletromiógrafos possuem, ainda, filtros eletrônicos projetados para combater esse ruído.

Outro ruído presente na aquisição de sinais eletromiográficos é o chamado ruído de movimento, que aparece como deslocamentos de corrente contínua ou deflexões abundantes nos potenciais do eletromiograma bruto (sem processamento). Isto ocorre quando o eletrodo desliza pela pele, gerando um potencial elétrico dele mesmo. A Figura 2.7 apresenta este fenômeno. No sinal processado, o ruído de movimento aparece como uma deflexão para cima, dificultando a sua identificação. Na Figura 2.8 podem ser observadas alterações no sinal eletromiográfico para diferentes posicionamentos do eletrodo. Este ruído pode ser reduzido com o emprego de eletrodos flutuantes (flutuam sobre uma camada de gel) no lugar de eletrodos de contato direto com a pele. Pode-se, ainda, afixar os eletrodos à pele com fita adesiva, impedindo o deslocamento relativo entre eles. A maior parte da energia dos sinais dessas fontes de ruído encontra-se na faixa de freqüências entre 0 e 20 Hz [20].

Outra fonte de ruído presente na aquisição de sinais eletromiográficos é o sinal de eletrocardiografia, captado com facilidade nas regiões lombar e do torso. Este é um sinal mais intenso que o muscular. Pode-se combater esse ruído com a fixação dos eletrodos mais próximos um ao outro. A Figura 2.7 traz sinais eletromiográficos contaminados por sinais eletrocardiográficos.

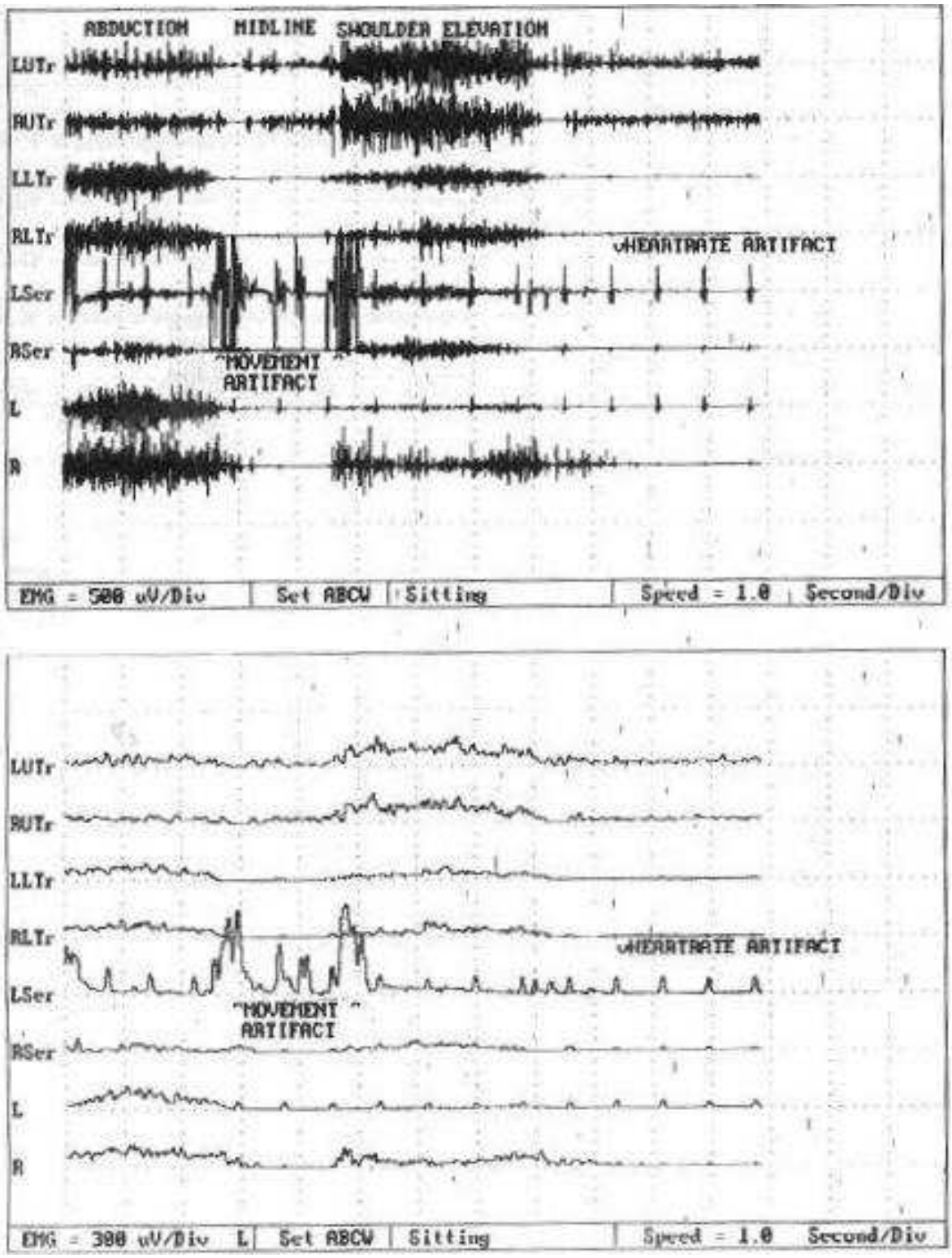


Figura 2.7 – Eletromiograma bruto (acima) e após processamento digital (abaixo). Percebem-se a influência de sinal eletrocardiográfico e de ruído de movimento [10].

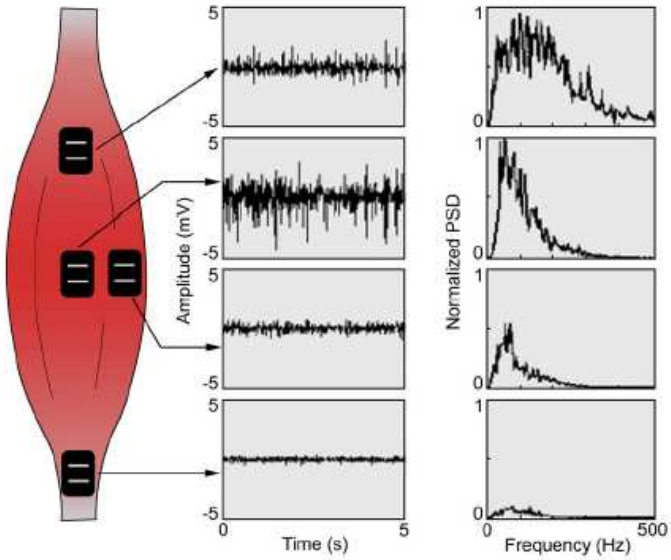


Figura 2.8 – Alterações do sinal para diversos posicionamentos do eletrodo [15].

O sinal dos músculos do sistema respiratório, especialmente quando eles são muito exigidos (por exemplo, na execução de exercícios aeróbicos), também é nocivo à aquisição e observação de sinais eletromiográficos. O treinamento do indivíduo, visando a educar seu processo de respiração e, conseqüentemente, permitindo que passe a apresentar um melhor padrão respiratório, pode minimizar os efeitos dessa fonte de ruído [20].

Não tão freqüente, mas prejudicial em certas condições, é o ruído causado por campos eletromagnéticos de radiofreqüência. Nesse caso, os cabos elétricos dos eletrodos atuam como antenas e captam sinais de alguma fonte de radiofreqüência. Pode-se minimizar os efeitos dessa fonte de ruído deslocando-se o eletromiógrafo para outra sala ou prédio.

O “cross-talk”, anteriormente mencionado, é outra importante fonte de ruído nesse tipo de experimento Consiste na interferência da energia de um músculo sobre o campo de gravação de outro músculo. É quando o sinal elétrico de um músculo adjacente atinge o eletrodo do músculo sob estudo. Apesar de ser muito freqüente nas contrações dinâmicas, é altamente indesejado nos protocolos em que há relaxamento. Com a ocorrência do “cross-talk”, o sinal do músculo observado estaria contaminado por um sinal indesejado proveniente de outro músculo, dificultando seu monitoramento. Um exemplo de “cross-talk” é ilustrado na Figura 2.9. Esta exibe uma situação na qual o sinal próximo da sobrancelha (à esquerda da figura) é contaminado pelo sinal do músculo masseter quando o indivíduo aperta os dois maxilares (à direita da figura). Pode-se diminuir esse ruído a partir da cuidadosa disposição dos eletrodos, de modo a evitar a captação de sinais interferentes provenientes de outros músculos.

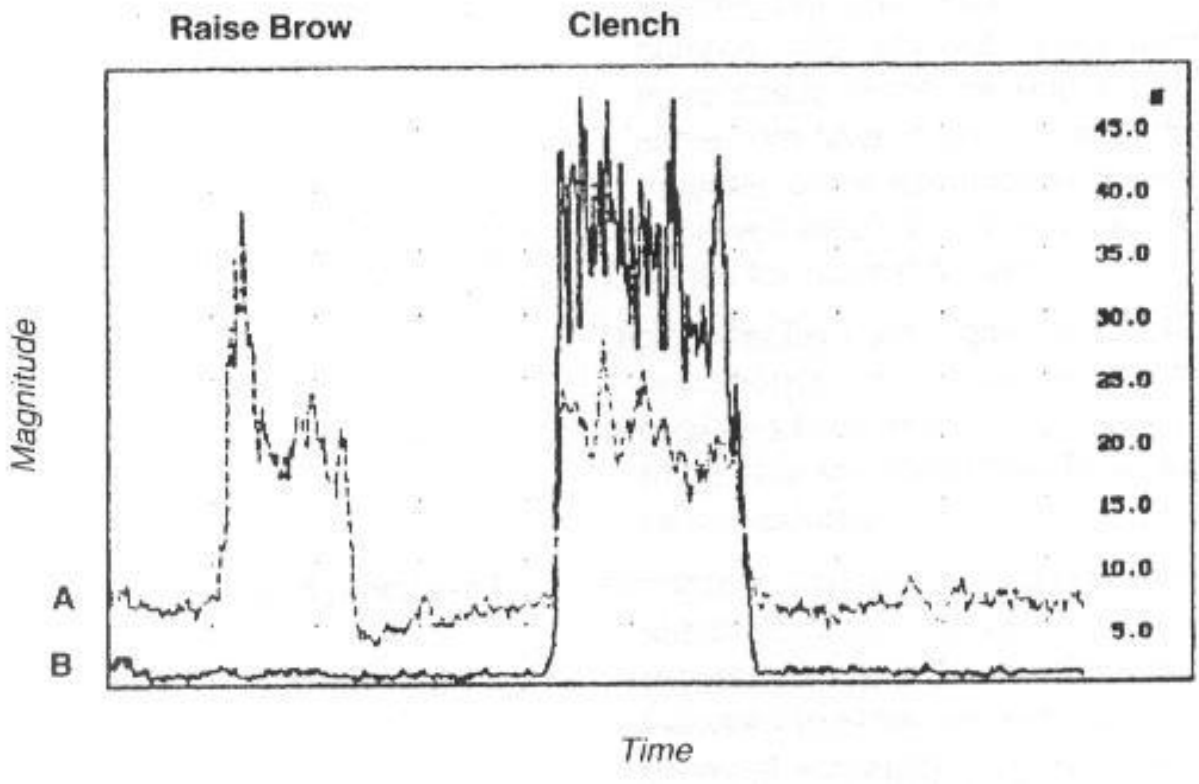


Figura 2.9 – Músculo masseter (A) ocasiona "cross-talk" em captação na sobrancelha (B) durante oclusão dos maxilares [10].

Capítulo 3: DESCRIÇÃO E APRESENTAÇÃO DO SOFTWARE

3.1. CONSIDERAÇÕES INICIAIS

Pesquisas e projetos recentemente desenvolvidos por alunos do Departamento de Engenharia Elétrica da Universidade de Brasília chegaram à implementação de alguns programas computacionais para o processamento de eletromiogramas. Dentre eles, um programa desenvolvido por Rodrigo Tapia Passos de Oliveira e Vitor Barata Ribeiro Blanco Barroso [19], orientado pelos professores Adson Ferreira da Rocha e Jake Carvalho do Carmo, processa o sinal eletromiográfico obtido e retorna os resultados de forma simples e direta, consistindo em uma ferramenta bastante útil para o registro e o *biofeedback* de sinais de força e eletromiográficos.

Contudo, este software foi desenvolvido no ambiente de programação *LabView*, o que exige a instalação deste último no computador. Por se tratar de um software de alto custo financeiro, esta é uma considerável limitação que deve ser superada. Por razões técnicas e financeiras, é interessante a implementação de um software semelhante no ambiente *C++ Builder*, o que permite a criação de um arquivo de instalação. Isso confere maior independência ao programa, uma vez que não se faz necessária a instalação do *LabView* ou mesmo do *C++ Builder* no computador utilizado.

Outra vantagem da utilização do ambiente *C++ Builder* em detrimento ao *Labview* é que aquele é menos denso (requer menos capacidade de processamento do hardware), podendo ser executado com mais facilidade pelo computador. Isso permite que se atinjam taxas de aquisição de dados superiores àquelas possíveis com o *Labview.*

O desenvolvimento do programa no ambiente *C++ Builder* passou a ser, portanto, um dos mais significativos objetivos deste projeto, comparado à utilização do microcontrolador MSP para a aquisição dos sinais.

Optou-se, no desenvolvimento do software, por implementar as funções mais utilizadas do software em *Labview*, aperfeiçoando-se certos pontos e adequando seus parâmetros às necessidades do Laboratório de Biomecânica da Faculdade de Educação Física. Neste ponto, foi importantíssimo o auxílio do professor Jake Carvalho do Carmo, que exibiu o programa existente e orientou as modificações necessárias, de modo a otimizar a utilidade do novo software. Optou-se por priorizar a aquisição com meta de força, já que a função de aquisição com meta de amplitude do eletromiograma não era utilizada.

Este capítulo destina-se a explicar o funcionamento do programa desenvolvido, além de exibir suas configurações, telas e opções, podendo servir como um manual de utilização.

3.2. INSTALAÇÃO

A instalação do software é bastante simples. Executando-se o arquivo setup.exe, bastará indicar a pasta na qual serão instalados os arquivos do programa e confirmar. A Figura 3.1 exibe os arquivos de instalação do software.

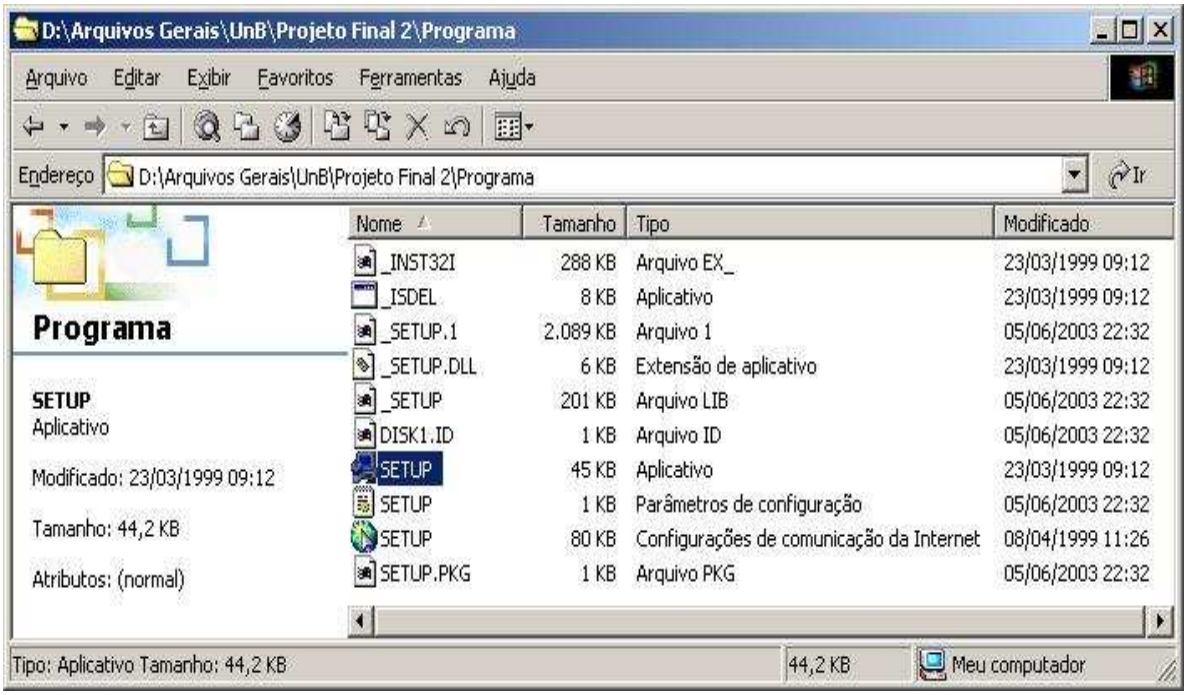


Figura 3.1 – Arquivos de instalação do software.

3.3. JANELA PRINCIPAL

Ao se iniciar o programa, será aberta sua janela principal, exibida na Figura 3.2. Além do menu principal, na parte superior da janela, há atalhos para diversas outras funções, como a calibração da célula de força e a obtenção da força máxima.

Além disso, na janela principal são exibidos o gráfico de força no decorrer do tempo e os gráficos com os valores RMS dos eletromiogramas dos músculos agonista e antagonista.

Há, ainda, a opção de controle da força desejada, na qual são configuradas a meta de força e a sua tolerância percentual. Todas essas funções serão explicadas posteriormente.

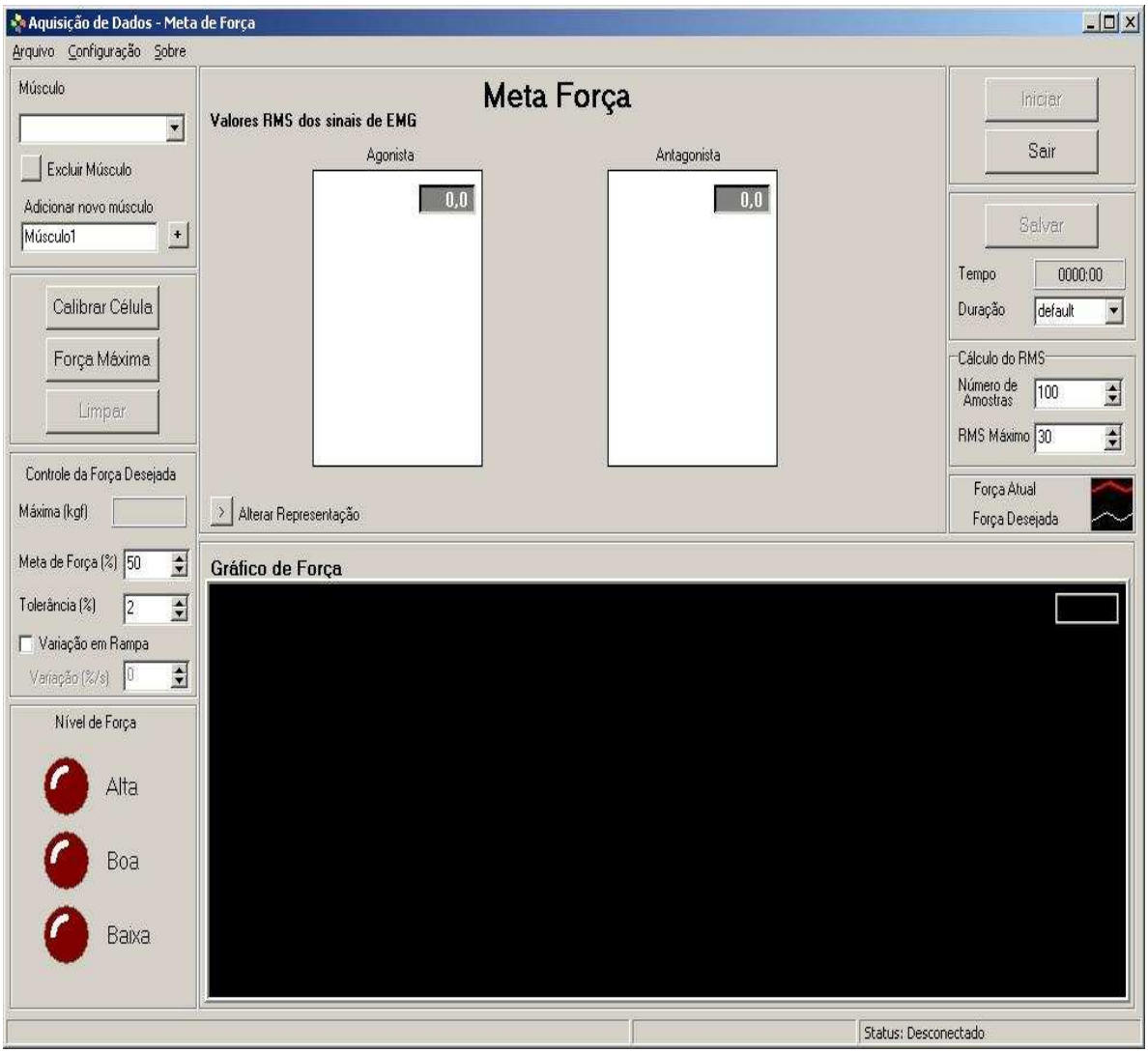


Figura 3.2 – Janela inicial do programa.

3.4. CADASTRO DE SUJEITOS

Esta é uma função essencial para o bom uso do programa desenvolvido, uma vez que o software pode ser utilizado para vários experimentos, observando-se diferentes grupos musculares de indivíduos distintos. Fazia-se necessário, portanto, que os dados provenientes das diferentes sessões de coleta fossem organizados de forma simples. Desta forma, os posteriores processamento e análise das informações obtidas, com o auxílio de softwares como o *Matlab* (Mathworks, EUA), seriam, também, simples. Isso permitiria o acompanhamento do indivíduo cadastrado ao longo do tempo – histórico e evolução.

O acesso à janela para cadastro de indivíduos é uma das opções do item “Arquivo” do menu principal. A Figura 3.3 exibe as opções do item “Arquivo”, enquanto a Figura 3.4 traz a janela para cadastro de indivíduos.

Figura 3.3 – Opções do item "Arquivo" no menu principal.

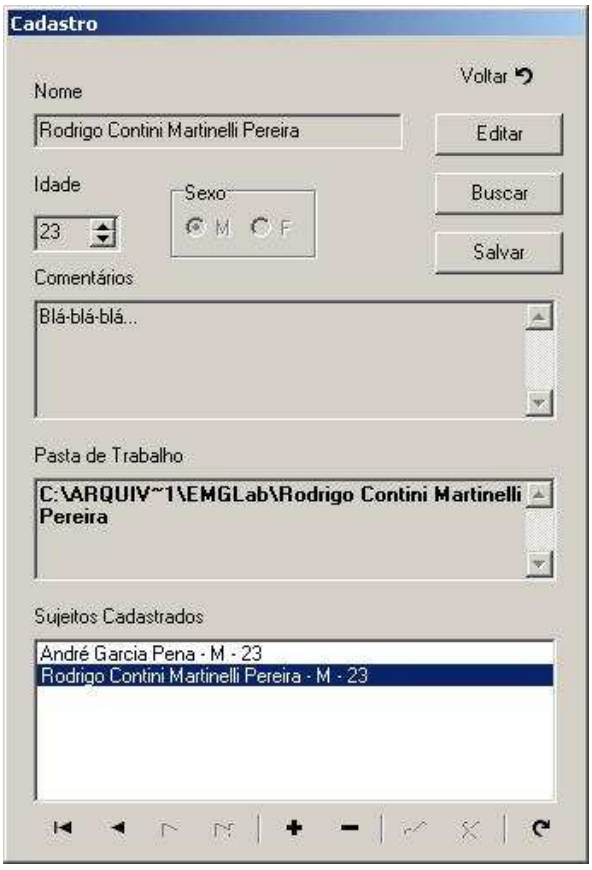


Figura 3.4 – Janela para cadastro dos indivíduos.

A estrutura de cadastro é composta por cinco campos, a saber: nome do indivíduo, idade, sexo, comentários e pasta de trabalho. O campo de comentários pode ser utilizado para a inclusão de informações relevantes à organização dos dados e sua posterior análise, como, por exemplo, as condições de realização do experimento, o número da sessão ou, ainda, informações adicionais sobre o indivíduo. O campo pasta de trabalho, por sua vez, traz o diretório no qual serão salvos os dados obtidos em experimentos realizados pelo indivíduo cadastrado.

Há nesta janela, ainda, o campo “Sujeitos Cadastrados”, que lista todos os indivíduos já cadastrados, dispostos em ordem alfabética, permitindo rápida seleção de suas pastas de trabalho, que receberão os dados obtidos na nova sessão. Abaixo deste campo, localiza-se a barra de busca de cadastros, detalhadamente exibida na Figura 3.5.

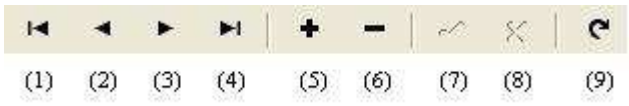


Figura 3.5 – Barra de busca de cadastros.

Os nove botões disponíveis na barra têm as seguintes funções:

- (1): seleciona o primeiro indivíduo da lista.
- (2): seleciona o indivíduo imediatamente anterior ao selecionado.
- (3): seleciona o indivíduo imediatamente posterior ao selecionado.
- (4): seleciona o último indivíduo da lista.
- (5): adiciona novo indivíduo. Acionado, este botão habilita os campos de cadastro de novo indivíduo.
- (6): exclui o indivíduo selecionado, além de sua pasta e dos dados nela contidos.
- (7): confirma o cadastramento ou alteração dos dados do indivíduo, adicionando-o à lista de sujeitos cadastrados.
- (8): cancela o cadastramento do indivíduo.
- (9): atualiza os dados.

Essa organização é necessária, pois, alocando-se os dados obtidos nas diferentes sessões em pastas únicas e exclusivas para cada indivíduo, garante-se o rápido acesso aos próprios.

A operação de cadastro, em si, é bastante simples. A seleção do botão (5) da barra de busca de cadastros habilita os campos de cadastro. Preenchidos os campos nome, idade, sexo e comentários, basta selecionar o botão (7) da barra supracitada. Feito isto, o programa exibirá, no campo pasta de trabalho, o diretório no qual serão alocados os dados relativos a experimentos realizados pelo indivíduo cadastrado.

Concluído o procedimento, basta acionar o botão “Salvar”, disposto no campo superior direito da janela, confirmando o novo cadastro.

Próximos ao botão “Salvar”, mencionado no parágrafo anterior, há outros botões, a saber:

- Voltar: fecha a janela de cadastro, retornando à janela principal do programa. Caso as alterações efetuadas na lista de cadastros não tenham sido salvas por meio do acionamento do botão “Salvar”, será perguntado se as alterações serão salvas.
- Editar: permite a alteração dos dados do sujeito selecionado.
- Buscar: abre a janela de busca por indivíduos cadastrados, exibida na Figura 3.6.

Figura 3.6 – Janela de busca por indivíduos cadastrados.

3.5. CALIBRAÇÃO DA CÉLULA DE CARGA

Uma etapa prévia à calibração da célula de carga é a seleção, na área “Músculo”, no canto superior esquerdo da tela principal, do músculo a ser estudado. Uma barra exibe os músculos anteriormente estudados, permitindo sua seleção. Além disso, há as opções de adicionar à lista um novo músculo ou remover dela aqueles já estudados.

O acionamento do botão “Calibrar Célula”, exibido no canto esquerdo da tela principal, abre a janela da calibração da célula de carga, exibida na Figura 3.7. A calibração da célula de carga é necessária, pois seu comportamento pode variar, por exemplo, com a temperatura ambiente. Essa calibração consiste na obtenção dos dados de calibração (massa e tensão elétrica de saída correspondente) e deve ser feita para cada experimento. Carrega-se a célula de carga com uma massa conhecida, indicando seu valor no campo apropriado. Acionado o botão iniciar, o programa calculará o fator de correção e o exibirá no campo “Resultado”. O fator de correção obtido garantirá a calibração da célula de carga, propiciando resultados confiáveis em unidades conhecidas.

Há, ainda, a opção de inserção manual do fator de correção, acionada com a seleção da caixa “Inserir Manualmente”. Neste caso, informa-se apenas o fator de correção a ser utilizado pelo programa.

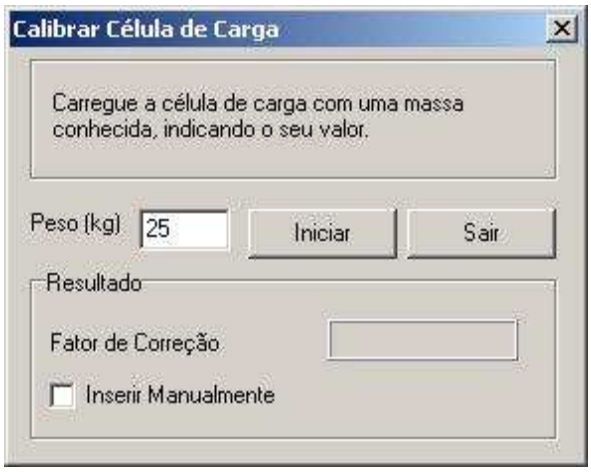


Figura 3.7 – Janela de calibração da célula de força.

3.6. CAPTURA DA FORÇA MÁXIMA

A meta de força para esse tipo de experimento é, muitas vezes, especificada como um percentual da força máxima que o indivíduo pode exercer com o músculo em estudo ao realizar um esforço não explosivo (contínuo e progressivo). O programa implementado possui uma janela para a determinação dessa força máxima, que servirá como valor de referência. A Figura 3.8 exibe essa janela.

Figura 3.8 – Janela de captura da força máxima.

Deve-se indicar o número de tentativas para a obtenção da força máxima. Sugere-se que esta seja determinada ao longo de três ou mais contrações progressivas do músculo. Nesse caso, a aquisição não deve ser interrompida entre uma contração e a seguinte. Por isso, o valor previamente ajustado é o de três tentativas. A força máxima será calculada como a maior obtida nas ‘n’ tentativas configuradas.

O botão “Iniciar” controla o início da aquisição de dados. Acionado, habilita a barra de ferramentas à esquerda da janela, bem como os campos de força máxima e força atual (em kgf), que apresentarão numericamente a situação exibida graficamente na barra.

Iniciado o processo de determinação da força máxima, a barra exibirá, na cor azul, a força atualmente exercida na célula de carga. Simultaneamente, os valores são exibidos numericamente na área “Resultados”, à direita da barra, como mencionado anteriormente. Durante a aquisição, o botão “Iniciar” passará a apresentar a inscrição “Parar” e, quando acionado, encerrará o processo.

Obtida a força máxima, e não havendo necessidade de repetição do processo, pressiona-se o botão “Sair”, que fechará a janela, retornando à tela principal. Esta já trará, na área “Controle da Força Desejada”, à esquerda, a força máxima obtida. O programa só permite a realização do experimento se a força máxima obtida for maior que zero, indicando que houve a captura da mesma.

3.7. AQUISIÇÃO COM META DE FORÇA

Esse tipo de teste consiste no esforço muscular exercido pelo indivíduo na tentativa de acompanhar um perfil de força determinado, dentro de uma faixa de tolerância pré-definida. Normalmente, o perfil é definido como uma porcentagem constante da força máxima medida para o músculo estudado, embora o programa forneça a opção da criação de perfis com variação do tipo rampa.

Cumpridas as etapas anteriores (seleção do músculo, calibração da célula de carga e obtenção da força máxima), pode-se iniciar o experimento de aquisição com meta de força. Para isso, na área “Controle da Força Desejada”, no canto esquerdo da janela principal, devem-se definir a meta de força e a tolerância. Estas são definidas em valores percentuais relativos à força máxima anteriormente obtida. Nesta mesma área, pode-se optar pela variação em rampa, definindo-se um percentual de variação do perfil a cada segundo. Desta forma, o sujeito deve seguir uma curva de variação de força, e não simplesmente uma meta constante.

A área “Gráfico de Força”, disposta na parte inferior da janela principal, traz as duas linhas de tolerância da meta de força, de acordo com as definições do parágrafo anterior. Originalmente, a cor das linhas de tolerância é vermelha, podendo, no entanto, ser alterada na opção “Aparência”, do item “Configuração” do menu principal. Iniciado o experimento, a partir do acionamento do botão “Iniciar”, no canto esquerdo da tela principal, o sinal de força capturado pela célula de carga será exibido na cor verde (na configuração padrão), sobreposto

às linhas de tolerância. Desta forma, o indivíduo, acompanhando em tempo real o gráfico que sobrepõe o esforço exercido à meta pré-determinada, pode variar a força exercida convenientemente, mantendo-a dentro dos limites estipulados.

No canto inferior esquerdo da janela, na área “Nível de Força”, há três LEDs (círculos luminosos), que se referem à força exercida, comparando-a aos limites estabelecidos. Eles representam:

- Alta: a força exercida é superior aos limites de tolerância estipulados.
- Boa: a força exercida encontra-se dentro dos limites de tolerância estipulados.
- Baixa: a força exercida é inferior aos limites de tolerância estipulados.

Durante a captação, um dos LEDs estará sempre aceso (em verde), relacionando a força exercida ao perfil traçado e exibido na área “Gráfico de Força”. As Figuras 3.9 e 3.10 ilustram aquilo que foi aqui descrito.

Paralelamente à captação do sinal de força está a captação dos eletromiogramas dos músculos agonista e antagonista. O cálculo do valor RMS é feito com base em uma janela deslizante, e seu número de amostras pode ser definido na área “Cálculo do RMS”, no canto direito da tela.

Os valores RMS desses sinais são exibidos na área “Valores RMS dos sinais de EMG”, na parte superior da janela. Há duas representações possíveis para esses sinais: a primeira, por meio de duas barras, de forma semelhante àquela utilizada para a obtenção da força máxima; e a segunda, por meio de um gráfico linear, semelhante ao gráfico de força. A alteração da representação exibida é feita a partir do acionamento do botão “Altera Representação”. A Figura 3.9 exibe a primeira representação, enquanto a Figura 3.10 traz a segunda.

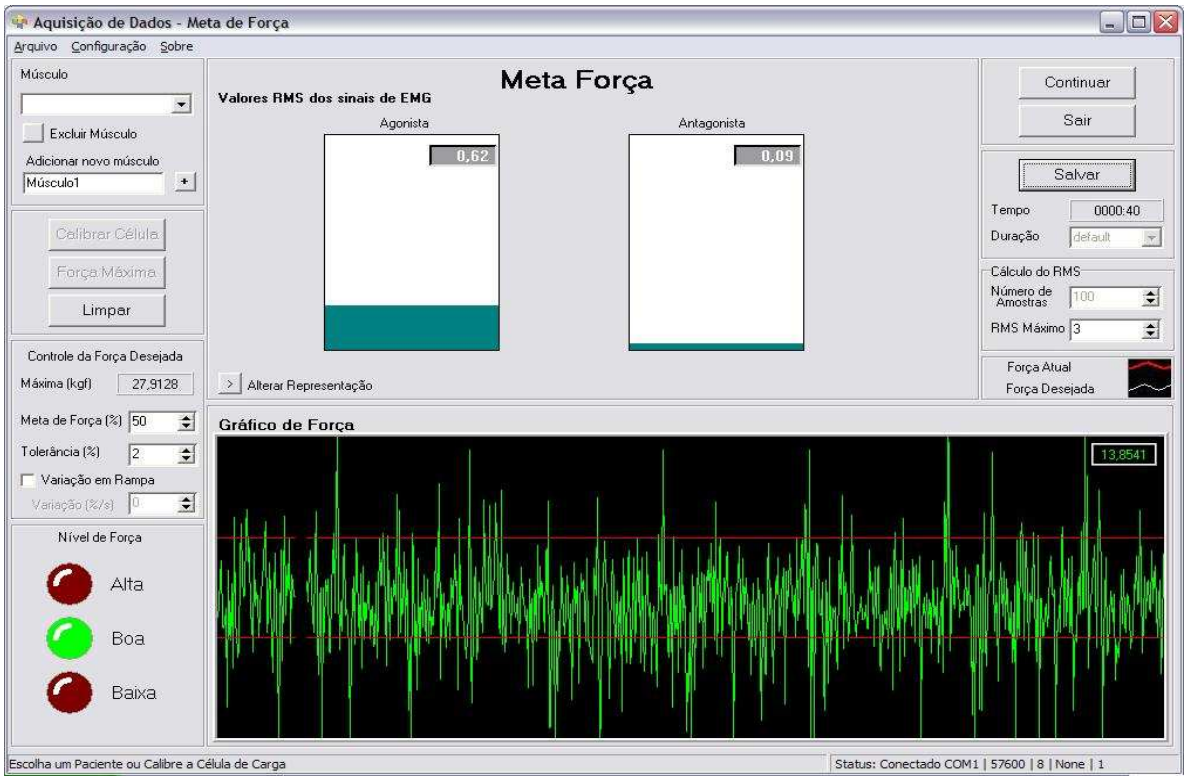


Figura 3.9 – Captação dos sinais de força e valor RMS dos eletromiogramas representados por barras.

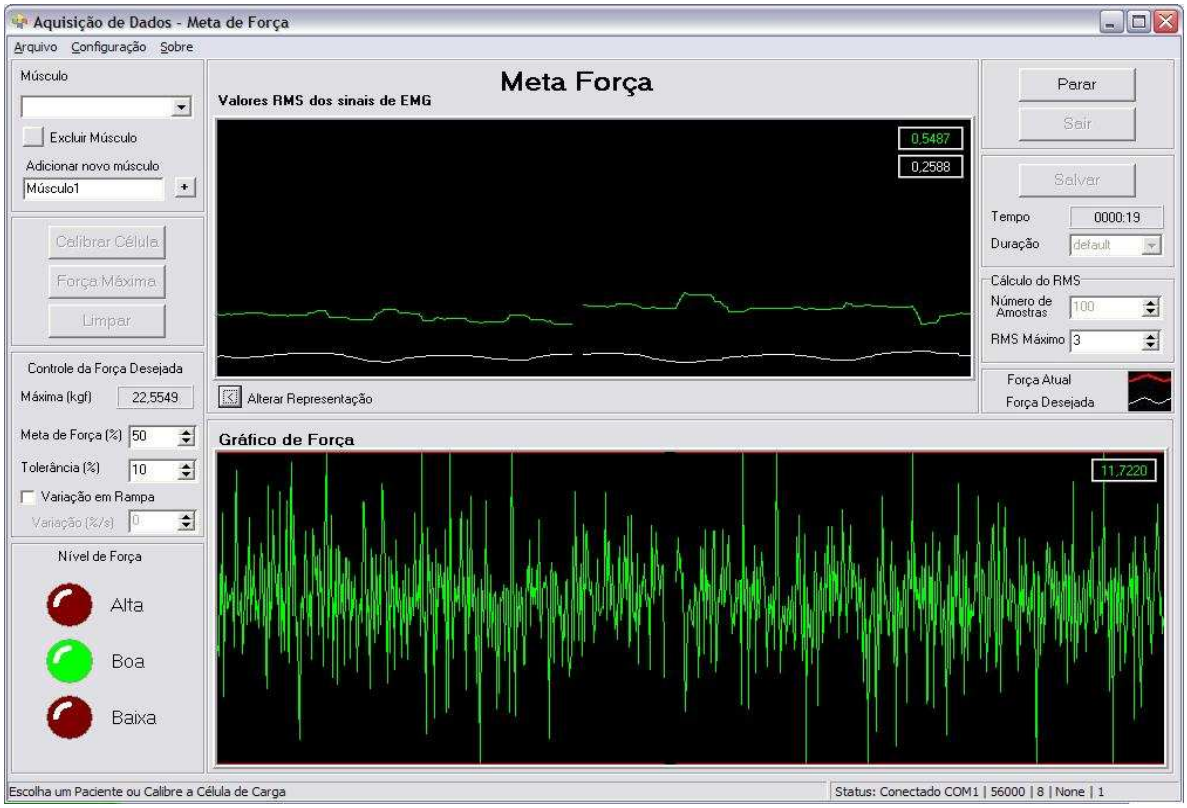


Figura 3.10 – Captação dos sinais de força e valor RMS dos eletromiogramas representados linearmente.

Os resultados capturados podem ser salvos em arquivos de texto em pastas organizadas, permitindo sua posterior análise, conforme explicado no início do capítulo. Para isso, pressiona-se o botão “Salvar”, no canto direito da janela.

3.8. OUTRAS FUNÇÕES

Além da aquisição com meta de força, o cerne do programa, e das operações correlatas, o programa disponibiliza, ainda, outras opções. Uma delas é a configuração da aparência, através da opção “Aparência”, no item “Configuração” do menu principal, como ilustra a Figura 3.11.

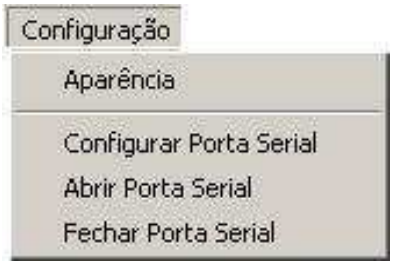


Figura 3.11 – Opções do item “Configuração” no menu principal.

A seleção do item “Aparência” abrirá a janela de configuração de aparência, exibida na Figura 3.12. Essa janela permite a configuração das cores dos gráficos e tanques (barras) exibidos durante o processo de aquisição de acordo com a preferência do usuário, sem merecer maiores comentários.

Figura 3.12 – Janela para configuração da aparência do programa.

Outra função desse item do menu principal é acessada pela opção “Configurar Porta Serial”. Feito isso, será aberta a janela de mesmo nome, que possibilitará a escolha da porta, da taxa, o número de bits de dado, e de parada, além da paridade.

Figura 3.13 – Janela para configuração da porta serial.

As opções “Abrir Porta Serial” e “Fechar Porta Serial”, servem para se habilitar a comunicação serial, permitindo a recepção de dados serialmente. Esses procedimentos são executados automaticamente quando se inicia ou termina uma captura.

3.9. PRINCIPAIS CÓDIGOS IMPLEMENTADOS

Dentre as diversas funções e procedimentos necessários para o funcionamento do programa, há aquelas que desempenham as principais tarefas. A seguir, serão discutidas e explicadas essas funções, de maneira a tornar claro o código desenvolvido e facilitar futuras alterações por outras pessoas.

3.9.1. Função CPortRxChar

A função CPortRxChar( ) é a responsável por receber os dados enviados pelo MSP através da porta serial. Ela é chamada toda vez em que há novos dados recebidos, identificando qual o canal que lhe enviou o dado e qual o seu valor. Para cada valor, a ela também chama a função Plota(canal,amostra) para que o mesmo seja mostrado na tela ou na forma de gráfico ou nas barras. A Figura 3.14 mostra o código desenvolvido.

```
//---------------------------------------------------------------------------
void __fastcall TfrmPrincipal::CPortRxChar(TObject *Sender, int Count)
{
01   unsigned char buffer[];
02   unsigned int lenrx;
03   static unsigned int canal;
04   unsigned int canal_temp;
05   static unsigned int amostra_temp;
06   unsigned int amostra;
07   static bool AmostraFlag=false;
08   unsigned int k;
09
10   if(!flag_plota)
11     return;
12   lenrx = CPort->Read(buffer,Count);
13   if(flagCapturando)
14     fwrite(buffer,sizeof(char),lenrx,pt1);
15
16   for(k=0;k<lenrx;k++)
17   {
18     canal_temp = (((unsigned int)(buffer[k]))>>4)&0x0F;
19     if((canal_temp == 0 || canal_temp == 1 || canal_temp == 2 ||
20        canal_temp == 3)&&(!AmostraFlag))
21     {
22       amostra_temp =  (((unsigned int)(buffer[k]))<<8)&0x0F00;
23       AmostraFlag=true;
24       canal=canal_temp;
25     }
26     else if(AmostraFlag)
27     {
28       amostra = amostra_temp + (((unsigned int)(buffer[k]))&0x00FF);
29       Plota(canal,amostra);
30       AmostraFlag=false;
31     }
32    }
33}
```

Figura 3.14 – Código da função CPortRxChar( ).

Na linha 12, tem-se que a função Read(buffer,Count) lê os dados contidos no buffer de recepção e armazena-os na variável do tipo char buffer. Em seguida, na linha 14, salvam-se os dados em um arquivo utilizando a função fwrite. Finalmente, o laço for( ), contido entre as

linhas 16 e 32, serve para se avaliar qual o canal enviou a informação e qual o seu respectivo valor.

A transmissão dos dados pelo MSP é feita segundo um protocolo amplamente utilizado por diversos trabalhos do Departamento de Engenharia Elétrica da Universidade de Brasília. Foi uma ótima solução que maximizou o processamento dos dados. Considerando-se que se faz necessário enviar 12 bits (precisão do conversor A/D) mais a informação do canal e que a UART envia no máximo 8 bits, seriam necessárias três transmissões para poder mandar todos os bits: primeiro byte, quatro bits que restaram e o canal. Como há apenas quatro canais, percebe-se que a informação do canal ocupa apenas 2 bits, portanto, a solução desenvolvida foi mandar o número do canal junto com os quatro bits restantes do dado. Desta maneira, precisa-se enviar apenas dois bytes para transferir toda a informação. A Figura 3.15 mostra o quadro padrão do protocolo de comunicação desenvolvido.

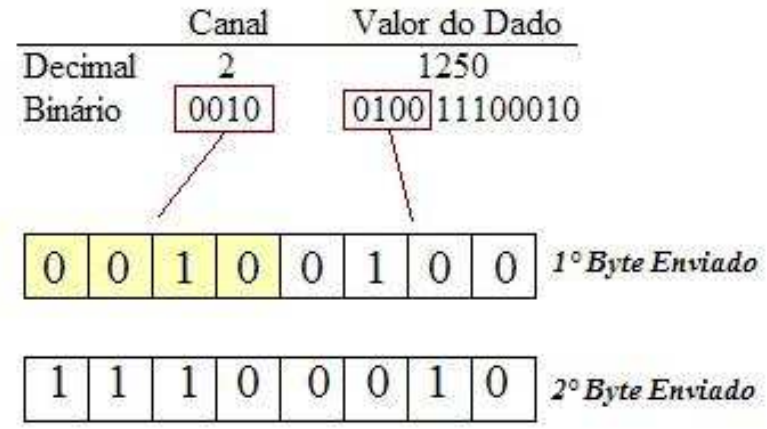


Figura 3.15 - Quadro do protocolo de comunicação.

3.9.2. Função Plota

A função Plota(Canal, Valor) é responsável por mostrar na tela os valores enviados pelo MSP. A variável Canal indica onde se deve plotar o Valor que foi passado à função. A Tabela 3.1 indica como a função Plota trata a variável Canal.

Tabela 3.1 – Locais onde são mostrados os valores obtidos serialmente.

Canal	Representação	Local onde será plotado o dado
0	-	Gráfico de força
1	Barra	Barra Agonista
1	Gráfico	Gráfico de RMS
2	Barra	Barra Antagonista
2	Gráfico	Gráfico de RMS

3.9.3. Função CalcRMS

A função CalcRMS(Canal, Valor) retorna o valor RMS de um vetor que armazena os valores dos sinais de EMG dos músculos agonista (Canal 1) e antagonista (Canal 2). Para um sinal com 'N' amostras, tem-se que o valor RMS será dado por:

$$RMS = \sqrt{\frac{1}{N}\sum_{k=0}^{N-1} EMG[k]^2} \tag{3.1}$$

Inicialmente, converte-se o valor recebido em valor de tensão, conforme Equação 4.2 abordada posteriormente. Armazena-se este valor em um vetor, conforme observado nas linhas 6 e 14 da Figura 3.16, de tamanho definido pelo número de amostras para o cálculo do RMS (Variável Winlen). Em seguida para o cálculo do valor RMS utilizam-se laços for( ) (linhas 7-8 e 15-16) que executam a seguinte operação:

$$\sum_{k=0}^{Winlen} EMG[k]^2 \tag{3.2}$$

Finalmente, a função retorna o valor RMS do vetor armazenado ao executar a operação da linha 18.

```
//---------------------------------------------------------------------------
float __fastcall TfrmPrincipal::CalcRMS(int canal, unsigned int valor)
{
  double ValorTensao, aux=0;

01  ValorTensao =((double)valor) * 25 / 40950; //Converte em Tensão
02  if(canal == 1)  //Agonista
03  {
04    if(iAgo > (int)Winlen)
05      iAgo = 0;
06    Agonista[iAgo++] = ValorTensao;
07   for(i=0;i<Winlen;i++)
08     aux += pow(Agonista[i],2);
09  }
10  else if(canal == 2)  // Antagonista
11  {
12    if(iAntago > (int)Winlen)
13      iAntago = 0;
14    Antagonista[iAntago++] = ValorTensao;
15    for(i=0;i<Winlen;i++)
16      aux += pow(Antagonista[i],2);
17  }
18  return(sqrt(aux/Winlen));
}
```

Figura 3.16 – Código da função CalcRMS(Canal, Valor).

3.10. ESTRUTURA DE PASTAS E ARQUIVOS GERADA PELO PROGRAMA

Para manter a organização dos dados, o programa possui uma estrutura própria de pastas e arquivos. Ao ser executado pela primeira vez, ele cria a pasta *Dados* no diretório onde fora instalado. Simultaneamente, cria-se o arquivo *Musculos.txt*, responsável por armazenar os músculos adicionados ao programa.

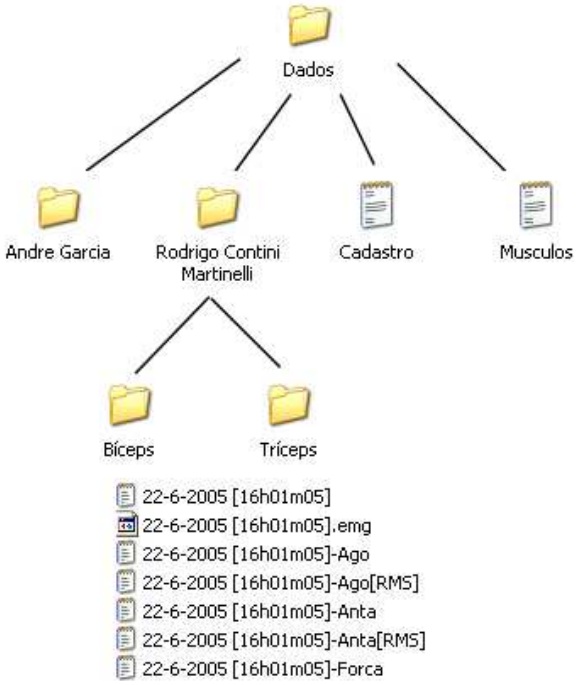
Para cada pessoa adicionada ao banco de dados, o programa cria uma pasta para ela e uma subpasta para cada músculo cadastrado. Após a adição, salva-se o banco de dados no arquivo *Cadastro.txt*, dentro da pasta *Dados*. Nele são armazenadas todas as informações contidas no cadastro dos sujeitos (nome, idade, sexo, comentários e pasta de trabalho).

Finalmente, cada vez que é realizada uma aquisição de dados e selecionada a opção de salvá-los, um arquivo, contendo os valores dos dados adquiridos, é criado para cada sinal capturado pelo programa. Além disso, um arquivo é responsável por armazenar os dados da aquisição, a citar: data e hora da realização do experimento, músculo em estudo, fator de calibração da célula de carga, força máxima, meta de força, tolerância e número de amostras para o cálculo do RMS. A Tabela 3.2 mostra os arquivos criados e sua respectiva designação. A Figura 3.17 mostra um exemplo de estrutura de pastas e arquivos criados pelo programa, além de ilustrar o conteúdo de cada tipo de arquivo texto.

Tabela 3.2 - Arquivos gerados pelo software.

Nome do Arquivo	Designação
DD-ME-AAAA [HHhMMmSS].emg	Arquivo binário utilizado pelo software para rever sinal.
DD-ME-AAAA [HHhMMmSS].txt	Armazena os dados da aquisição: Força máxima, Meta,etc.
DD-ME-AAAA [HHhMMmSS]-Ago.txt	Armazena os dados de EMG do músculo Agonista.
DD-ME-AAAA [HHhMMmSS]-Ago[RMS].txt	Armazena os dados do RMS do EMG do músculo Agonista
DD-ME-AAAA [HHhMMmSS]-Anta.txt	Armazena os dados de EMG do músculo Antagonista.
DD-ME-AAAA [HHhMMmSS]-Anta[RMS].txt	Armazena os dados do RMS do EMG Antagonista
DD-ME-AAAA [HHhMMmSS]-Força.txt	Armazena os dados da força.

Legenda: DD – Dia ME – Mês AAAA – Ano HH – Hora MM – Minuto SS – Segundo



Cadastro.txt

```
Nome:   Andre Garcia
Sexo:   M
Idade:  23
Pasta de Trabalho:      C:\Arquivos de programas\EMGLab\Andre Garcia
Comentários:
01/35208

------------------------------------------------------------------------
Nome:   Rodrigo Contini Martinelli
Sexo:   M
Idade:  24
Pasta de Trabalho:      C:\ARQUIV~1\EMGLab\Rodrigo Contini Martinelli
Comentários:
10/05/2005

------------------------------------------------------------------------
```

Músculos.txt

```
Bíceps
Tríceps
```

22-6-2005 [16h01m05].txt

```
                DADOS DA AQUISIÇÃO
Data:   22/6/2005                       Hora:   16:01:18
------------------------------------------------------------
Músculo:                                Bíceps
Fator de Conversão Força/Tensão:        0,0120000001043081
Força Máxima (kgf):                     24
Meta de Força (%):                      50
Tolerância (%):                         2
N° de Amostras Cálculo RMS:             100
```

Figura 3.17 – Exemplo da estrutura de pastas e arquivos criados.

Capítulo 4: HARDWARE DE AQUISIÇÃO

4.1. DETALHAMENTO DO SISTEMA

O hardware de aquisição tem por objetivo adquirir os sinais provenientes da célula de carga e dos músculos agonista e antagonista em estudo, enviando-os ao computador para assim iniciar o processamento dos dados. O circuito, alimentado por uma bateria de 9 V, possui quatro canais de aquisição e utiliza o padrão RS-232C para comunicar-se com o computador.

Basicamente, é composto por quatro módulos principais:

i) *Módulo de Alimentação*: responsável por fornecer os níveis de tensão desejados para o correto funcionamento dos circuitos da placa;

ii) *MSP430*: microcontrolador responsável pela digitalização dos sinais e transmissão dos dados ao computador;

iii) *MAX232*: CI responsável pela conversão dos sinais transmitidos pelo MSP para os níveis de tensão do padrão RS-232.

iv) *Parte Analógica*: corresponde ao sistema externo composto pelos eletromiógrafos da Delsys, Célula de Carga e Trigger.

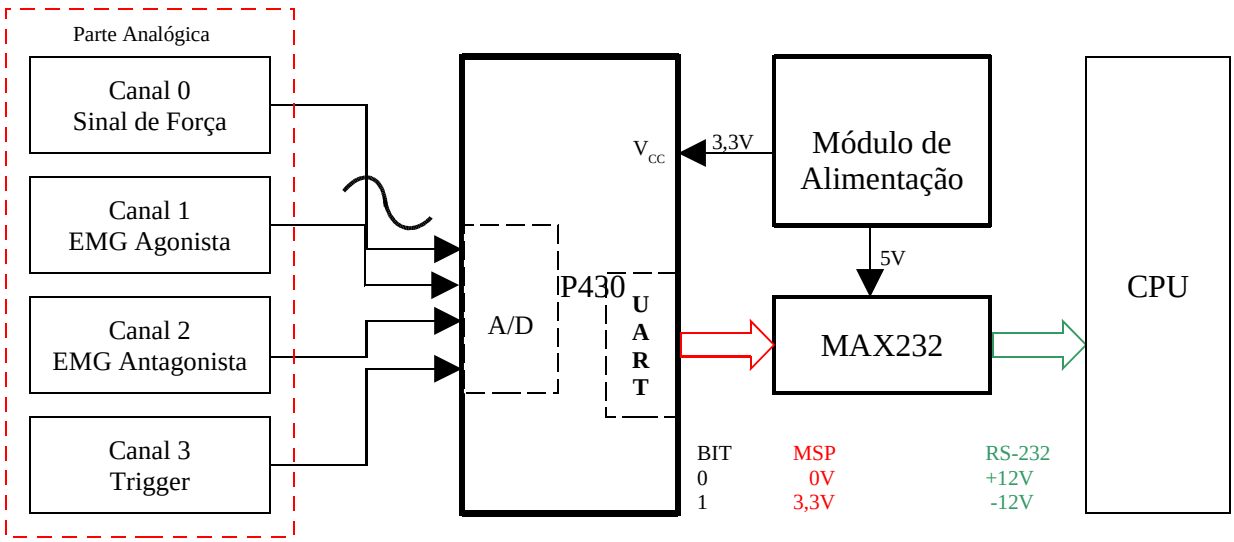


Figura 4.1 - Diagrama de blocos do hardware de aquisição.

4.1.1. Módulo de alimentação

O módulo de alimentação converte a tensão de 9 V fornecida pela bateria em dois níveis de tensão: 3,3 V para alimentar o MSP e 5 V para alimentar o MAX232.

Foram utilizados reguladores de tensão para obter as tensões desejadas. A vantagem desse tipo de topologia é a obtenção de níveis de tensão estáveis, isto é, para uma determinada faixa de variação da tensão de entrada, o regulador mantém a saída constante, mantendo o funcionamento correto do circuito, além de conseguir um sinal mais próximo do requerido e confiável.

Utilizou-se o Regulador de Tensão LP2950CZ5.0 da National Semiconductor que fornece uma tensão de saída de 5 V para uma tensão de entrada de até 30 V. A Figura 4.2 mostra suas conexões.

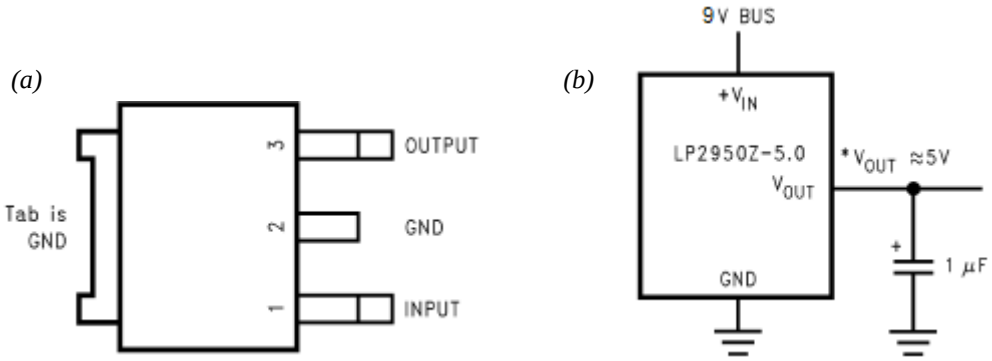


Figura 4.2 - (a) Regulador de Tensão LP2950CZ5.0; (b) Esquema de Ligação.

Similarmente, para obter o nível de tensão de 3,3 V para alimentar o MSP utilizou-se o Regulador de Tensão TPS76133 da Texas Instruments e como tensão de entrada os 5 V fornecidos pelo LP2950CZ5.0. Esse regulador opera com níveis de entrada na faixa de 3,68 a 16 V, fornecendo uma saída de tensão contínua de 3,3 V.

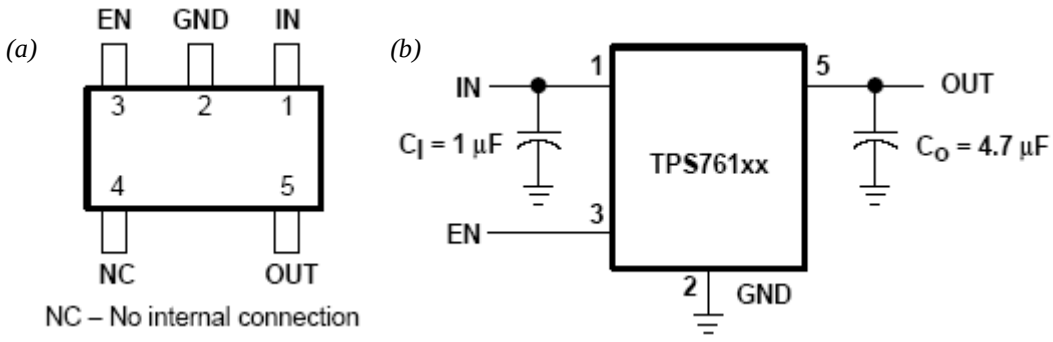


Figura 4.3 - (a) Regulador de Tensão TPS 76133; (b) Esquema de Ligação.

4.1.2. Microcontrolador MSP

O MSP430 é uma família de microcontroladores da Texas Instruments. Possui um moderno processador (de tecnologia RISC) de 16-bits, capaz de processar instruções em bytes ou em palavras (16 bits), contando com um conjunto de 27 instruções e 7 formas de endereçamento. É um dispositivo de elevado desempenho projetado para o consumo baixo de potência, amplamente utilizado em sistemas alimentados por baterias ou aplicações de longa duração.

Possui uma elevada versatilidade por conter internamente diversos periféricos como temporizadores, conversor analógico-digital de 12 bits, USART (Universal Synchronous-Asynchronous Receiver/Transmitter) para transmissão serial síncrona ou assíncrona, hardware multiplicador, controlador de *display* e um comparador. Possui 6 portas digitais, com 8 bits cada, para entrada/saída de dados.

O MSP permite a utilização de diversas fontes flexíveis de relógio (*clock*), podendo-se utilizar relógio interno ou externo. Permite a alteração do relógio a qualquer tempo, viabilizando, assim, para o desenvolvedor aumentar ou diminuir a velocidade de execução das instruções de acordo com a sua necessidade. A Figura 4.4 mostra os diversos pinos e suas respectivas designações para o MSP430F149, e a Figura 4.5 mostra o diagrama de blocos funcional do microcontrolador.

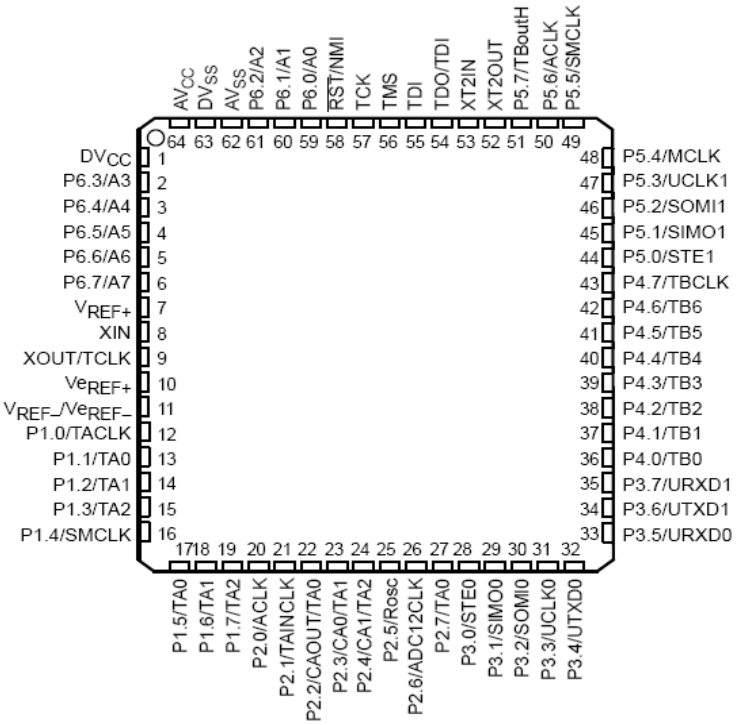


Figura 4.4 - Pinagem do MSP430F149.

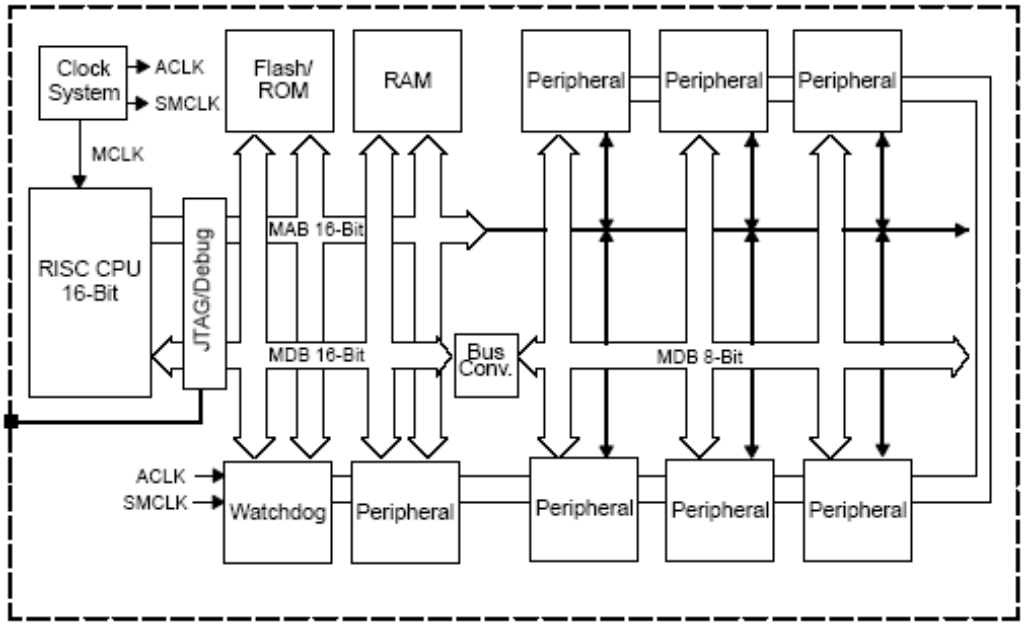


Figura 4.5 - Diagrama de blocos funcional.

A aquisição do sinal analógico foi realizada utilizando-se o conversor A/D (analógico-digital) existente no microcontrolador. Quatro canais, dentre os 16 possíveis, amostram os sinais provenientes da célula de carga, sincronismo (*trigger*), EMG agonista e antagonista, utilizando um circuito *sample and hold*. As tensões de referência podem ser programadas para ser externas ou internas de maneira que o valor da conversão obtido é dado pela fórmula:

$$N_{ADC} = 4095x\frac{V_{in} - V_{R-}}{V_{R+} - V_{R-}} \tag{4.1}$$

Onde:

N_{ADC}: Resultado da conversão compreendido entre 0h, quando $V_{in} = V_{R-}$, e FFFh, quando $V_{in} = V_{R+}$.
V_{in}: Valor do sinal no momento da conversão.
V_{R+} e V_{R-}: Tensões de referência.

O sistema foi programado para trabalhar com tensões de referência interna, configurando $V_{R+} = 2{,}5V$ e $V_{R-} = 0$, resultando na seguinte fórmula para a conversão:

$$N_{ADC} = 4095x\frac{V_{in}}{2{,}5} = 1638xV_{in} \tag{4.2}$$

Assim como qualquer conversor A/D de alta resolução, técnicas apropriadas de aterramento devem ser implementadas para eliminar ruídos, retorno de corrente, etc. Esses

ruídos podem se adicionar ao sinal adquirido, comprometendo todo o processo. A Figura 4.6 mostra a conexão, proposta pelo fabricante, necessária para minimizar esse problema.

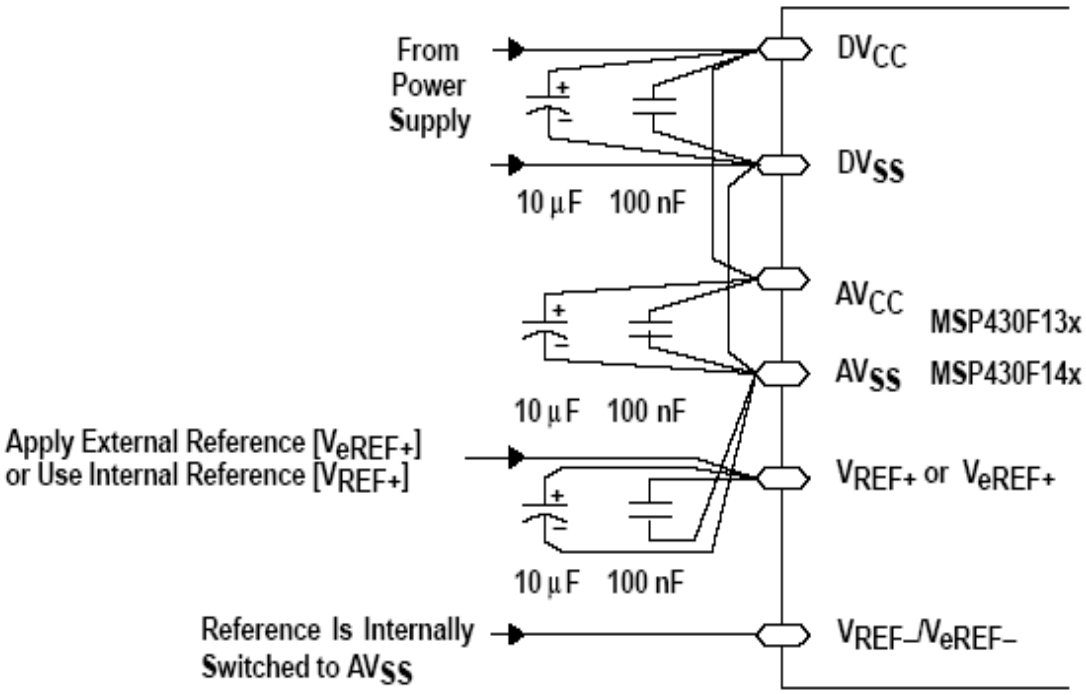


Figura 4.6 - Conexão utilizada para minimizar ruídos.

A taxa de amostragem foi determinada através da banda de cada sinal adquirido. Segundo o teorema da amostragem, para se conseguir recompor um sinal amostrado, este deverá ser amostrado a uma freqüência de amostragem mínima igual ao dobro da largura de banda do mesmo sinal, isto é:

$$f_s \geq 2B \quad [Hz] \text{ ou } t_s \leq \frac{1}{2B} \quad [s] \tag{4.3}$$

Onde:

f_s: Taxa de Amostragem [Hz]
t_s: Período de Amostragem [s]
B: Banda do Sinal [Hz]

Através das interrupções do timer, onde a cada interrupção gerada, uma nova amostra do sinal é obtida, determinou-se a taxa de amostragem de cada sinal. A Tabela 4.1 mostra qual a taxa de amostragem utilizada para cada canal e a largura de banda de cada sinal:

Tabela 4.1 - Taxa de amostragem utilizada em cada canal.

Canal	Descrição	Banda (Hz)	Taxa de Amostragem (Hz)
0	Célula de Carga*	25	50
1	EMG Agonista	500	1000
2	EMG Antagonista	500	1000
3	Trigger*	25	50

* Sabendo que os sinais de força e trigger são sinais lentos, i. e., possuem baixas componentes de freqüência, estipulou-se sua banda como sendo de 25 Hz de maneira a garantir uma boa amostragem e evitar perdas.

A comunicação serial do MSP com o computador é feita através da USART operando em seu modo assíncrono. Ela é responsável por preparar o quadro da comunicação, bastando para o usuário colocar o byte a ser transmitido no *buffer* de transmissão. Através de seus registradores, configuram-se as características da comunicação, tais como: taxa de transmissão, bits de parada, número de bits, e o bit de paridade. A Figura 4.7 mostra um quadro típico da comunicação serial.

Figura 4.7 - Quadro (*frame*) de uma comunicação serial.

Para o projeto, a comunicação serial ficou configurada como:

Tabela 4.2 - Configuração da comunicação serial.

Parâmetro	
Baud Rate	115200 bps
Bits	8
Paridade	None
Bits de Parada	2

4.1.3. MAX232

O padrão RS-232C, apesar de antigo, é ainda amplamente utilizado na conexão entre dois equipamentos. Muitos dispositivos, tais como terminais, *ploters*, osciloscópios, impressoras e unidades de fita, interconectam-se através da porta serial RS-232C. Em qualquer aplicação prática, é necessário utilizar circuitos que convertam os níveis TTL para os exigidos pela interface e também conversões para o caminho inverso.

O circuito integrado MAX232 da Texas Instruments é o responsável pela conversão dos níveis de tensão do sinal transmitido serialmente pelo MSP para níveis utilizados no padrão RS-232C. A regra da conversão é muito simples: nível alto TTL é convertido em -12 V e nível baixo TTL em +12 V. A Tabela 4.3 mostra a conversão dos sinais e a Figura 4.8 detalha os níveis de tensão válidos para a comunicação serial RS-232.

Tabela 4.3 - Interpretação dos sinais marca e espaço.

	TTL		RS – 232C	
Sinal	Tensão	Lógico	Tensão	Controle
Espaço	0 V	0	+12 V	ON
Marca	5 V	1	- 12 V	OFF

Figura 4.8 - Níveis de tensão válidos.

A Figura 4.9 apresenta as conexões do MAX232 ao MSP, bem como a ligação ao conector do tipo DB9 (nove pinos) fêmea e a ligação dos capacitores externos necessários ao seu funcionamento.

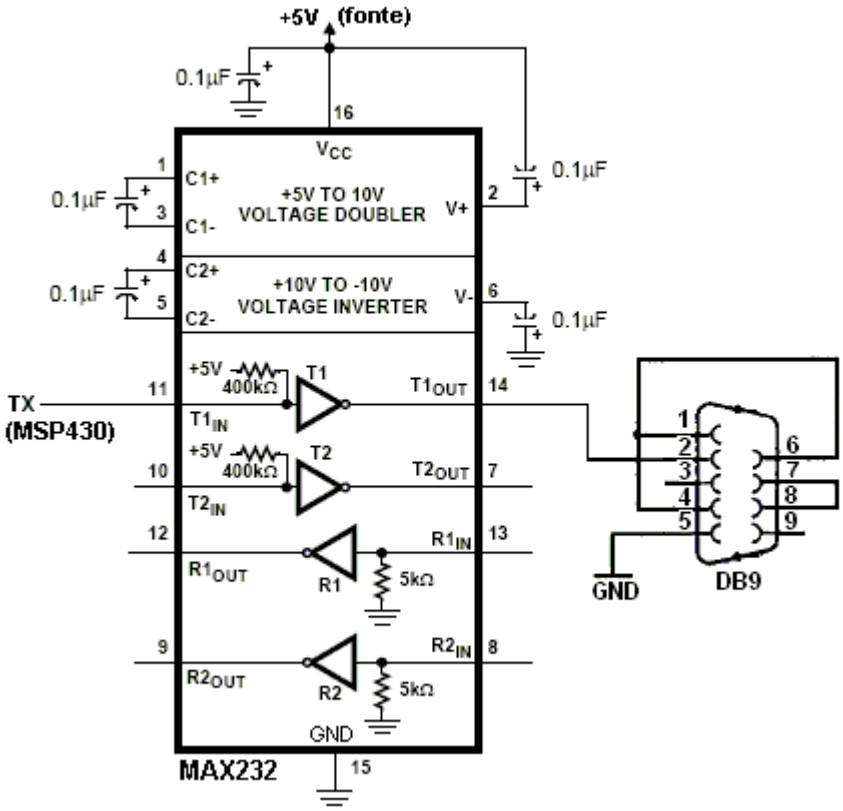


Figura 4. 9 - Ligação do MAX232 ao MSP.

4.1.4. Parte analógica

Essa parte, externa à placa, corresponde aos circuitos de aquisição/conversão dos sinais analógicos, sendo eles:

i) *Célula de Carga*: responsável por converter a força exercida por um indivíduo em sinais elétricos. Essa conversão é feita utilizando-se extensômetros (*strain-gauges*) presos a uma superfície que receberá a força.

ii) *Elétromiógrafo Bagnoli-2 da Delsys e eletrodos*: responsáveis por captar e amplificar o sinal de EMG. Composto por eletrodos bipolares ativos que fornecem uma pré-amplificação do sinal, aumentando a relação sinal-ruído, e por eletromiógrafos da Delsys (Boston, EUA) que permite um ganho variável do sinal. A Figura 4.10 mostra esses dispositivos.

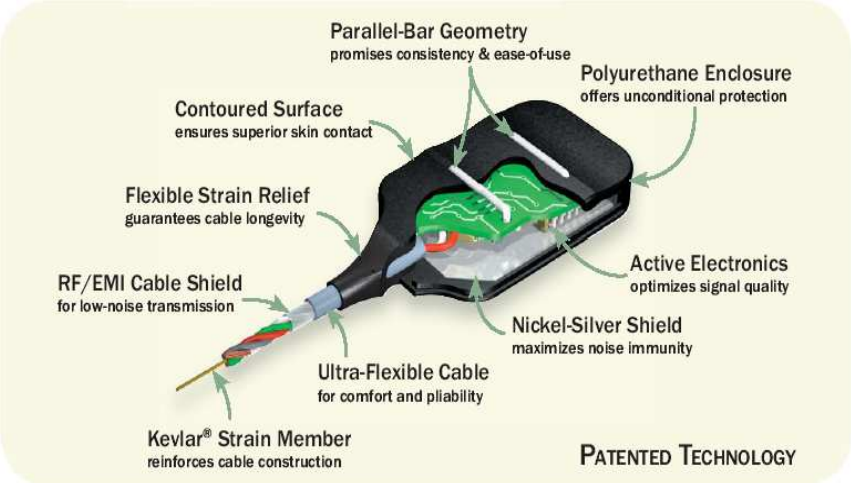


Figura 4.10 - Eletromiógrafo e eletrodos ativos da Delsys.

iii) Trigger: O sinal de *trigger* corresponde a um trem de pulsos gerado por um dispositivo externo ao sistema de aquisição de dados (e de responsabilidade do usuário), cuja freqüência é bem definida para a marcação de passos em experimentos que o requeiram.

4.2. MONTAGEM DO CIRCUITO

4.2.1. Placas de circuito impresso

A placa de circuito impresso, também denominada de PCB (*Printed Circuit Board*), tem como sua função básica proporcionar suporte mecânico e interligação elétrica para os componentes utilizados no circuito eletrônico. As exigências cada vez maiores no que se diz respeito ao desempenho mecânico e elétrico mostram que projetar uma placa de circuito

impresso exige que sejam respeitados alguns requisitos básicos para que seja atingido o objetivo final de fabricar uma PCB com qualidade.

A concepção de uma placa de circuito impresso parte de dois pontos principais: *o projeto mecânico e o projeto elétrico*. O projeto mecânico leva em consideração detalhes estéticos e funcionais, tais como emissores luminosos, que deverão aparecer externamente ao gabinete, posição de chaves e botões, localização de componentes críticos, como transistores e resistores de potência, e componentes que possam deformar a placa em função do seu peso, como transformadores.

Geralmente, a geometria da placa está limitada também às dimensões do gabinete onde ela será acondicionada, e não são raras as situações onde é necessário implementar o projeto elétrico em um espaço muito crítico. Porém, vale ressaltar que restrições excessivamente rígidas quanto ao espaço disponível para o circuito eletrônico levam a situações onde é necessário diminuir a largura/espaçamento de trilhas, ou até aumentar o número de camadas de cobre, aumentando desnecessariamente o custo de fabricação da placa.

O projeto elétrico é o que define uma funcionalidade para a placa de circuito impresso. Devem ser considerados alguns aspectos básicos no projeto elétrico visando possível redução no custo de fabricação da placa, bem como melhorar a qualidade final do equipamento. Sempre que possível, deve-se dimensionar os componentes de potência adequadamente. Superdimensionar estes componentes pode acarretar em aumento da área ocupada por eles, sem falar no custo do próprio componente, o que em muitas vezes é maior. Deve-se prever dissipadores de calor para componentes que aquecem muito. Quando o número de placas/mês for elevado, é justificável pensar em formas de automatizar ao máximo a montagem de placas. Nesses casos deve-se optar por utilizar componentes de montagem em superfície (SMD – *surface mounted device*), os quais permitem a montagem por equipamentos de inserção automática.

Para equipamentos que possuam várias versões, deve-se projetar placas que permitam sub-equipação, ou seja, a mesma placa pode ser utilizada em versões de equipamento diferentes, bastando suprimir/acrescentar componentes que sejam diferentes em ambas as versões prevendo facilidades de montagem.

4.2.1.1. Tipos de placa de circuito impresso

Existem três tipos básicos de placas de circuito impresso: a de face simples, a de dupla face e a multicamada (*multilayer*).

i) *Face simples*: São confeccionadas tanto em fenolite como em fibra de vidro. São utilizadas em montagens simples, onde o grau de complexidade de conexões e quantidade de componentes é pequeno e onde o tamanho reduzido da placa não é relevante. As trilhas localizam-se apenas em uma face da placa, independentemente da montagem usar componentes SMD ou convencionais. No caso de montagem ser feita com componentes convencionais há a delimitação clara do lado chamado dos componentes e do lado da solda.

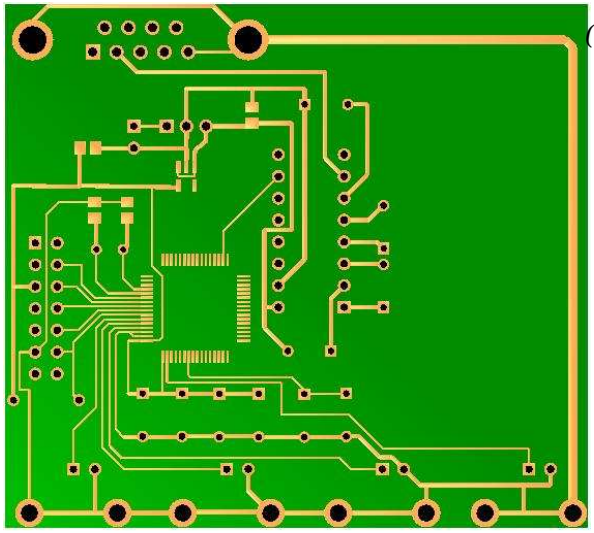
ii) *Dupla Face*: São confeccionadas em sua grande maioria em fibra de vidro. São empregadas em montagens mais complexas, onde a quantidade de componentes é maior e a necessidade de menores dimensões é indispensável. Existem trilhas em ambas as faces da placa de circuito impresso, inclusive na face dos componentes.

iii) *Multicamada*: São assim chamadas porque existem duas, três, quatro ou mais camadas de trilhas com os respectivos furos de passagem para interligação entre as camadas, dependendo do nível tecnológico e do grau de complexidade do equipamento. Pode ser aplicado a qualquer número de camadas, sendo que a placa é constituída, na realidade, de um conjunto de “n” placas de face simples e mais uma placa de dupla face. Dessa forma existe uma grande complexidade nesse tipo de placa e um grau elevado de tecnologia química é empregado para sua confecção.

4.2.2. Placa projetada

Para o sistema funcionar corretamente, houve a necessidade de confeccionar uma placa de circuito impresso. Para seu projeto utilizou-se o software próprio para desenvolver placas, o *CAD Protel* (Altium, EUA) que, a partir do esquemático do circuito (Anexo D), indica as ligações necessárias a serem feitas com os componentes corretamente posicionados na placa. O CAD possibilita que a interconexão entre os componentes seja feita manualmente ou automaticamente. A Figura 4.11 mostra o resultado final obtido, bem como o arquivo de saída para a impressão da placa.

(a)



(b)

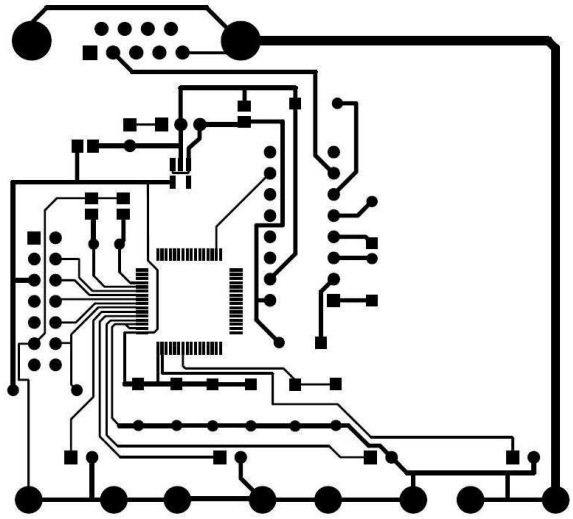


Figura 4.11 - (a) Placa projetada. (b) Arquivo de saída para impressão da placa.

4.2.3. **Confecção da placa**

Para a confecção da placa projetada, utilizou-se um método não industrial. Consiste basicamente em transferir o desenho do circuito para a placa de fenolite e corroê-la com percloreto de ferro (cloreto férrico) ou ácido nítrico diluído a 50%.

Imprimiu-se o leiaute da placa em papel adesivo (empregado para encapar livros e cadernos) utilizando uma impressora a *laser*. Em seguida, com um ferro de passar roupas, a uma temperatura mediana, transferiu-se o desenho do papel para a placa de fenolite. Com o calor do ferro, a tinta se solta do papel e prende na placa. Feito isso, utiliza-se água corrente para ajudar a desprender o papel da placa sem danificar as trilhas obtidas.

Mergulha-se a placa no percloreto de ferro (cloreto férrico) ou ácido nítrico diluído a 50% para corroer o cobre desnecessário. Neste processo, o ácido irá corroer mais rápido as áreas de cobre descobertas, onde após um tempo de exposição ao ácido, obtém-se o leiaute definido pelo desenho do circuito. Finalmente, com uma palha de aço, retirou-se a tinta sobre placa.

Capítulo 5: VALIDAÇÃO DO SISTEMA

Sabe-se que as senóides são autofunções, isto é, ao se inserir um sinal senoidal em um sistema linear, obtém-se, na saída, uma senóide de mesma freqüência, mas com fase e/ou amplitude diferentes. Sabendo-se disso, pode-se avaliar a resposta em freqüência do sistema de aquisição e verificar se a amostragem está feita conforme as taxas previamente definidas.

Por não ter sido possível confeccionar a placa de circuito impresso do sistema de aquisição e, ainda, devido ao pouco tempo disponível, não foi possível validar o sistema por meio de medições em músculos.

Portanto, a metodologia utilizada na validação tanto do sistema de aquisição quanto do software implementado foi a realização de experimentos em bancada. Para isso, inseriu-se um sinal conhecido em cada canal do conversor A/D do MSP, o qual foi capturado por meio do software implementado. Utilizando-se o *MatLab*, pôde-se analisar se o sistema perdia dados ou inseria distorções.

Plotando o sinal em função do tempo, pode-se saber se há algum valor de tensão mal convertido ou se o conversor A/D implementado não converte os valores conforme a equação (4.2), já que é possível avaliar a amplitude do sinal. Similarmente, através da Transformada de Fourier, pode-se avaliar a resposta em freqüência do conversor, verificando se a amostragem está correta e dentro da faixa estipulada para cada canal.

5.1. PROCEDIMENTOS

O procedimento adotado, então, foi a inserção de sinais senoidais em quatro freqüências distintas e com amplitude escolhida de maneira que cobrisse toda faixa de aquisição do conversor A/D. A Tabela 5.1 mostra os parâmetros do sinal escolhido.

Tabela 5.1 – Características da senóide.

			Senóides			
			V_{PP} = 2,5 V		DC = 1,25 V	
Canal	**Designação**	**Taxa de Amostragem**	**Freqüência [Hz]**			
0	Força	50 Hz	0,5	5	10	20
1	EMG Agonista	1000 Hz	50	100	250	450
2	EMG Antagonista	1000 Hz	50	100	250	450
3	*Trigger*	50 Hz	0,5	5	10	20

Montado o circuito na placa de matriz de contatos (*protoboard*) e configurado o gerador de funções para gerar os sinais conforme a tabela supracitada, executou-se o software implementado, configurando-o da seguinte maneira:

i) Configuração Serial
  a) Baud Rate: 115200 bps;
  b) Bits: 8;
  c) Paridade: None;
  d) Bits de Parada: 2

ii) Célula de Carga
  a) Peso: 25 kg
  b) Fator Força: 0,012

iii) Força Máxima: 25 kg;

iv) Tempo de Aquisição: 20 segundos.

Os dados salvos pelo programa foram processados e analisados utilizando-se o *MatLab*. O código implementado (Anexo C) tem por função abrir os arquivos, plotar os dados em uma escala conveniente e calcular a Transformada de Fourier para cada sinal, plotando-a. Assim, pode-se verificar se a senóide foi corretamente adquirida e se ela está na freqüência correta. Verifica-se, então, se houve alguma perda ou distorção durante o processo de aquisição.

Analogamente, repetiu-se o procedimento acima utilizando como sinal uma onda quadrada em diversas freqüências e com amplitude pico a pico dentro da faixa do conversor A/D. Este teste serve para avaliar a aquisição de sinais com diversas componentes de freqüência.

Tabela 5.2 – Características da onda quadrada.

			Onda Quadrada			
			V_{PP} = 2,5 V		DC = 1,25 V	
Canal	**Designação**	**Taxa de Amostragem**	**Freqüência [Hz]**			
0	Força	50 Hz	0,5	1	2	5
1	EMG Agonista	1000 Hz	10	20	50	60
2	EMG Antagonista	1000 Hz	10	20	50	60
3	*Trigger*	50 Hz	0,5	1	2	5

5.2. RESULTADOS OBTIDOS

Feito isso, obtiveram-se os seguintes resultados:

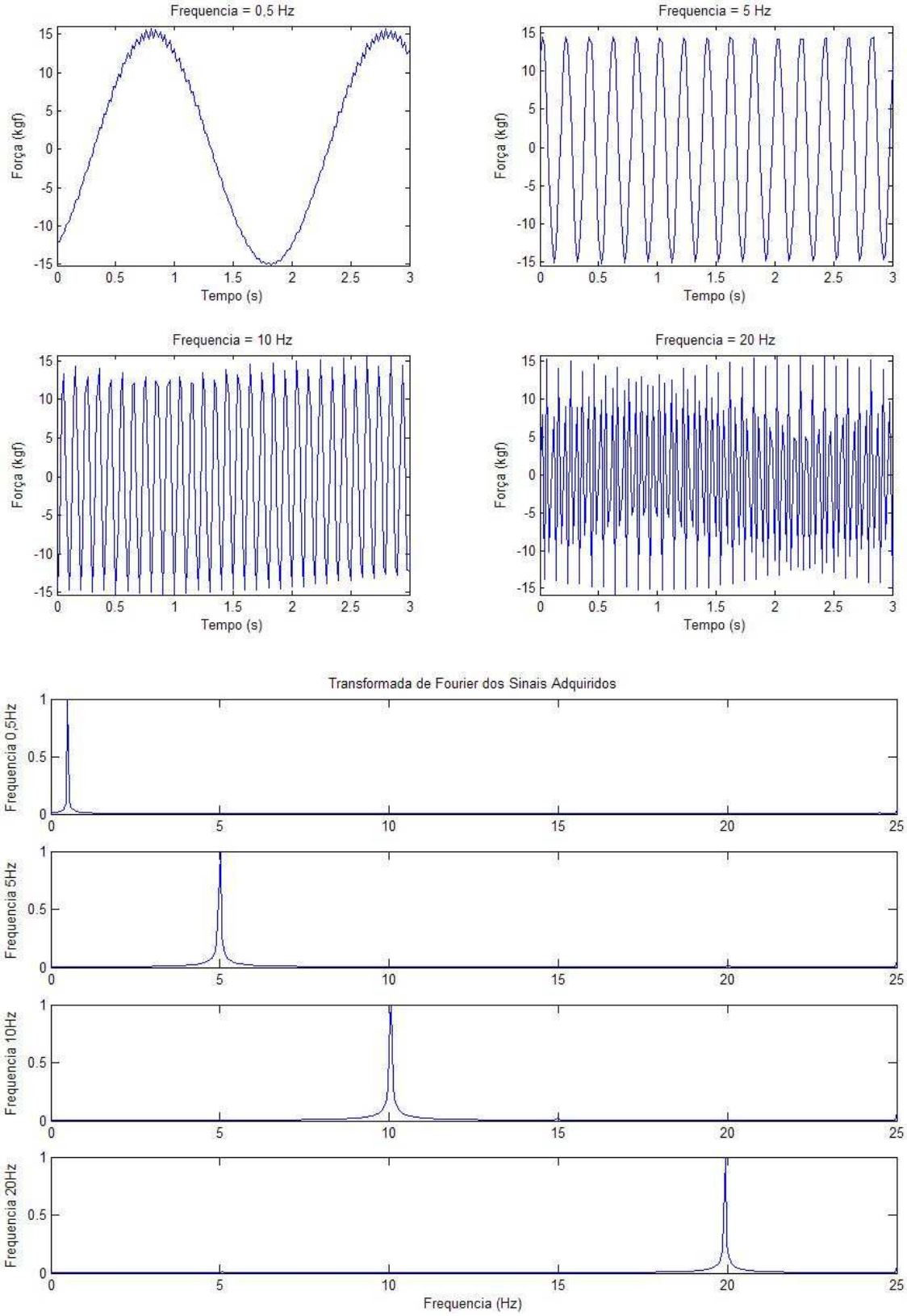


Figura 5.1 – Canal 0 (força) para sinais senoidais na entrada. Obseva-se que não há perda de dados durante a aquisição, pois as senóides não apresentam distorções ao longo do tempo. Adicionalmente, verifica-se, através da FFT do sinal, que ele foi corretamente amostrado, pois as componentes freqüenciais não sofreram desvio.

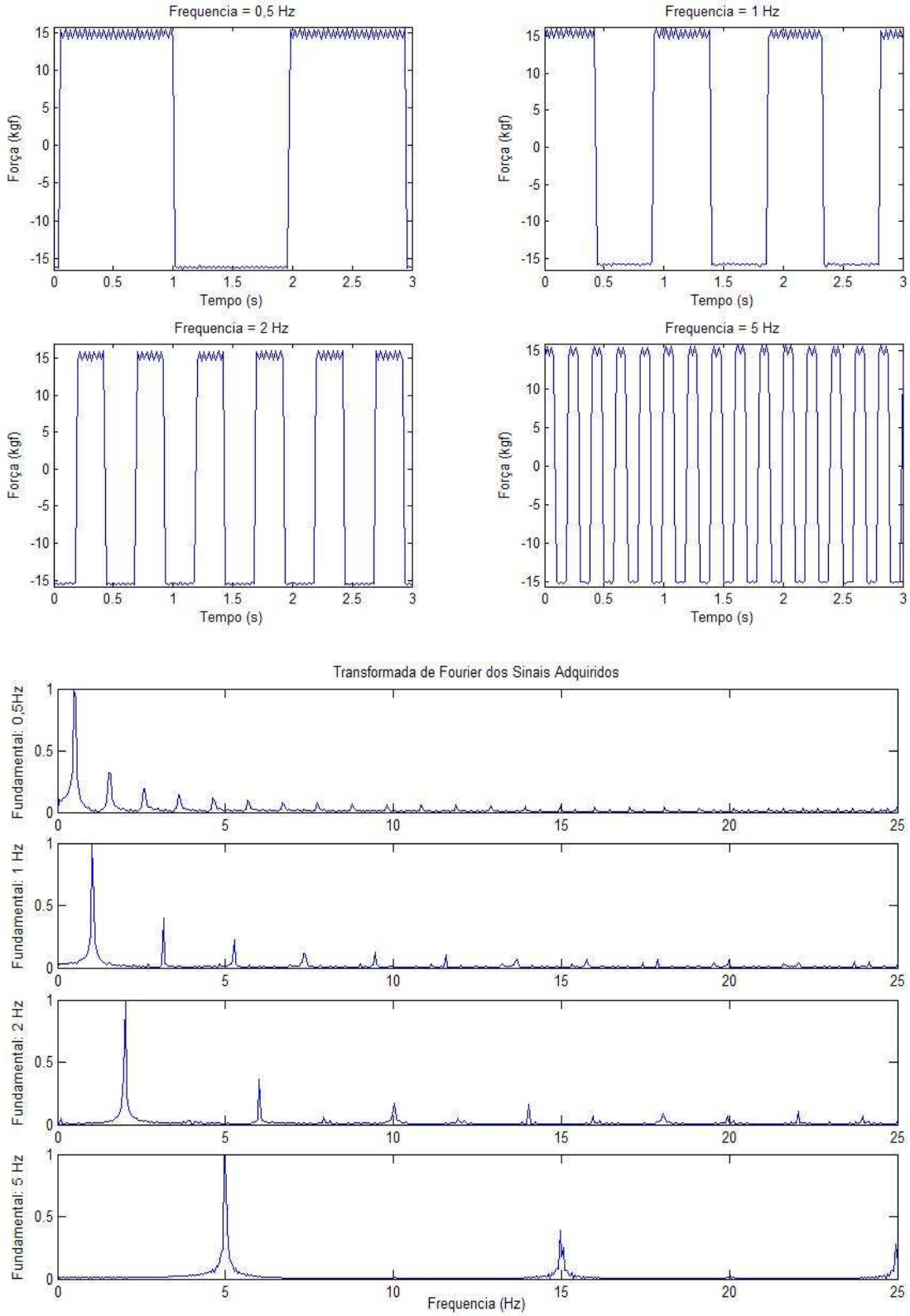


Figura 5.2 – Canal 0 (força) para onda quadrada na entrada. Obseva-se que não há perda de dados durante a aquisição, pois se verifica que o formato das ondas não sofreu distorções ao longo do tempo. Adicionalmente, através da FFT do sinal, observa-se que ele foi corretamente amostrado. O espectro obtido caracteriza uma onda quadrada por possuir as componentes impares, múltiplos da freqüência fundamental.

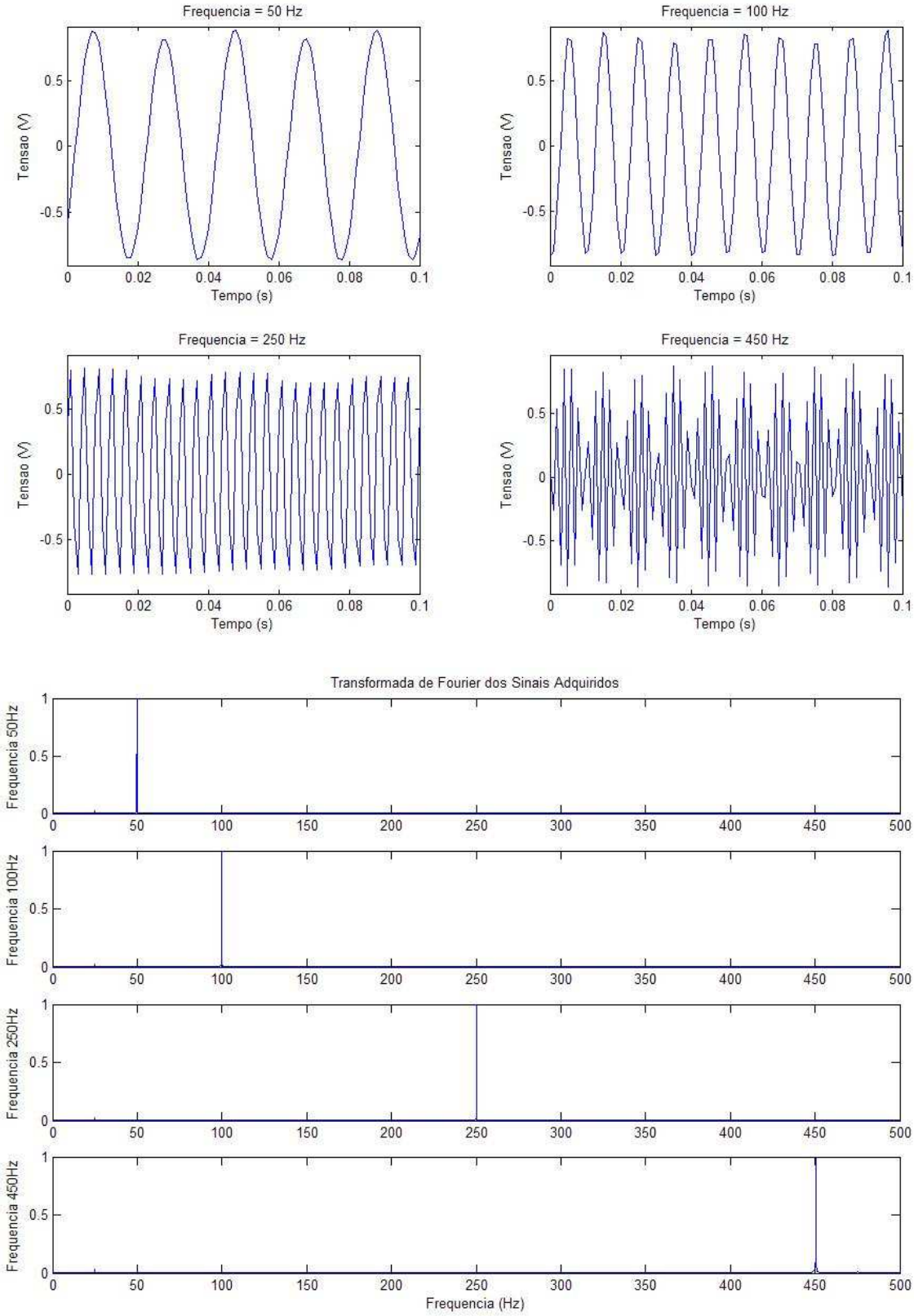


Figura 5.3 – Canal 1 (EMG do agonista) para sinais senoidais na entrada. Obseva-se que não há perda de dados durante a aquisição, pois as senóides não apresentam distorções ao longo do tempo. Adicionalmente, verifica-se, através da FFT do sinal, que ele foi corretamente amostrado, pois as componentes freqüenciais não sofreram desvio.

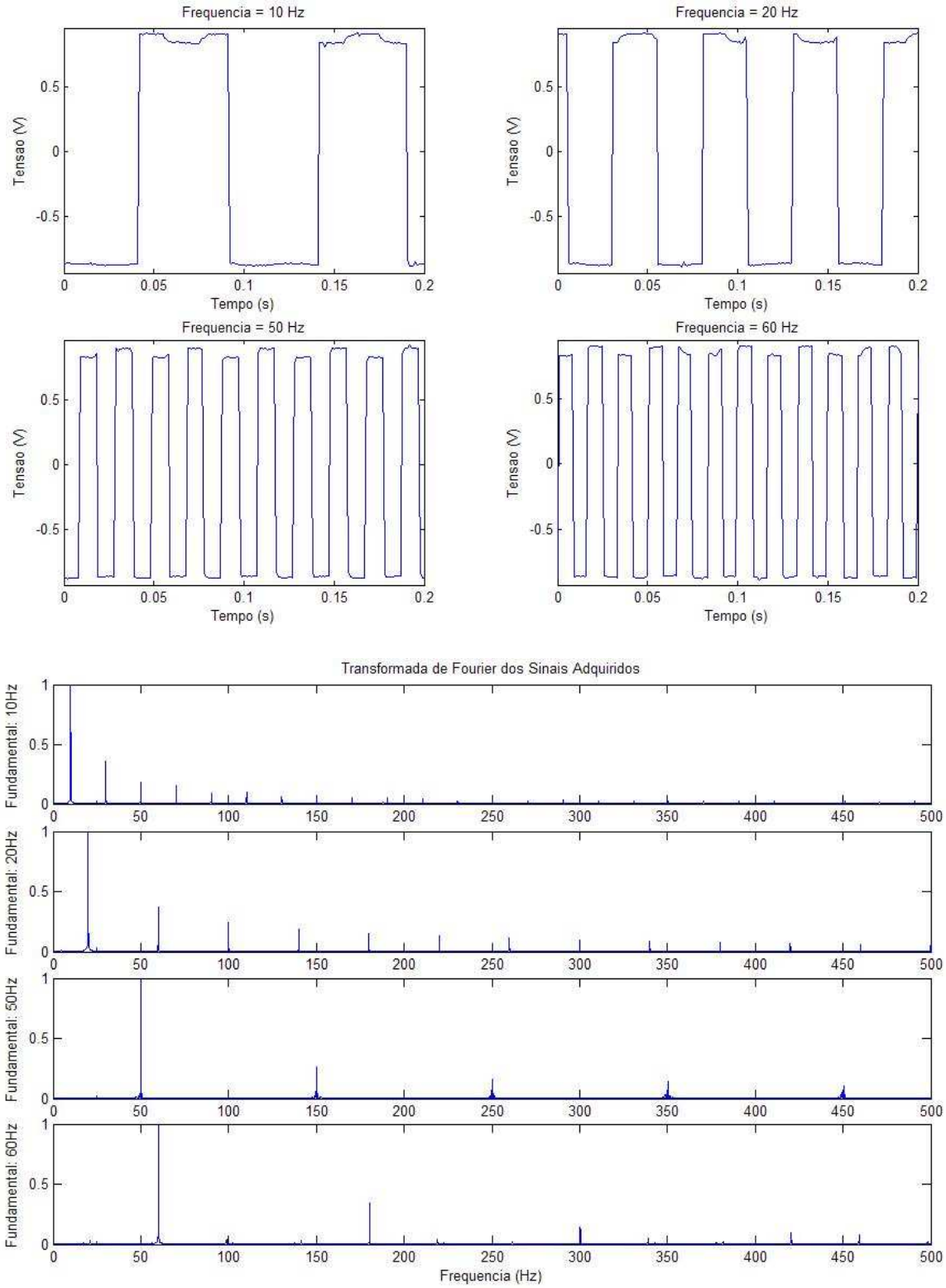


Figura 5.4 – Canal 1 (EMG do agonista) para onda quadrada na entrada. Obseva-se que não há perda de dados durante a aquisição, pois se verifica que o formato das ondas não sofreu distorções ao longo do tempo. Adicionalmente, através da FFT do sinal, observa-se que ele foi corretamente amostrado. O espectro obtido caracteriza uma onda quadrada por possuir as componentes impares, múltiplos da freqüência fundamental.

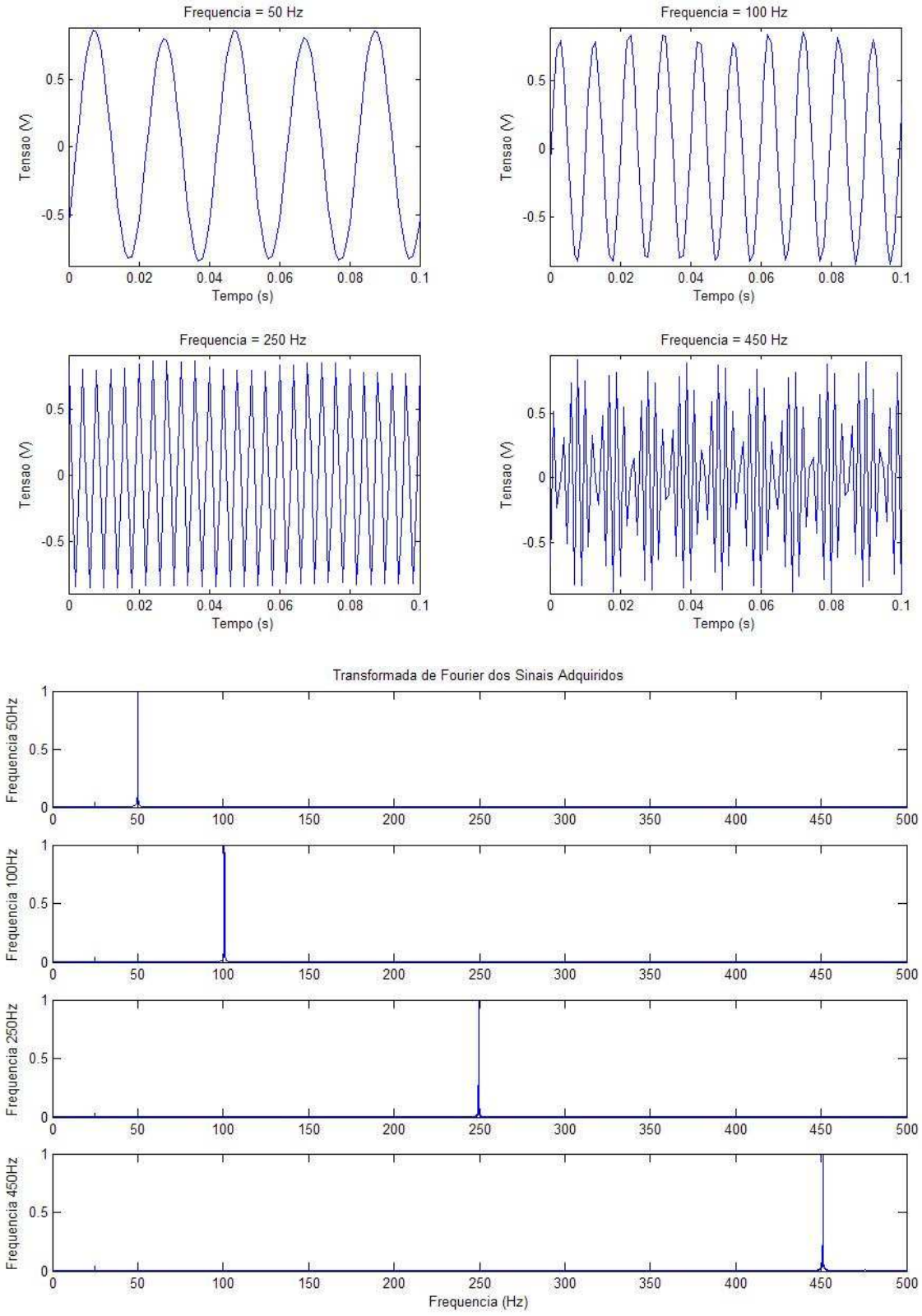


Figura 5.5 – Canal 2 (EMG do antagonista) para sinais senoidais na entrada. Obseva-se que não há perda de dados durante a aquisição, pois as senóides não apresentam distorções ao longo do tempo. Adicionalmente, verifica-se, através da FFT do sinal, que ele foi corretamente amostrado, pois as componentes freqüenciais não sofreram desvio.

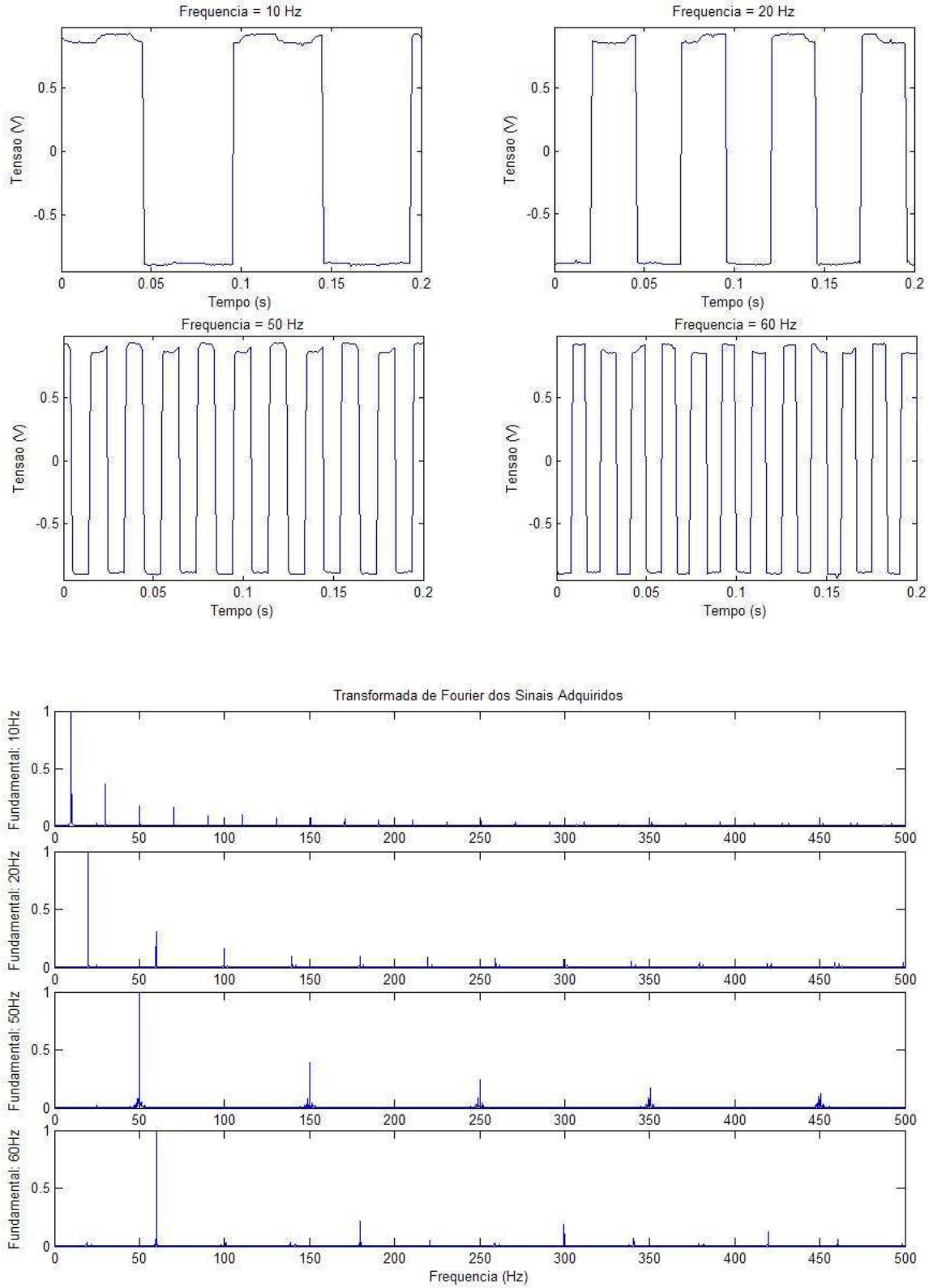


Figura 5.6 – Canal 2 (EMG do antagonista) para onda quadrada na entrada. Obseva-se que não há perda de dados durante a aquisição, pois se verifica que o formato das ondas não sofreu distorções ao longo do tempo. Adicionalmente, através da FFT do sinal, observa-se que ele foi corretamente amostrado. O espectro obtido caracteriza uma onda quadrada por possuir as componentes impares, múltiplos da freqüência fundamental.

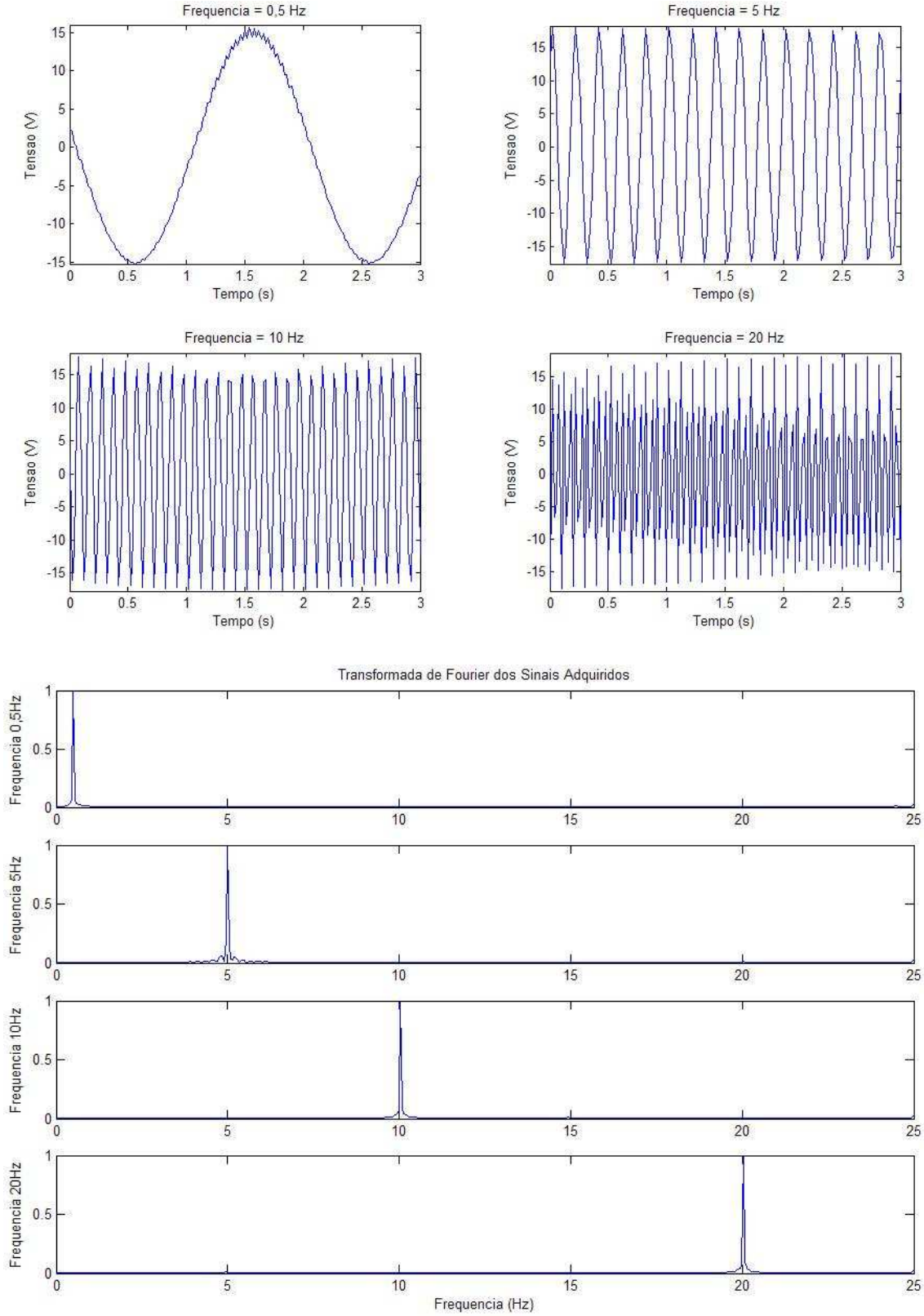


Figura 5.7 – Canal 3 (*trigger*) para sinais senoidais na entrada. Obseva-se que não há perda de dados durante a aquisição, pois as senóides não apresentam distorções ao longo do tempo. Adicionalmente, verifica-se, através da FFT do sinal, que ele foi corretamente amostrado, pois as componentes freqüenciais não sofreram desvio.

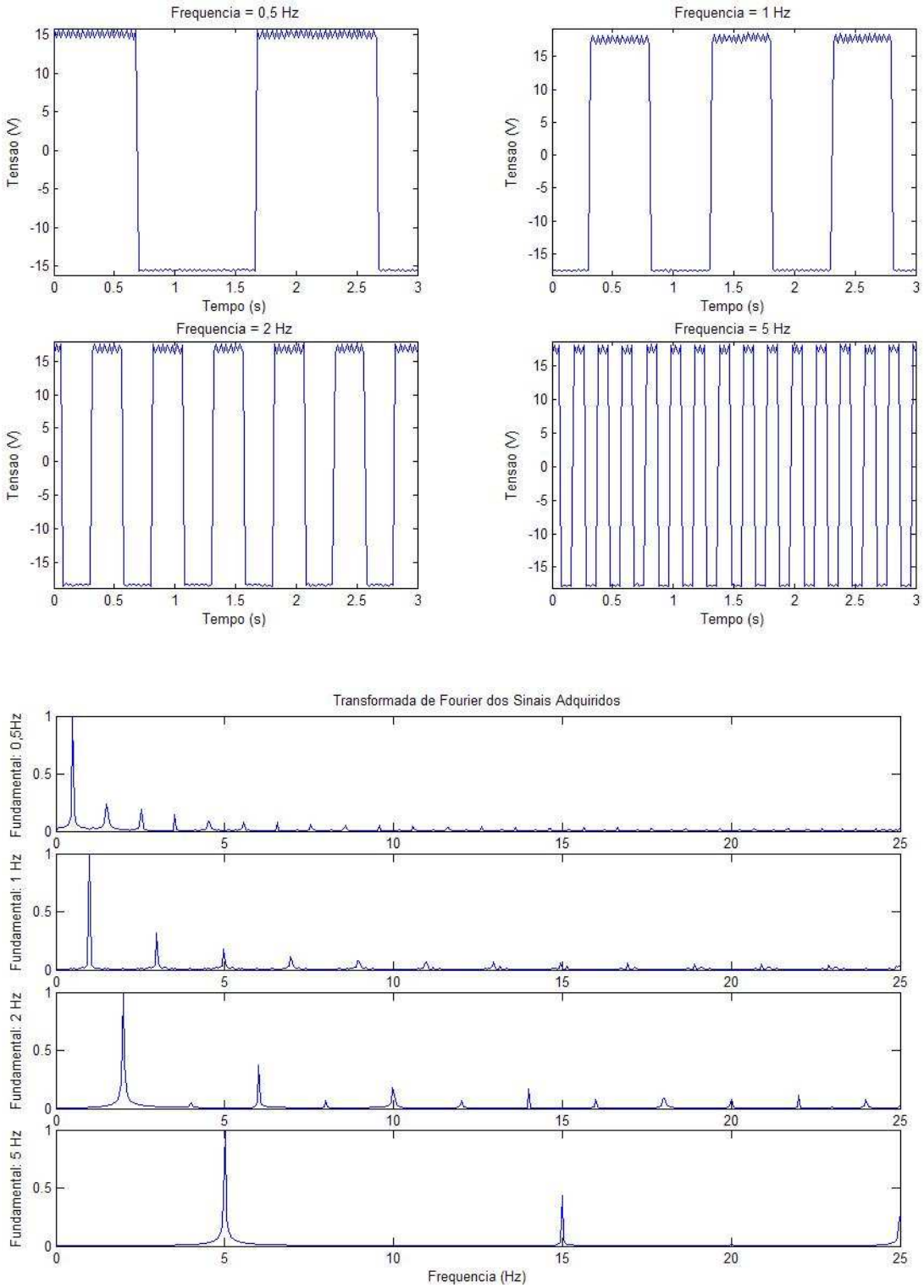


Figura 5.8 – Canal 3 (*trigger*) para onda quadrada na entrada. Obseva-se que não há perda de dados durante a aquisição, pois se verifica que o formato das ondas não sofreu distorções ao longo do tempo. Adicionalmente, através da FFT do sinal, observa-se que ele foi corretamente amostrado. O espectro obtido caracteriza uma onda quadrada por possuir as componentes impares, múltiplos da freqüência fundamental.

5.3. DISCUSSÃO

Analisando as Figuras 5.1 a 5.8, verifica-se que a conversão está sendo feita corretamente e que nenhum dado foi perdido. Por meio da análise dos espectros, verifica-se que a taxa de amostragem, programada no MSP para cada canal, está de acordo com o requerido, pois, observando-se as componentes, percebe-se não ter havido desvio na freqüência. Se a taxa de amostragem estivesse errada, verificar-se-ia que as compentes espectrais estariam deslocadas na freqüência. Vale ressaltar que as freqüências dos sinais foram escolhidas de modo a atender o princípio de Nyquist, evitando o *aliasing* (dobramento espectral), que poderia distorcer os resultados. Contudo, pelo fato de a onda quadrada possuir um espectro bastante espalhado, não houve como evitar que as componentes de alta freqüência distorcessem o espectro resultante. Por isso, o experimento foi realizado utilizando sinais com freqüência bem inferior ao limite de metade da taxa de amostragem.

Acrescente-se, ainda, que a conversão feita pelo conversor A/D respeitou a equação (4.2), visto que o formato das ondas não apresentou grandes diferenças. Verificou-se, ainda, que o protocolo de comunicação explicado em capítulos anteriores funciona corretamente, conforme ilustram os resultados obtidos.

Capítulo 6: DIFICULDADES NO PROJETO E PROPOSTAS PARA TRABALHOS FUTUROS

O objetivo deste capítulo é mostrar as dificuldades e soluções obtidas ao longo do trabalho, servindo como um material de consulta para futuros projetos. Além disso, estão sugeridos trabalhos que dêem continuidade ao que foi desenvolvido. Espera-se com isso incentivar novos estudos na área de instrumentação biomédica e auxiliar outros estudantes que se envolvam em projetos semelhantes.

6.1. DIFICULDADES NO PROJETO

A primeira dificuldade encontrada foi aprender a utilizar o MSP430. Pelo fato de não se ter experiência prévia com esse componente, houve a necessidade de um estudo mais atencioso sobre este microcontrolador. A pequena quantidade de material e bibliografia fez com que fosse gasto um tempo maior para aprender a trabalhar com ele. Para isso, foram feitas várias práticas experimentais, onde se desenvolveram códigos para a utilização da comunicação serial, rotinas de interrupção, utilização do conversor analógico-digital. Contudo, esta etapa não ofereceu grandes problemas devido à experiência no uso de outros microcontroladores, tal como o 8051 (Intel, EUA), bastando aprender a estrutura interna do MSP, bem como as instruções disponíveis para o seu uso.

Para a implementação do software em C++, houve dificuldade para se encontrar e utilizar bibliotecas que atendessem às necessidades. Em várias ocasiões, não havia documentação explicando a maneira de se utilizá-la, e a solução foi procurar outra biblioteca de códigos, gastar tempo tentando aprender a utilizá-la ou desenvolver o próprio código. As bibliotecas permitem um melhor resultado e também economia de tempo, pois não há a necessidade de implementar o código, às vezes complicado, para realizar certa tarefa.

Para realizar a comunicação serial, utilizou-se um pacote que disponibiliza funções que acessam a porta e enviam/recebem dados. Para plotar os dados, houve a necessidade de adaptar uma biblioteca, pois era necessário plotar duas curvas, no caso dos valores RMS dos sinais de EMG dos músculos agonista e antagonista, e plotar uma curva mais os limites, no caso da meta de força. A biblioteca permitia apenas plotar uma curva por painel. Um conhecimento mais aprofundado em linguagem C++ facilita bastante a solução desse tipo de problema, pois, além do programador conhecer funções que o auxiliam, ele já está acostumado com este tipo de situação.

A maior dificuldade enfrentada foi a implementação do hardware. Inicialmente, a placa de circuito impresso seria fabricada no Laboratório de Linhas e Fitas do Departamento de Engenharia Elérica, que possui uma máquina (fresadora) específica para isso. Contudo, essa máquina não possui resolução suficiente para poder fazer as diminutas trilhas do MSP430. Apesar de se ter confeccionado a peça, o resultado foi insatisfatório, pois várias trilhas ficaram interrompidas ou foram arrancadas.

Tentou-se, então, um método quase artesanal (papel adesivo), conforme explicado no capítulo 4. Esse método não permitiu resultados satisfatórios, pois, ocorriam principalmente dois tipos de problemas: ou parte do desenho não passava para a placa, pelo fato de o ferro de passar roupas encontrar-se em uma temperatura baixa e durante um intervalo insuficiente; ou algumas trilhas muito próximas se uniam, formando um curto-circuito, pelo fato de o ferro estar em uma temperatura excessivamente elevada. Novamente, o problema ocorreu em torno das trilhas do MSP430, e mesmo após várias tentativas não foi possível obter uma placa satisfatória.

Há, na internet, outros métodos de confecção, a citar: existe um papel próprio para passar o desenho impresso em impressora *laser* para a placa, também utilizando um ferro quente; outro método utiliza uma resina fotográfica que, quando exposta à luz, torna-se mais rígida, dificultando a corrosão do cobre pelo percloreto de ferro.

Sem dúvida, a melhor opção seria fabricar a placa em uma empresa especializada em confecção de placas de circuito impresso. Isso, contudo, resultaria em um elevado custo financeiro, onerando todo o processo.

Ainda na confecção da placa, outro problema é a soldagem de componentes pequenos como o MSP e outros dispositivos do tipo SMD na placa, o que necessita de uma maior prática com equipamentos de solda, além de materiais específicos. Por serem componentes pequenos, a chance de ocasionar curto-circuito nas trilhas é grande, comprometendo, assim, todo o processo.

6.2. PROPOSTA PARA TRABALHOS FUTUROS

Sugerem-se, aqui, alguns trabalhos que podem ser implementados, possibilitando uma melhor avaliação de um atleta, bem como determinação de um possível diagnóstico.

6.2.1. Alterações no software

Inicialmente, poderia ser adicionado ao programa um módulo que calcule e plote a freqüência mediana dos músculos agonista e antagonista para acompanhar se o atleta está entrando em fadiga. Para o seu cálculo, o usuário determinaria o tamanho e o tipo da janela (por exemplo, Hanning, Bartlet, Retangular etc.), bem como o passo em que ela desliza sobre o sinal.

Analogamente, propõe-se a inclusão de um módulo para processamentos específicos para os sinais de EMG, tais como: branqueamento, análises estatísticas, filtros digitais para diminuir ruídos externos, utilização de transformadas do tipo *wavelets*, etc.

Para propiciar uma avaliação dinâmica e periódica dos músculos agonista e antagonistas, poder-se-ia utilizar, em vez de meta de força constante, uma variação senoidal, onde através do *biofeedback* o atleta tem que variar o nível de força segundo uma senóide de baixa amplitude.

Seria interessante também elaborar na parte de cadastro, um histórico das aquisições feita por cada atleta, de maneira que se possa acompanhar sua evolução. Automaticamente, o software montaria um resumo de todo o trabalho feito em cada pessoa. Poderia, ainda, implementar a parte de impressão dos dados, permitindo seu arquivamento ou que a pessoa levasse consigo os resultados de suas atividades.

6.2.2. Hardware

Para o hardware, poderia ser incluído um circuito eliminador de pilha, permitindo que o sistema fosse alimentado por corrente alternada da rede 110/220 V. Um projeto mais elaborado poderia substituir a comunicação via interface RS-232 pela comunicação via interface USB (*Universal Serial Bus*). Com isso, obter-se-ia uma maior taxa na transmissão dos dados, permitindo um aumento no número de canais de EMG.

Sugere-se, também, o projeto de um sistema integrado onde, na própria placa, estaria todo o hardware (os amplificadores de EMG, filtros analógicos etc.), bem como a confecção da placa de circuito impresso em empresas especializadas.

Capítulo 7: CONCLUSÃO

O presente trabalho consistiu na elaboração de um sistema para a aquisição, o registro, o processamento e o monitoramento em tempo real de sinais de força e eletromiográficos. Optou-se pela implementação do sistema a partir do microcontrolador MSP430 e a plataforma *C++ Builder*, o que garantiu a funcionalidade do sistema e minimizou seus custos financeiros de implementação.

O uso da plataforma *C++ Builder*, por sua vez, confere maior independência e portabilidade ao sistema, já que não há a necessidade de instalação daquele no computador utilizado. Outra vantagem da utilização do ambiente *C++ Builder* em detrimento ao *Labview*, anteriormente empregado, é que aquele requer menos capacidade de hardware computacional, podendo ser executado com mais facilidade pelo computador. Isso permite que se atinjam taxas de aquisição de dados superiores àquelas possíveis com o *Labview*.

Necessitou-se, para isso, aprender a fundo o funcionamento e os princípios do MSP430 e do ambiente *C++ Builder*, de modo a implementar corretamente todas as funções propostas para o sistema.

A previsão inicial era a de validar o sistema realizando medições em músculos de voluntários. Não foi possível, contudo, realizar a validação dessa forma, já que não se conseguiu construir a placa de circuito impresso pretendida, tendo sido os componentes eletrônicos montados em uma placa do tipo matriz de contatos. Optou-se, então, pela validação em bancada. Para isso, inseriam-se sinais conhecidos em cada canal do conversor analógico-digital do microcontrolador, capturando-os por meio do software implementado. Utilizando-se o software *MatLab*, comprovou-se que o sistema não perde dados ou insere distorções no sinal capturado. Pelo fato de o sistema adquirir e processar os dados corretamente, a etapa de validação foi bem sucedida.

Os objetivos centrais do projeto foram, portanto, alcançados, já que se provou ser possível e viável a implementação do sistema a partir do MSP430 e do *C++ Builder*. Como a topologia do hardware implementado em placa de matriz de contato operou corretamente, bastará, no futuro, montar os componentes eletrônicos em uma placa de circuito impresso. Além disso, o trabalho permitiu significativo aprendizado sobre as propriedades do microcontrolador e do ambiente de programação utilizados.

REFERÊNCIAS BIBLIOGRÁFICAS

Livros

[1] SEDRA, A. S.; SMITH, K. C.; *Microeletrônica*, 4ª edição, Makron Books, São Paulo, 2000.

[2] BENTLEY, J. P.; *Principles of Measurement Systems*, 2nd edition, Longman Scientific & Technical, New York, 1988.

[3] NAGY, C.; *Embedded Systems Design using the TI MSP430 Series*, 1ª edição, Newnes, 2003.

[4] BASMANJIAN, J. V.; DE LUCA, C. J.; *Muscles Alive*. Baltimore: Williams & Wilkins, 1985.

[5] LATHI, B. P.; *Modern Digital and Analog Communication Systems*, 3rd edition, Oxford University Press, 1998.

[6] ALVES, W. P.; *C++ Builder 6: Desenvolva Aplicações para Windows*, 1ª edição, Editora Érica, 2002.

[7] MATEUS, C. A.; *C++ Builder 5: Guia Prático*, Editora Érica, 1ª edição, 2000.

[8] ZELENOVSKY, R.; MENDONÇA, A.; *PC: Um Guia Pratico de Hardware e Intefaceamento*, MZ Editora, 3ª edição.

[9] BARROS, FISCHER & ASSOCIADOS; *Anatomia (Série de Medicina 1)*, BF&A, 4ª edição, 2004.

[10] CRAM, J. R.; KASMAN, G. S.; HOLTZ, J.; *Introduction to Surface Eletromyography*. Gaithersburg: Aspen Publishers, 1998.

[11] KUMAR, S.; MITAL, A.; *Electromiography in Eergonomics*. Taylor & Francis, 1996.

[12] CORREIA, P. P.; SANTOS, P. M. & VELOSO, A; *Eletromiografia. Fundamentação Fisiológica, métodos de recolha e processamento. Aplicações cinesiológicas.* FMH, 1993.

Artigos Científicos

[13] VENEZIANO, W. H.; PENA, A. G.; ROCHA, A. F.; GONÇALVES, C. A.; CARMO, J. C.; NASCIMENTO, F. A. O.; ANDRADE, M. M.; *Influência do Ambiente Subaquático na Amplitude do Sinal Eletromiográfico de Superfície do Grupo Tênar*, CLAEB, João Pessoa, 2004.

[14] RODRIGUEZ-AÑEZ, C. R.; *A Eletromiografia na Análise da Postura*, UFSC, Dez/2000.

[15] LUCA, C. J.; *Surface Electromyography: Detection and Recording.* Disponível em <http://delsys.com>. Acesso em: 10 de abril de 2005.

[16] SENIAM Project: *European Recommendations for Surface Electromuography*. Freriks, B.; Hermens, H. J. Roessingh Reseach and Development, 1999.

Projetos Finais

[17] BARBOSA, B. O.; MELO, D. F. DE O.; *Sistema de Monitoramento Biomédico*. Trabalho de conclusão do curso de graduação em Engenharia de Redes de Comunicação, Universidade de Brasília, DF, 2003.

[18] ARAÚJO, V. V.; *Sistema de Captação e Tratamento de Sinais de Eletrocardiografia e Eletromiografia – Parte Integrante do Sistema de Telemetria para Monitoramento de Ciclistas*. Trabalho de conclusão do curso de graduação em Engenharia Elétrica, Universidade de Brasília, DF, 2003.

[19] OLIVEIRA, R. T. P.; BARROSO, V. B. R. B.; *Projeto e Implementação de um Sistema de Aquisição de Sinais de Força e Eletromiográficos com Software de Monitoração, Registro e Biofeedback*. Trabalho de conclusão do curso de graduação em Engenharia de Controle e Automação, Universidade de Brasília, DF, 2005.

Teses

[20] VENEZIANO, Wilson Henrique ; *Estudo do Comportamento do Sinal Eletromiográfico de Superfície em Atividades Subaquáticas.* Tese de doutorado em Engenharia Elétrica, Universidade de Brasília, DF. A ser defendida em abril de 2006.

Catálogos e Manuais

[21] *MSP430x13x, MSP430x14x Mixed Signal Microcontroller*, Texas Instruments, EUA, 2001.

[22] *MSP430x1xx Family User Guide*, Texas Instruments, EUA, 2003.

[23] INA121 Very High Accuracy Instrumentation Amplifier, Texas Instruments, EUA.

[24] MAX220-MAX249 +5 V – Powered, Multi-Channel RS-232 Drivers/Receivers, Maxim Integrated Products, EUA.

[25] LM3940 1 A Low Dropout Regulator for 5 V to 3.3 V Conversion, National Semiconductor, EUA.

[26] TL074 Low-Noise JFET-Input Operational Amplifiers, Texas Instruments, EUA, 2004.

Sítios na Internet de fabricantes

[27] TEXAS INSTRUMENTS. Disponível em: < http://www.ti.com >. Acesso em: 15 de março de 2005.

[28] NATIONAL INSTRUMENTS. Disponível em: < http://www.ni.com>. Acesso em: 2 de abril de 2005.

[29] DELSYS. Disponível em : < http://www.delsys.com>. Acesso em: 20 de março de 2005.

[30] KRATOS. Disponível em: < http://www.kratos.com.br>. Acesso em: 20 de maio de 2005.

Anexo A: CÓDIGO FONTE DO PROGRAMA DESENVOLVIDO

ANEXO A.1 – Unit_EMG.cpp (TELA PRINCIPAL)

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include "Unit_EMG.h"
#include "Unit_Serial.h"
#include "Unit_Forca_Max.h"
#include "Unit_Calibrar.h"
#include "Unit_Aparencia.h"
#include "Unit_Sobre.h"
#include "untCadastro.h"

//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma link "CSPIN"
#pragma link "GraphicPanel"
#pragma link "_GClass"
#pragma link "AbLED"
#pragma link "AbTank"
#pragma link "CPort"
#pragma link "VrEdit"
#pragma resource "*.dfm"
TfrmPrincipal *frmPrincipal;
//---------------------------------------------------------------------------
__fastcall TfrmPrincipal::TfrmPrincipal(TComponent* Owner)
  : TForm(Owner)
{
}
//---------------------------------------------------------------------------
AnsiString __fastcall TfrmPrincipal::MostraValor(double Valor)
{
  int intInteiro,intDecimal;
  double  dblDecimal;
  char  chrInteiro[10],chrDecimal[10],chrValor[20];

  intInteiro = (int) Valor;
  dblDecimal = (Valor - intInteiro)*10000;
  intDecimal = (int)dblDecimal;
  itoa(intInteiro,chrInteiro,10);
  itoa(intDecimal,chrDecimal,10);
  strcpy(chrValor,chrInteiro);
  strcat(chrValor,",");
  if(intDecimal < 10)
    strcat(chrValor,"000");
  else if(intDecimal < 100)
    strcat(chrValor,"00");
  else if(intDecimal < 1000)
    strcat(chrValor,"0");
  strcat(chrValor,chrDecimal);

  return((AnsiString)chrValor);
}
//---------------------------------------------------------------------------
void __fastcall TfrmPrincipal::Plota(int canal, unsigned int valor)
{
  float ValorRMS;
  int   ValorAux;

  if(((canal == 1)||(canal == 2))&&(flagCapturando))
  {
    ValorRMS=CalcRMS(canal,valor);
    LastRMS[canal-1] = ValorRMS * 4095 / 2.5;
  }

  switch (canal)
  {
    case 0: //Canal de Força
      ValorForca = (float)((double)valor * fatorForca);
      if(flagCapturando)
      {
```

```
        charTemp = FloatToStr(ValorForca);
        j = charTemp.Pos(",");
        if(j)
        {
          charTemp = charTemp.Delete(j,1);
          charTemp = charTemp.Insert(".",j);
        }
        memoForca->Lines->Add(charTemp);

        pnlForcaValor->Caption = MostraValor(ValorForca);
        gpanelForca->InserePonto(valor - metaForcaBin,0);
        gpanelForca->Plota();
        //Atualiza Leds
        if(ValorForca > LimForcaSup)
        {
          ledAlta->StatusInt = 1;
          ledBoa->StatusInt = 0;
          ledBaixa->StatusInt = 0;
        }
        else if(ValorForca < LimForcaInf)
        {
          ledBaixa->StatusInt = 1;
          ledBoa->StatusInt = 0;
          ledAlta->StatusInt = 0;
        }
        else
        {
          ledBoa->StatusInt = 1;
          ledBaixa->StatusInt = 0;
          ledAlta->StatusInt = 0;
        }
      }
      else if(flagForcaMax) //Captando a Forca Maxima
      {
        frmForcaMax->pnlForcaAtual->Caption = MostraValor(ValorForca);
        if(ValorForca > ForcaMax)
        {
          ForcaMax = ValorForca;
          frmForcaMax->pnlForcaMax->Caption = MostraValor(ForcaMax);
          frmForcaMax->tankForcaMax->SignalSettings->ValueTo = ForcaMax;
        }
        frmForcaMax->tankForcaMax->Value = ValorForca;
      }
      else if (flagCalibrar) //Calibrando Celula de Carga
      {
        if(valor > 0)
        {
          fatorForca = (StrToFloat(frmCalibrar->edtPeso->Text)/valor);
          frmCalibrar->pnlFatorCorr->Caption = MostraValor(fatorForca);
        }
      }
    break;

    case 1: //EMG Agonista
      if(flagCapturando)
      {
        charTemp = FloatToStr(ValorRMS);
        j = charTemp.Pos(",");
        if(j)
        {
          charTemp = charTemp.Delete(j,1);
          charTemp = charTemp.Insert(".",j);
        }
        memoRMSAgo->Lines->Add(charTemp);
        if(flagTank)  // Se flagTank = True --> Usando os Tanques
        {
          if(ValorRMS > tankAgonista->SignalSettings->ValueTo)
            tankAgonista->SignalSettings->ValueTo = ValorRMS;
          tankAgonista->Value = ValorRMS;
        }
        else  // flagTank = false  --> Usando o GPanel.
        {
          pnlRMSAgo->Caption = MostraValor(ValorRMS);
          ValorRMS = (int)(ValorRMS*4095 / 2.5);
          gpanelRMS->InserePonto((int)ValorRMS * gpanelRMS->Height/4095 - gpanelRMS->Height/2,
                                 (int)LastRMS[1] * gpanelRMS->Height/4095 - gpanelRMS-
>Height/2);
```

```
          gpanelRMS->Plota();
        }
      }
    break;

    case 2: //EMG Agntagonista
      if(flagCapturando)
      {
        charTemp = FloatToStr(ValorRMS);
        j = charTemp.Pos(",");
        if(j)
        {
          charTemp = charTemp.Delete(j,1);
          charTemp = charTemp.Insert(".",j);
        }
        memoRMSAnta->Lines->Add(charTemp);
        if(flagTank)  // Se flagTank = True --> Usando os Tanques
        {
         if(ValorRMS > tankAntagonista->SignalSettings->ValueTo)
            tankAntagonista->SignalSettings->ValueTo = ValorRMS;
          tankAntagonista->Value = ValorRMS;
        }
        else  // flagTank = false  --> Usando o GPanel.
        {
          pnlRMSAnta->Caption = MostraValor(ValorRMS);
          ValorRMS = (int)(ValorRMS*4095 / 2.5);
          gpanelRMS->InserePonto((int)LastRMS[0] * gpanelRMS->Height/4095 - gpanelRMS-
>Height/2,
                                 (int)ValorRMS * gpanelRMS->Height/4095 - gpanelRMS-
>Height/2);
          gpanelRMS->Plota();
        }
      }
    break;
  }
}
//---------------------------------------------------------------------------
float __fastcall TfrmPrincipal::CalcRMS(int canal, unsigned int valor)
{
  double ValorTensao, aux=0;

  ValorTensao =((double)valor) * 25 / 40950;
  if(canal == 1)  //Agonista
  {
    charTemp = FloatToStr(ValorTensao);
    j = charTemp.Pos(",");
    if(j)
    {
      charTemp = charTemp.Delete(j,1);
      charTemp = charTemp.Insert(".",j);
    }
    memoAgo->Lines->Add(charTemp);
    if(iAgo > (int)Winlen)
      iAgo = 0;
    Agonista[iAgo++] = ValorTensao;
    for(i=0;i<Winlen;i++)
      aux += pow(Agonista[i],2);
  }
  else if(canal == 2)  // Antagonista
  {
    charTemp = FloatToStr(ValorTensao);
    j = charTemp.Pos(",");
    if(j)
    {
      charTemp = charTemp.Delete(j,1);
      charTemp = charTemp.Insert(".",j);
    }
    memoAnta->Lines->Add(charTemp);
    if(iAntago > (int)Winlen)
      iAntago = 0;
    Antagonista[iAntago++] = ValorTensao;
    for(i=0;i<Winlen;i++)
      aux += pow(Antagonista[i],2);
  }
  return(sqrt(aux/Winlen));
}
//---------------------------------------------------------------------------
```

```
void __fastcall TfrmPrincipal::CPortRxChar(TObject *Sender, int Count)
{
   unsigned char buffer[128];
   unsigned int lenrx;
   static unsigned int canal;
   unsigned int canal_temp;
   static unsigned int amostra_temp;
   unsigned int amostra;
   static bool AmostraFlag=false;
   unsigned int k;

   if(!flag_plota)
     return;
   lenrx = CPort->Read(buffer,128);
   if(flagCapturando)
     fwrite(buffer,sizeof(char),lenrx,pt1);

   for(k=0;k<lenrx;k++)
   {
       canal_temp = (((unsigned int)(buffer[k]))>>4)&0x0F;
       if((canal_temp == 0 || canal_temp == 1 || canal_temp == 2 || canal_temp == 3)&&(!
AmostraFlag))
       {
          amostra_temp =  (((unsigned int)(buffer[k]))<<8)&0x0F00;
          AmostraFlag=true;
          canal=canal_temp;
       }
       else if(AmostraFlag)
       {
         amostra = amostra_temp + (((unsigned int)(buffer[k]))&0x00FF);
         Plota(canal,amostra);
         AmostraFlag=false;
       }
    }
}
//---------------------------------------------------------------------------
void __fastcall TfrmPrincipal::mnuFecharClick(TObject *Sender)
{
  frmPrincipal->Close();
  if(CPort->Connected)
  {
    CPort->ClearBuffer(true,true);
    CPort->Close();
  }
}
//---------------------------------------------------------------------------
void __fastcall TfrmPrincipal::btnAlteraClick(TObject *Sender)
{
  if(btnAltera->Caption == ">")
  {
    btnAltera->Caption = "<";
    flagTank = false;
  }
  else
  {
    btnAltera->Caption = ">";
    flagTank = true;
  }
  gpanelRMS->Visible ^=0x1;

}
//---------------------------------------------------------------------------
void __fastcall TfrmPrincipal::cfgSerialClick(TObject *Sender)
{
  frmSerial->ShowModal();
}
//---------------------------------------------------------------------------
void __fastcall TfrmPrincipal::btnForcaMaxClick(TObject *Sender)
{
   flagForcaMax = true;
   frmForcaMax->ShowModal();
}
//---------------------------------------------------------------------------
void __fastcall TfrmPrincipal::chkRampaClick(TObject *Sender)
{
  if(chkRampa->State == cbChecked)
  {
```

```
    lblVariacao->Enabled = true;
    spinVariacao->EditorEnabled = true;
    spinVariacao->Enabled = true;
  }
  else
  {
    lblVariacao->Enabled = false;
    spinVariacao->EditorEnabled = false;
    spinVariacao->Enabled = false;
  }
}
//---------------------------------------------------------------------------
void __fastcall TfrmPrincipal::FormCreate(TObject *Sender)
{
  AnsiString strPathCadastro;

  //Inicializa as Variáveis
  flagTank = true;            //Inicia Programa com o Tanque
  flag_plota = false;
  flagCadastro = false;

  chkRampa->State = cbUnchecked;
  lblVariacao->Enabled = false;
  spinVariacao->EditorEnabled = false;
  spinVariacao->Enabled = false;

  btnIniciar->Enabled =false;

  gpanelForca->DefineHandle();
  gpanelForca->AtualizaPar();
  gpanelForca->LigaBarra();
  gpanelForca->SegundaCurva(false);
  gpanelForca->LigaRange(true);
  gpanelForca->MudaCorLinha(3,clRed);

  gpanelRMS->DefineHandle();
  gpanelRMS->AtualizaPar();
  gpanelRMS->LigaBarra();
  gpanelRMS->SegundaCurva(true);
  gpanelRMS->LigaRange(false);

  pnlRMSAgo->Font->Color = (TColor) gpanelRMS->RecuperaCorLinha(1);
  pnlRMSAnta->Font->Color = (TColor) gpanelRMS->RecuperaCorLinha(2);
  pnlForcaValor->Font->Color = (TColor) gpanelForca->RecuperaCorLinha(1);
  pnlRMSAgo->Color = gpanelRMS->Color;
  pnlRMSAnta->Color = gpanelRMS->Color;
  pnlForcaValor->Color = gpanelForca->Color;

  cmbTempo->ItemIndex = 0;

  btnLimpar->Enabled = false;
  btnSalvar->Enabled = false;
  mnuSalvar->Enabled = false;


  //Cria o Diretorio ..\Dados
  strPathCadastro =GetCurrentDir() + "\\Dados";
  mkdir(strPathCadastro.c_str()); //Cria a Pasta Dados

  fileMusculo = strPathCadastro + "\\Musculos.txt";    // Abre arquivo Músculos
  if((pt1 = fopen(fileMusculo.c_str(),"r")) != NULL)
    boxMusculo->Items->LoadFromFile(fileMusculo);
  fclose(pt1);
}
//---------------------------------------------------------------------------
void __fastcall TfrmPrincipal::btnIniciarClick(TObject *Sender)
{
  if(btnIniciar->Caption == "Iniciar") //Iniciar Captura
  {
    switch(cmbTempo->ItemIndex)
    {
      case 0:
        TempoCaptura = -1;
      break;

      case 1:
        TempoCaptura = 59;
```

```
      break;

      case 2:
        TempoCaptura = 179;
      break;

      case 3:
        TempoCaptura = 299;
      break;
    }
    tmrTempo->Enabled = true;

    DeleteFile(caminho_final);
    // cria arquivo temporário
    if ((pt1 = fopen(caminho_final, "ab+")) == NULL)
    {
       ShowMessage("Erro na criação do arquivo!");
       btnSalvar->Enabled = false;
       mnuSalvar->Enabled = false;
    }
    else
      flag_aberto = true;

    flag_plota = true;
    flagCapturando = true;

    btnIniciar->Caption = "Parar";
    btnSair->Enabled = false;
    pnlMusculo->Enabled = false;
    btnForcaMax->Enabled = false;
    btnCelConfig->Enabled = false;
    btnLimpar->Enabled = false;
    cmbTempo->Enabled = false;
    pnlRMSCtrl->Enabled = false;
    spinAmostras->Enabled = false;

    //Inicializa Variáveis de armazenamento
    Winlen = (int)spinAmostras->Value;
    iAgo = iAntago = 0;
    for(i=0;i<Winlen;i++)
    {
      Agonista[i] = 0;
      Antagonista[i] = 0;
    }
    LastRMS[0] = LastRMS[1] = 0;

    //Inicializa Tempo
    min = seg = 0;

    //Dados da Força
    metaForca = ((float)spinMetaForca->Value)*ForcaMax/100;
    metaForcaBin = (unsigned int)(metaForca/fatorForca);
    LimForcaSup = ForcaMax*((float)(spinMetaForca->Value + spinTolerancia->Value)/100);
    LimForcaInf = ForcaMax*((float)(spinMetaForca->Value - spinTolerancia->Value)/100);
    LimSupBin = (unsigned int)(LimForcaSup / fatorForca);
    LimInfBin = (unsigned int)(LimForcaInf / fatorForca);
    gpanelForca->DefineLimites(LimSupBin - metaForcaBin,LimInfBin - metaForcaBin);

    //Dados do RMS
    tankAntagonista->SignalSettings->ValueTo = (int)spinRMSMax->Value;
    tankAgonista->SignalSettings->ValueTo = (int)spinRMSMax->Value;

    if(!CPort->Connected)
      CPort->Open();      //Habilita Porta Serial
    CPort->ClearBuffer(true,true);

    AtualizaBarra();
  }
  else if (btnIniciar->Caption == "Parar")  //Parar Captura
  {
    flag_plota = false;
    flagCapturando = false;

    btnLimpar->Enabled = true;
    btnIniciar->Caption = "Continuar";
    btnSair->Enabled = true;
    btnSalvar->Enabled =true;
```

```
    mnuSalvar->Enabled = true;

    AtualizaBarra();
    tmrTempo->Enabled = false;
  }
  else if (btnIniciar->Caption == "Pause")  //Pausar a revisao do sinal
  {
    btnIniciar->Caption = "Resume";
    btnLimpar->Enabled = true;
    tmrRevendo->Enabled = false;
    tmrTempo->Enabled = false;
    btnSair->Enabled = true;
  }
  else if((btnIniciar->Caption == "Resume")||(btnIniciar->Caption == "Rever")) // Continuar
Revendo
  {
    flag_plota = true;
    btnIniciar->Caption = "Pause";
    btnLimpar->Enabled = false;
    tmrRevendo->Enabled = true;
    tmrTempo->Enabled = true;
    btnSair->Enabled = false;
  }
  else if((btnIniciar->Caption == "Continuar")||(btnIniciar->Caption == "Repetir")) //
Continuar Capturando
  {
    if(btnIniciar->Caption == "Repetir")
    {
      if(Application->MessageBox("Deseja salvar os dados?", "Informação", MB_YESNO |
MB_ICONINFORMATION) == IDYES)
        btnSalvarClick(btnSalvar);

      switch(cmbTempo->ItemIndex)
      {
        case 0:
          TempoCaptura = -1;
        break;

        case 1:
          TempoCaptura = 59;
        break;

        case 2:
          TempoCaptura = 179;
        break;

        case 3:
          TempoCaptura = 299;
        break;
      }
    }
    flag_plota = true;
    flagCapturando = true;
    btnIniciar->Caption = "Parar";
    btnLimpar->Enabled = false;
    btnSair->Enabled = false;
    btnSalvar->Enabled = false;
    mnuSalvar->Enabled = false;
    tmrTempo->Enabled = true;
    if(CPort->Connected)
      CPort->ClearBuffer(true,true);
  }
}
//---------------------------------------------------------------------------
void __fastcall TfrmPrincipal::mnuAbrirSerialClick(TObject *Sender)
{
  if(!CPort->Connected)
    CPort->Open();
  AtualizaBarra();
}
//---------------------------------------------------------------------------
void __fastcall TfrmPrincipal::mnuFecharSerialClick(TObject *Sender)
{
  if(CPort->Connected)
    CPort->Close();
  AtualizaBarra();
}
```

```
//---------------------------------------------------------------------------
void __fastcall TfrmPrincipal::btnCelConfigClick(TObject *Sender)
{
  flagCalibrar = true;
  Winlen = spinAmostras->Value;
  frmCalibrar->ShowModal();
}
//---------------------------------------------------------------------------
void __fastcall TfrmPrincipal::btnAddMusculoClick(TObject *Sender)
{
  boxMusculo->Items->Add(edtAddMusculo->Text);
  boxMusculo->ItemIndex = boxMusculo->Items->IndexOf(edtAddMusculo->Text);
  boxMusculo->Items->SaveToFile(fileMusculo);
  edtAddMusculo->Text = "Músculo1";
}
//---------------------------------------------------------------------------
void __fastcall TfrmPrincipal::btnDelMusculoClick(TObject *Sender)
{
  int BIndex = boxMusculo->Items->IndexOf(boxMusculo->Text);

  if (BIndex > -1)
  {
    boxMusculo->Items->Delete(BIndex);
    boxMusculo->Items->SaveToFile(fileMusculo);
  }
  boxMusculo->ItemIndex =0;
}
//---------------------------------------------------------------------------
void __fastcall TfrmPrincipal::Cores1Click(TObject *Sender)
{
  frmAparencia->ShowModal();
}
//---------------------------------------------------------------------------
void __fastcall TfrmPrincipal::mnuAjudaClick(TObject *Sender)
{
  frmSobre->ShowModal();
}
//---------------------------------------------------------------------------
void __fastcall TfrmPrincipal::btnSalvarClick(TObject *Sender)
{
  FILE *origem, *destino;
  AnsiString CamOri,NomeUnSave,Temp;
  char caracter;
  flag_plota = false;
  long int len_amostras;
  unsigned char *buffer;
  unsigned int  Pos;

  if(flag_aberto)
  {
   fclose(pt1);
   flag_aberto = false;
  }

  if(flagCadastro)
    dlgSave->InitialDir = GetCurrentDir() + "\\Dados\\" + PessoaEscolhida + "\\"+ boxMusculo-
>Text;
  else
    dlgSave->InitialDir = GetCurrentDir() + "\\Dados";
  // Cria nome do arquivo segundo Modelo Dia-Mes-Ano [Hora h Min m Seg]
  NomeUnSave = DateToStr(Date())+" ["+TimeToStr(Time())+"]";
  Pos = NomeUnSave.Pos("/");
  NomeUnSave = NomeUnSave.Delete(Pos,1);
  NomeUnSave = NomeUnSave.Insert("-",Pos);
  Pos = NomeUnSave.Pos("/");
  NomeUnSave = NomeUnSave.Delete(Pos,1);
  NomeUnSave = NomeUnSave.Insert("-",Pos);
  Pos = NomeUnSave.Pos(":");
  NomeUnSave = NomeUnSave.Delete(Pos,1);
  NomeUnSave = NomeUnSave.Insert("h",Pos);
  Pos = NomeUnSave.Pos(":");
  NomeUnSave = NomeUnSave.Delete(Pos,1);
  NomeUnSave = NomeUnSave.Insert("m",Pos);
  dlgSave->FileName = NomeUnSave + ".emg";

  if(dlgSave->Execute())
  {
```

```
    if ((origem = fopen(caminho_final, "rb")) == NULL)
    {
      ShowMessage("Erro ao acessar arquivo.");
      btnSalvar->Enabled = false;
      mnuSalvar->Enabled = false;
      return;
    }

    destino = fopen(dlgSave->FileName.c_str(), "wb");

    len_amostras = filesize(origem);
    buffer = new unsigned char[len_amostras];
    fread(buffer,sizeof(char),len_amostras,origem);
    fwrite(buffer,sizeof(char),len_amostras,destino);

    fclose(origem);
    fclose(destino);

    //Salva Dados de Aquisição
    memoMusculo->Clear();
    memoMusculo->Lines->Add("\t\tDADOS DA AQUISIÇÃO");
    memoMusculo->Lines->Add("Data:\t" + DateToStr(Date())+"\t\t\tHora:\t"+TimeToStr(Time()));
    memoMusculo->Lines->Add("-------------------------------------------------------------");
    memoMusculo->Lines->Add("Músculo:\t\t\t\t" + boxMusculo->Text);
    memoMusculo->Lines->Add("Fator de Conversão Força/Tensão:\t" + FloatToStr(fatorForca));
    memoMusculo->Lines->Add("Força Máxima (kgf):\t\t\t" + FloatToStr(ForcaMax));
    memoMusculo->Lines->Add("Meta de Força (%):\t\t\t" + IntToStr((int)spinMetaForca->Value));
    memoMusculo->Lines->Add("Tolerância (%):\t\t\t\t" + IntToStr((int)spinTolerancia->Value));
    memoMusculo->Lines->Add("N° de Amostras Cálculo RMS:\t\t" + IntToStr((int)spinAmostras-
>Value));
    NomeUnSave = dlgSave->FileName;
    Pos = NomeUnSave.Pos(".");
    NomeUnSave = NomeUnSave.SubString(1,Pos) + "txt";
    memoMusculo->Lines->SaveToFile(NomeUnSave);

    // Salva valores da aquisição em arquivos do tipo txt
    // Arquivo de Força Nome do Arquivo: Dia-Mes-Ano [Hora h Min m Seg]-Forca.txt
    NomeUnSave = dlgSave->FileName;
    Pos = NomeUnSave.Pos(".");
    NomeUnSave = NomeUnSave.SubString(1,Pos - 1);
    Temp = NomeUnSave +"-Forca"+ ".txt";
    memoForca->Lines->SaveToFile(Temp);

    // Musculo Agonista: Dia-Mes-Ano [Hora h Min m Seg]-Ago[RMS].txt
    Temp = NomeUnSave +"-Ago"+ ".txt";
    memoAgo->Lines->SaveToFile(Temp);
    Temp = NomeUnSave +"-Ago[RMS]"+ ".txt";
    memoRMSAgo->Lines->SaveToFile(Temp);

    // Musculo Agonista: Dia-Mes-Ano [Hora h Min m Seg]-Anta[RMS].txt
    Temp = NomeUnSave +"-Anta"+ ".txt";
    memoAnta->Lines->SaveToFile(Temp);
    Temp = NomeUnSave +"-Anta[RMS]"+ ".txt";
    memoRMSAnta->Lines->SaveToFile(Temp);
  }

  memoForca->Clear();
  memoAgo->Clear();
  memoRMSAgo->Clear();
  memoAnta->Clear();
  memoRMSAnta->Clear();

  //if (buffer != NULL)
  //  delete[] buffer;

  if(!flag_aberto)
  {
   pt1 = fopen(caminho_final, "ab+");
   flag_aberto = true;
  }
  if(CPort->Connected)
      CPort->ClearBuffer(true,true);
}
//---------------------------------------------------------------------------
// Verifica tamanho de arquivo
long __fastcall TfrmPrincipal::filesize(FILE *stream)
{
```

```
        long curpos, length;
        curpos = ftell(stream);
        fseek(stream, 0L, SEEK_END);
        length = ftell(stream);
        fseek(stream, curpos, SEEK_SET);
        return length;
}
//---------------------------------------------------------------------------
bool __fastcall TfrmPrincipal::abrir_arquivo(void)
{
  AnsiString NomeArquivo,strTemp;
  FILE *fid;         // ponteiro para arquivo
  bool temp;
  int Pos;

  if(flagCadastro)
    dlgSave->InitialDir = GetCurrentDir() + "\\Dados\\" + PessoaEscolhida + "\\"+ boxMusculo-
>Text;
  else
    dlgSave->InitialDir = GetCurrentDir() + "\\Dados";

  // verifica se um caminho_install foi selecionado pelo usuário
  if (dlgOpen->Execute())
  {
      // abre o arquivo, lê colocando os bytes no vetor sinal e fecha
      if((fid = fopen(dlgOpen->FileName.c_str(),"rb")) != NULL)
      {
        nome_arquivo = dlgOpen->FileName.c_str();
        tamarq = filesize(fid);
        sinal = new unsigned char[tamarq];
        //flag_limpou_sinal = false;
        fread(sinal,sizeof(char),tamarq,fid);
        fclose(fid);
        temp = true;
      }

      // abre arquivo.txt que armazena os dados da captura
      NomeArquivo = dlgOpen->FileName;
      Pos = NomeArquivo.Pos(".");
      NomeArquivo = NomeArquivo.SubString(1,Pos) + "txt";
      if((fid = fopen(NomeArquivo.c_str(),"rb")) != NULL)
      {
        memoMusculo->Lines->LoadFromFile(NomeArquivo);

        strTemp = memoMusculo->Lines->Strings[3]; // Musculo
        Pos = strTemp.Pos("\t");
        strTemp = strTemp.SubString(Pos + 4,strTemp.Length() - (Pos + 3));

        strTemp = memoMusculo->Lines->Strings[4]; // Fator Forca
        Pos = strTemp.Pos("\t");
        strTemp = strTemp.SubString(Pos + 1,strTemp.Length() - Pos);
        boxMusculo->ItemIndex = boxMusculo->Items->IndexOf(strTemp.Trim());

        strTemp = memoMusculo->Lines->Strings[5]; // Forca Maxima
        Pos = strTemp.Pos("\t");
        strTemp = strTemp.SubString(Pos + 3,strTemp.Length() - (Pos + 2));
        ForcaMax = StrToFloat(strTemp.Trim());
        pnlMaxForca->Caption = MostraValor(ForcaMax);

        strTemp = memoMusculo->Lines->Strings[6]; // Meta Forca
        Pos = strTemp.Pos("\t");
        strTemp = strTemp.SubString(Pos + 3,strTemp.Length() - (Pos + 2));
        spinMetaForca->Value = StrToFloat(strTemp.Trim());
        metaForca = ((float)spinMetaForca->Value)*ForcaMax/100;

        strTemp = memoMusculo->Lines->Strings[7]; // Tolerancia
        Pos = strTemp.Pos("\t");
        strTemp = strTemp.SubString(Pos + 4,strTemp.Length() - (Pos + 3));
        spinTolerancia->Value = StrToFloat(strTemp.Trim());
        metaForcaBin = (unsigned int)(metaForca/fatorForca);
        LimForcaSup = ForcaMax*((float)(spinMetaForca->Value + spinTolerancia->Value)/100);
        LimForcaInf = ForcaMax*((float)(spinMetaForca->Value - spinTolerancia->Value)/100);
        LimSupBin = (unsigned int)(LimForcaSup / fatorForca);
        LimInfBin = (unsigned int)(LimForcaInf / fatorForca);
        gpanelForca->DefineLimites(LimSupBin - metaForcaBin,LimInfBin - metaForcaBin);

        strTemp = memoMusculo->Lines->Strings[8]; // WinLen
```

```
        Pos = strTemp.Pos("\t");
        strTemp = strTemp.SubString(Pos + 2,strTemp.Length() - (Pos + 1));
        spinAmostras->Value = StrToFloat(strTemp.Trim());
        Winlen = (int)spinAmostras->Value;
       }
   }
   else
    temp = false;
   return temp;
}
//---------------------------------------------------------------------------
void __fastcall TfrmPrincipal::tmrRevendoTimer(TObject *Sender)
{
   static unsigned int canal;
   unsigned int canal_temp;
   static unsigned int amostra_temp;
   unsigned int amostra;
   static bool AmostraFlag=false;

   for(i=0;i<100;i++)
   {
     canal_temp = (((unsigned int)(sinal[ult_amostra]))>>4)&0x0F;
     if((canal_temp == 0 || canal_temp == 1 || canal_temp == 2 || canal_temp == 3)&&(!
AmostraFlag))
     {
        amostra_temp =  (((unsigned int)(sinal[ult_amostra]))<<8)&0x0F00;
        AmostraFlag=true;
        canal=canal_temp;
     }
     else if(AmostraFlag)
     {
       amostra = amostra_temp + (((unsigned int)(sinal[ult_amostra]))&0x00FF);
       Plota(canal,amostra);
       AmostraFlag=false;
     }
     ult_amostra ++;
     if(ult_amostra >= tamarq)
     {
        tmrRevendo->Enabled = false;
        tmrTempo->Enabled = false;
        btnIniciar->Caption = "Rever";
        btnLimpar->Enabled = true;
        ult_amostra = min = seg = 0;
        return;
     }
   }
}
//---------------------------------------------------------------------------
void __fastcall TfrmPrincipal::mnuAbrirClick(TObject *Sender)
{
  //abre o sinal
  if(abrir_arquivo())
  {
    btnSalvar->Enabled = false;
    mnuSalvar->Enabled = false;
    ult_amostra = 0;
    btnIniciar->Enabled =true;
    btnIniciar->Caption = "Pause";
    btnLimpar->Enabled = false;
    btnIniciar->SetFocus();
    tmrRevendo->Enabled = true;
    tmrTempo->Enabled = true;
    TempoCaptura = -1;
    flagCapturando = true;

    btnSair->Enabled = false;
    pnlMusculo->Enabled = false;
    btnForcaMax->Enabled = false;
    btnCelConfig->Enabled = false;
    btnLimpar->Enabled = false;
    pnlRMSCtrl->Enabled = false;
    pnlSalvar->Enabled = false;
    spinAmostras->Enabled = false;

    //Inicializa Variáveis de armazenamento
    Winlen = (int)spinAmostras->Value;
    iAgo = iAntago = 0;
```

```
    for(i=0;i<Winlen;i++)
    {
      Agonista[i] = 0;
      Antagonista[i] = 0;
    }
    LastRMS[0] = LastRMS[1] = 0;
    //Dados do RMS
    tankAntagonista->SignalSettings->ValueTo = (int)spinRMSMax->Value;
    tankAgonista->SignalSettings->ValueTo = (int)spinRMSMax->Value;

    AtualizaBarra();
  }
}
//---------------------------------------------------------------------------
void __fastcall TfrmPrincipal::tmrTempoTimer(TObject *Sender)
{
  char Tempo[10],chrMin[10],chrSec[10];
  //Ajustar Tempo
  if(seg++ > 58) // Passou 1 Minuto
  {
    seg=0;
    min++;
  }

  itoa(seg,chrSec,10);
  itoa(min,chrMin,10);

  if(min < 10)
    strcpy(Tempo,"000");
  else if(min < 100)
    strcpy(Tempo,"00");
  else if(min < 1000)
    strcpy(Tempo,"0");
  else if(min < 10000)
    strcpy(Tempo,chrMin);

  strcat(Tempo,chrMin);
  strcat(Tempo,":");

  if(seg < 10)
    strcat(Tempo,"0");

  strcat(Tempo,chrSec);
  pnlTempo->Caption = Tempo;

  if(TempoCaptura == -1)  // Livre
    return;

  if(TempoCaptura)
    TempoCaptura--;
  else
  {
    tmrTempo->Enabled = false;
    flag_plota = false;
    btnIniciar->Caption = "Repetir";
    btnSalvar->Enabled = true;
    mnuSalvar->Enabled = true;
    btnLimpar->Enabled = true;
  }
}
//---------------------------------------------------------------------------
void __fastcall TfrmPrincipal::btnLimparClick(TObject *Sender)
{
  TColor ColorTemp1,ColorTemp2,ColorTemp3,ColorTemp4;

  ForcaMax = 0;
  gpanelRMS->ZeraPontos();
  gpanelForca->ZeraPontos();
  ColorTemp1 = (TColor)gpanelRMS->RecuperaCorLinha(1);
  ColorTemp2 = (TColor)gpanelRMS->RecuperaCorLinha(2);
  ColorTemp3 = (TColor)gpanelForca->RecuperaCorLinha(1);
  ColorTemp4 = (TColor)gpanelForca->RecuperaCorLinha(3);

  gpanelRMS->MudaCorLinha(1,gpanelRMS->Color);
  gpanelRMS->MudaCorLinha(2,gpanelRMS->Color);
  gpanelForca->MudaCorLinha(1,gpanelForca->Color);
  gpanelForca->MudaCorLinha(3,gpanelForca->Color);
```

```
  for(i = 0; i < (unsigned int)gpanelForca->DataBufSize(); i++)
  {
    gpanelRMS->Plota();
    gpanelForca->Plota();
  }

  pnlRMSAgo->Caption = "";
  pnlRMSAnta->Caption = "";
  pnlForcaValor->Caption = "";

  gpanelRMS->MudaCorLinha(1,ColorTemp1);
  gpanelRMS->MudaCorLinha(2,ColorTemp2);
  gpanelForca->MudaCorLinha(1,ColorTemp3);
  gpanelForca->MudaCorLinha(3,ColorTemp4);

  tankAgonista->Value = 0;
  tankAntagonista->Value = 0;

  flag_plota = false;
  flagCapturando = false;
  btnIniciar->Caption = "Iniciar";
  btnIniciar->Enabled = false;
  btnForcaMax->Enabled = true;
  btnCelConfig->Enabled = true;
  btnLimpar->Enabled = false;
  btnSair->Enabled = true;

  pnlMaxForca->Caption = "";
  pnlTempo->Caption = "0000:00";
  min = seg = 0;

  pnlMusculo->Enabled = true;
  pnlBtnForca->Enabled = true;
  pnlRMSCtrl->Enabled = true;
  btnSalvar->Enabled = false;
  mnuSalvar->Enabled = false;
  cmbTempo->Enabled = true;
  spinAmostras->Enabled = true;
  cmbTempo->Enabled = true;
  //Apaga Leds
  ledAlta->StatusInt = 0;
  ledBoa->StatusInt = 0;
  ledBaixa->StatusInt = 0;

  memoForca->Clear();
  memoAgo->Clear();
  memoRMSAgo->Clear();
  memoAnta->Clear();
  memoRMSAnta->Clear();
}
//---------------------------------------------------------------------------
void __fastcall TfrmPrincipal::FormShow(TObject *Sender)
{
  //flagCadastro = false;
  AtualizaBarra();
  caminho_install = "";
  // pega diretório de instalação
  caminho_install = GetCurrentDir();
  sprintf(caminho_final, "%s\\Temp.emg", caminho_install);
  PastaEscolhido = GetCurrentDir() + "\\Dados";
  dlgOpen->InitialDir = PastaEscolhido;
  dlgSave->InitialDir = PastaEscolhido;
}
//---------------------------------------------------------------------------
void __fastcall TfrmPrincipal::Cadastro2Click(TObject *Sender)
{
  frmCadastro->ShowModal();
}
//---------------------------------------------------------------------------
void __fastcall TfrmPrincipal::AtualizaBarra(void)
{
  stsBar->Panels->Items[0]->Text = "";
  stsBar->Panels->Items[1]->Text = "";

  if(flagCadastro)
    stsBar->Panels->Items[0]->Text = PastaEscolhido;
```

```
  else
    stsBar->Panels->Items[0]->Text = "Escolha um Paciente ou Calibre a Célula de Carga";

  if(CPort->Connected)
    stsBar->Panels->Items[1]->Text = " Status: Conectado "+ frmSerial->CboPorta->Text +
             " | " + frmSerial->CboBaud->Text + " | " + frmSerial->CboDat->Text +
             " | " +  frmSerial->CboPar->Text + " | " + frmSerial->CboStop->Text;
  else
    stsBar->Panels->Items[1]->Text = " Status: Desconectado";
}
```

ANEXO A.2 – Unit_Aparencia.cpp (TELA DE CONFIGURAR APARÊNCIA)

```
 //---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "Unit_Aparencia.h"
#include "Unit_EMG.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TfrmAparencia *frmAparencia;
//---------------------------------------------------------------------------
__fastcall TfrmAparencia::TfrmAparencia(TComponent* Owner)
  : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TfrmAparencia::btnCancelClick(TObject *Sender)
{
  Close();
}
//---------------------------------------------------------------------------
void __fastcall TfrmAparencia::spdGAgoClick(TObject *Sender)
{
  if(dlgColor->Execute())
  {
    if(Sender == spdGAgo)
      pnlCorAgo->Color = dlgColor->Color;
    else if (Sender == spdGAnta)
      pnlCorAnta->Color = dlgColor->Color;
    else if (Sender == spdGFundo)
      pnlCorFundoRMS->Color = dlgColor->Color;
    else if (Sender == spdTAgo)
      pnlCorTAgo->Color = dlgColor->Color;
    else if (Sender == spdTAnta)
      pnlCorTAnta->Color = dlgColor->Color;
    else if (Sender == spdForca)
      pnlCorForca->Color = dlgColor->Color;
    else if (Sender == spdLimites)
      pnlCorLim->Color = dlgColor->Color;
    else if (Sender == spdFundoForca)
      pnlCorFundoForca->Color = dlgColor->Color;
  }
}
//---------------------------------------------------------------------------
void __fastcall TfrmAparencia::btnOkClick(TObject *Sender)
{
  frmPrincipal->gpanelRMS->MudaCorLinha(1,pnlCorAgo->Color);
  frmPrincipal->gpanelRMS->MudaCorLinha(2,pnlCorAnta->Color);
  frmPrincipal->gpanelRMS->MudaCorFundo(pnlCorFundoRMS->Color);
  frmPrincipal->pnlRMSAgo->Font->Color = pnlCorAgo->Color;
  frmPrincipal->pnlRMSAnta->Font->Color = pnlCorAnta->Color;
  frmPrincipal->tankAgonista->TankSettings->Color = pnlCorTAgo->Color;
  frmPrincipal->tankAntagonista->TankSettings->Color = pnlCorTAnta->Color;

  frmPrincipal->gpanelForca->MudaCorLinha(1,pnlCorForca->Color);
  frmPrincipal->gpanelForca->MudaCorLinha(3,pnlCorLim->Color);
  frmPrincipal->gpanelForca->MudaCorFundo(pnlCorFundoForca->Color);
  frmPrincipal->pnlForcaValor->Font->Color = pnlCorForca->Color;

  frmPrincipal->pnlRMSAgo->Color = pnlCorFundoRMS->Color;
  frmPrincipal->pnlRMSAnta->Color = pnlCorFundoRMS->Color;
  frmPrincipal->pnlForcaValor->Color = pnlCorFundoForca->Color;
```

```
  Close();

}
//---------------------------------------------------------------------------
void __fastcall TfrmAparencia::FormShow(TObject *Sender)
{
  pnlCorAgo->Color = (TColor) frmPrincipal->gpanelRMS->RecuperaCorLinha(1);
  pnlCorAnta->Color =(TColor) frmPrincipal->gpanelRMS->RecuperaCorLinha(2);
  pnlCorFundoRMS->Color = frmPrincipal->gpanelRMS->Color;
  pnlCorTAgo->Color = frmPrincipal->tankAgonista->TankSettings->Color;
  pnlCorTAnta->Color = frmPrincipal->tankAntagonista->TankSettings->Color;
  pnlCorForca->Color = (TColor) frmPrincipal->gpanelForca->RecuperaCorLinha(1);
  pnlCorLim->Color = (TColor) frmPrincipal->gpanelForca->RecuperaCorLinha(3);
  pnlCorFundoForca->Color = frmPrincipal->gpanelForca->Color;
}
//---------------------------------------------------------------------------
```

ANEXO A.3 – Unit_Calibrar.cpp (TELA PARA CALIBRAR CÉLULA DE CARGA)

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "Unit_Calibrar.h"
#include "Unit_EMG.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TfrmCalibrar *frmCalibrar;
//---------------------------------------------------------------------------
__fastcall TfrmCalibrar::TfrmCalibrar(TComponent* Owner)
  : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TfrmCalibrar::ckboxManualClick(TObject *Sender)
{
  edtFator->Visible ^= 0x1;
  btnCalibrarCel->Enabled ^=0x1;
}
//---------------------------------------------------------------------------
void __fastcall TfrmCalibrar::btnSairClick(TObject *Sender)
{
  if(frmPrincipal->CPort->Connected)
    frmPrincipal->CPort->Close();

  if(ckboxManual->Checked)
    frmPrincipal->fatorForca = StrToFloat(edtFator->Text);
  frmPrincipal->flagCalibrar = false;
  Close();
}
//---------------------------------------------------------------------------
void __fastcall TfrmCalibrar::btnCalibrarCelClick(TObject *Sender)
{
  if(btnCalibrarCel->Caption == "Iniciar")
  {
    frmPrincipal->flag_plota = true;
    btnCalibrarCel->Caption = "Parar";
    btnSair->Enabled = false;
    ckboxManual->Enabled = false;
    edtPeso->ReadOnly = true;

    if(!frmPrincipal->CPort->Connected)
      frmPrincipal->CPort->Open();
    frmPrincipal->CPort->ClearBuffer(true,true);
  }
  else if(btnCalibrarCel->Caption == "Parar")
  {
    frmPrincipal->flag_plota = false;
    btnCalibrarCel->Caption = "Iniciar";
    btnSair->Enabled = true;
    ckboxManual->Enabled = true;
```

```
    edtPeso->ReadOnly = false;

    if(frmPrincipal->CPort->Connected)
      frmPrincipal->CPort->Close();
  }
}
//---------------------------------------------------------------------------
void __fastcall TfrmCalibrar::FormShow(TObject *Sender)
{
  pnlFatorCorr->Caption = "";
}
//---------------------------------------------------------------------------
```

ANEXO A.4 – Unit_Forca_Max.cpp (TELA PARA CAPTURAR A FORÇA MÁXIMA)

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "Unit_Forca_Max.h"
#include "Unit_EMG.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma link "_GClass"
#pragma link "AbTank"
#pragma link "CSPIN"
#pragma resource "*.dfm"
TfrmForcaMax *frmForcaMax;
//---------------------------------------------------------------------------
__fastcall TfrmForcaMax::TfrmForcaMax(TComponent* Owner)
  : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TfrmForcaMax::btnSairClick(TObject *Sender)
{
  spinTentativas->Value = tentativa;
  if(btnSair->Caption == "Cancelar")
  {
    btnIniciar->Caption = "Iniciar";
    btnSair->Caption = "Sair";
    delete vetForcaMax;
    frmPrincipal->ForcaMax = 0;
    FormShow(btnSair);
  }
  else if(btnSair->Caption == "Sair")
  {
    frmPrincipal->flagForcaMax = false;
    if(frmPrincipal->CPort->Connected)
      frmPrincipal->CPort->Close();
    frmPrincipal->ForcaMax =0;
    for(i=0;i<tentativa;i++)
      frmPrincipal->ForcaMax += vetForcaMax[tentativa];
    frmPrincipal->ForcaMax /= tentativa;
    frmPrincipal->pnlMaxForca->Caption = frmPrincipal->MostraValor(frmPrincipal->ForcaMax);
    if(frmPrincipal->ForcaMax)  //Se ForcaMax != 0 -> Ocorreu a captura
      frmPrincipal->btnIniciar->Enabled = true;   //Habilita o Botão Iniciar do Form Principal
    Close();
  }
}
//---------------------------------------------------------------------------
void __fastcall TfrmForcaMax::btnIniciarClick(TObject *Sender)
{
  if(btnIniciar->Caption == "Iniciar")
  {
    frmPrincipal->flag_plota = true;
    btnIniciar->Caption = "Parar";
    btnSair->Enabled = false;
    if(!frmPrincipal->CPort->Connected)
      frmPrincipal->CPort->Open();
    frmPrincipal->CPort->ClearBuffer(true,true);
    frmPrincipal->ForcaMax = 0;
    tentativa = (int)spinTentativas->Value;
    vetForcaMax = new float[tentativa];
```

```
  }
  else if(btnIniciar->Caption == "Parar")   // Se Caption = Parar
  {
    frmPrincipal->flag_plota = false;
    vetForcaMax[(int)spinTentativas->Value] = frmPrincipal->ForcaMax;
    spinTentativas->Value--;
    if((int)spinTentativas->Value)
    {
      btnIniciar->Caption = "Continuar";
      btnSair->Caption = "Cancelar";
    }
    else
    {
      btnIniciar->Caption = "Iniciar";
      btnSair->Caption = "Sair";
      spinTentativas->Value = tentativa;
    }
    btnSair->Enabled = true;
    if(frmPrincipal->CPort->Connected)
      frmPrincipal->CPort->Close();
  }
}
//---------------------------------------------------------------------------


void __fastcall TfrmForcaMax::FormShow(TObject *Sender)
{
  tankForcaMax->Value = 0;
  pnlForcaMax->Caption = "";
  pnlForcaAtual->Caption = "";
}
//---------------------------------------------------------------------------
```

ANEXO A.5 – Unit_Serial.cpp (TELA PARA CONFIGURAR COMUNICAÇÃO SERIAL)

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "Unit_Serial.h"
#include "Unit_EMG.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma link "CPortCtl"
#pragma resource "*.dfm"
TfrmSerial *frmSerial;
//---------------------------------------------------------------------------
__fastcall TfrmSerial::TfrmSerial(TComponent* Owner)
  : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TfrmSerial::btnOkClick(TObject *Sender)
{
  try
   {
      CboPorta->ApplySettings();
      CboBaud->ApplySettings();
      CboDat->ApplySettings();
      CboPar->ApplySettings();
      CboStop->ApplySettings();
   }
   catch(...)
   {
      ShowMessage("Erro ao alterar as configurações da porta serial!\nAs configurações default
serão restauradas.");
      Restaura();
   }
   Close();
}
//---------------------------------------------------------------------------
void __fastcall TfrmSerial::FormShow(TObject *Sender)
```

```
{
  Restaura();
}
//---------------------------------------------------------------------------
void __fastcall TfrmSerial::Restaura()
{
   CboPorta->UpdateSettings();
   CboBaud->UpdateSettings();
   CboDat->UpdateSettings();
   CboPar->UpdateSettings();
   CboStop->UpdateSettings();
}
//---------------------------------------------------------------------------
void __fastcall TfrmSerial::btnCancelClick(TObject *Sender)
{
  Close();
}
```

ANEXO A.6 – Unit_Sobre.cpp (TELA SOBRE)

```
//---------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "Unit_Sobre.h"
#include "Unit_EMG.h"
//---------------------------------------------------------------------
#pragma resource "*.dfm"
TfrmSobre *frmSobre;
//---------------------------------------------------------------------
__fastcall TfrmSobre::TfrmSobre(TComponent* AOwner)
        : TForm(AOwner)
{
}
//---------------------------------------------------------------------
void __fastcall TfrmSobre::OKButtonClick(TObject *Sender)
{
  Close();
}
//---------------------------------------------------------------------------
```

ANEXO A.7 – Unit_Cadastro.cpp (TELA PARA CADASTRO DE SUJEITOS)

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "untCadastro.h"
#include "Unit_EMG.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma link "CSPIN"
#pragma resource "*.dfm"
TfrmCadastro *frmCadastro;
//---------------------------------------------------------------------------
__fastcall TfrmCadastro::TfrmCadastro(TComponent* Owner)
  : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TfrmCadastro::CampoStat(bool Stat)
{
  TColor CorFundo;

  if(Stat)
    CorFundo = clWhite;
  else
    CorFundo = clBtnFace;

  edtNome->ReadOnly = !Stat;
  edtNome->Color = CorFundo;
```

```
  spinIdade->ReadOnly = !Stat;
  spinIdade->Color = CorFundo;
  rdSexo->Enabled = Stat;
  memoComentario->ReadOnly = !Stat;
  memoComentario->Color = CorFundo;

}
//---------------------------------------------------------------------------
void __fastcall TfrmCadastro::BotoesStat(unsigned short Stat)
{
  btnPrimeiro->Enabled = (Stat & 0x0100) >> 8;
  btnAnterior->Enabled = (Stat & 0x0080) >> 7;
  btnProximo->Enabled  = (Stat & 0x0040) >> 6;
  btnUltimo->Enabled   = (Stat & 0x0020) >> 5;
  btnAdicionar->Enabled= (Stat & 0x0010) >> 4;
  btnExcluir->Enabled  = (Stat & 0x0008) >> 3;
  btnConfirmar->Enabled= (Stat & 0x0004) >> 2;
  btnCancelar->Enabled = (Stat & 0x0002) >> 1;
  btnAtualizar->Enabled= (Stat & 0x0001);
}
//---------------------------------------------------------------------------
void __fastcall TfrmCadastro::AtualizaBotoes(bool Add)
{
  int Count,Index;

  Count = listCadastrados->Items->Count,
  Index = listCadastrados->ItemIndex;

  if(Add)//Adicionando
  {
    BotoesStat(0x0006);
    btnEditar->Enabled = false;
  }
  else
  {
    if(Count == 1)
    {
      BotoesStat(0x0019);
      btnEditar->Enabled = true;
    }
    else if(Count>1) // A lista possui itens
    {
      btnEditar->Enabled = true;
      if(Index == 0)
        BotoesStat(0x0079);
      else if((Index > 0)&&(Index<Count-1))
        BotoesStat(0x01F9);
      else if(Index == Count -1)
        BotoesStat(0x0199);
      else if(Index == -1)
      {
        BotoesStat(0x0070);
        btnEditar->Enabled = false;
      }
    }
    else
    {
      BotoesStat(0x0010);
      btnEditar->Enabled = false;
    }
  }

}
//---------------------------------------------------------------------------
void __fastcall TfrmCadastro::EMG1Click(TObject *Sender)
{
  if(!flagSalvo)
  {
    if(Application->MessageBox("Deseja salvar os Dados?","Informação", MB_YESNO |
MB_ICONINFORMATION) == IDYES)
      btnSalvarClick(btnSalvar);
    else
    {
      frmPrincipal->flagCadastro = false;
      listCadastrados->ItemIndex = -1;
      frmPrincipal->AtualizaBarra();
      Close();
```

```
      return;
    }
  }

  if(listCadastrados->ItemIndex > -1) // Alguem Foi escolhido
  {
    btnAtualizarClick(btnAtualizar);  // Atualiza os Campos
    frmPrincipal->flagCadastro = true;              //Alguem foi escolhido
    frmPrincipal->PastaEscolhido = memoPasta->Text;
    frmPrincipal->PessoaEscolhida = edtNome->Text.Trim();
    frmPrincipal->AtualizaBarra();
  }

  Close();
}
//---------------------------------------------------------------------------
void __fastcall TfrmCadastro::btnBuscarClick(TObject *Sender)
{
  dlgProcurar->Execute();
}
//---------------------------------------------------------------------------
void __fastcall TfrmCadastro::btnAdicionarClick(TObject *Sender)
{
  edtNome->Text = "";
  edtNome->SetFocus();
  spinIdade->Value = 18;
  rdSexo->ItemIndex = -1;
  memoComentario->Lines->Clear();
  memoPasta->Clear();
  listCadastrados->Enabled = false;
  btnEditar->Enabled = false;
  btnBuscar->Enabled = false;
  btnSalvar->Enabled = false;

  CampoStat(true);
  AtualizaBotoes(true);
}
//---------------------------------------------------------------------------
void __fastcall TfrmCadastro::btnConfirmarClick(TObject *Sender)
{
  int index,chrPos,intTemp,line;
  AnsiString  strTemp,strNome,strOld,strNew,
              defDiretorio = GetCurrentDir();

  if(flagEditando)
  {
    index = listCadastrados->ItemIndex;
    if((rdSexo->ItemIndex > -1)&&(edtNome->Text != "")&&(index > -1))
    {
      if(flagApagar)
      {
        memoDados->Lines->Delete(memoDados->Lines->Count - 1);
        flagApagar = false;
      }
      btnSalvar->Enabled = true;
      flagSalvo = false;

      listCadastrados->Enabled = true;
      index = listCadastrados->ItemIndex;
      strTemp = listCadastrados->Items->Strings[index];

      chrPos = strTemp.Pos("-");

      strNome = strTemp.SubString(1,chrPos - 1);
      strNome = strNome.Trim();
      listCadastrados->Items->Delete(index);

    //Atualizando Memo Dados
      index = memoDados->Lines->IndexOf(strNome);
      intTemp = StrToInt(memoDados->Lines->Strings[index + 4]);
      for(line=0;line<5+intTemp;line++)
        memoDados->Lines->Delete(index);
      //Realoca novas Entradas
      memoDados->Lines->Add(edtNome->Text);
      memoDados->Lines->Add(IntToStr(rdSexo->ItemIndex));
      memoDados->Lines->Add(IntToStr(spinIdade->Value));
      memoDados->Lines->Add(defDiretorio + "\\Dados\\" + edtNome->Text.Trim());
```

```
      memoDados->Lines->Add(IntToStr(memoComentario->Lines->Count));
      memoDados->Lines->AddStrings(memoComentario->Lines);

      //Atualizando Memo Arquivo
      index = memoArquivo->Lines->IndexOf("Nome:\t" + strNome);
      for(line=0;line<7+intTemp;line++)
        memoArquivo->Lines->Delete(index);
      //Realocando novos Dados no MemoArquivo
      memoArquivo->Lines->Add("Nome:\t" + edtNome->Text.Trim());
      memoArquivo->Lines->Add("Sexo:\t" + rdSexo->Items->Strings[rdSexo->ItemIndex]);
      memoArquivo->Lines->Add("Idade:\t" + IntToStr(spinIdade->Value));
      memoArquivo->Lines->Add("Pasta de Trabalho:\t" + defDiretorio + "\\" + edtNome-
>Text.Trim());
      memoArquivo->Lines->Add("Comentários:");
      memoArquivo->Lines->AddStrings(memoComentario->Lines);
      memoArquivo->Lines->Add(" ");
      memoArquivo->Lines->Add(strTemp.StringOfChar('-',200));

      index = listCadastrados->Items->Add(edtNome->Text+" - "+rdSexo->Items->Strings[rdSexo-
>ItemIndex]
                                +" - "+IntToStr(spinIdade->Value));
      listCadastrados->ItemIndex = index;
      memoPasta->Clear();
      strTemp = defDiretorio + "\\Dados\\";
      memoPasta->Lines->Add(strTemp + edtNome->Text.Trim());
      strOld = strTemp + strNome;
      strNew = strTemp + edtNome->Text;
      rename(strOld.c_str(),strNew.c_str());
      CampoStat(false);
      AtualizaBotoes(false);
      flagEditando = false;
      listCadastrados->Enabled = true;
      btnEditar->Enabled = true;
      btnBuscar->Enabled = true;
      btnSalvar->Enabled = true;

    }
    else
      ShowMessage("Dados Incompletos!");
  }
  else  // Adicionando
  {
    if((rdSexo->ItemIndex > -1)&&(edtNome->Text != ""))
    {
      if(flagApagar)
      {
        memoDados->Lines->Delete(memoDados->Lines->Count - 1);
        flagApagar = false;
      }
      btnSalvar->Enabled = true;
      flagSalvo = false;

      index = listCadastrados->Items->Add(edtNome->Text.Trim()+" - "+rdSexo->Items-
>Strings[rdSexo->ItemIndex]
                                +" - "+IntToStr(spinIdade->Value));

      //Adiciona no memoArquivo
      memoArquivo->Lines->Add("Nome:\t" + edtNome->Text.Trim());
      memoArquivo->Lines->Add("Sexo:\t" + rdSexo->Items->Strings[rdSexo->ItemIndex]);
      memoArquivo->Lines->Add("Idade:\t" + IntToStr(spinIdade->Value));
      memoArquivo->Lines->Add("Pasta de Trabalho:\t" + defDiretorio + "\\" + edtNome-
>Text.Trim());
      memoArquivo->Lines->Add("Comentários:");
      memoArquivo->Lines->AddStrings(memoComentario->Lines);
      memoArquivo->Lines->Add(" ");
      memoArquivo->Lines->Add(strTemp.StringOfChar('-',200));

      listCadastrados->ItemIndex = index;
      strNome = edtNome->Text;
      strNome = strNome.Trim();
      memoPasta->Lines->Add(defDiretorio + "\\" + strNome);
      totalCadastro++;
      CampoStat(false);
      AtualizaBotoes(false);
      listCadastrados->Enabled = true;
      btnEditar->Enabled = true;
      btnBuscar->Enabled = true;
```

```
      btnSalvar->Enabled = true;

      memoDados->Lines->Add(edtNome->Text);
      memoDados->Lines->Add(IntToStr(rdSexo->ItemIndex));
      memoDados->Lines->Add(IntToStr(spinIdade->Value));
      memoDados->Lines->Add(defDiretorio + "\\" + strNome);
      memoDados->Lines->Add(IntToStr(memoComentario->Lines->Count));
      memoDados->Lines->AddStrings(memoComentario->Lines);

      //Criando as Pastas
      defDiretorio += "\\Dados\\" + strNome;
      mkdir(defDiretorio.c_str());
      for(line=0;line < frmPrincipal->boxMusculo->Items->Count;line++)
      {
        strTemp = defDiretorio + "\\" + frmPrincipal->boxMusculo->Items->Strings[line];
        mkdir(strTemp.c_str());
      }
    }
    else
      ShowMessage("Dados Incompletos!");
  }
}
//---------------------------------------------------------------------------
void __fastcall TfrmCadastro::btnCancelarClick(TObject *Sender)
{
   flagEditando = false;
   btnAtualizarClick(btnCancelar);
   listCadastrados->Enabled = true;
   btnBuscar->Enabled = true;
   if(flagSalvo)
     btnSalvar->Enabled = false;
   else
    btnSalvar->Enabled = true;
   CampoStat(false);
   AtualizaBotoes(false);
}
//---------------------------------------------------------------------------
void __fastcall TfrmCadastro::btnExcluirClick(TObject *Sender)
{
  int index,chrPos,intTemp,line;
  AnsiString strTemp,strIdade,strSexo,strNome;

  if((listCadastrados->ItemIndex > -1)&&(Application->MessageBox("Deseja apagar o Registro?",
       "Informação", MB_YESNO | MB_ICONINFORMATION) == IDYES))
  {
    if(flagApagar)
    {
      memoDados->Lines->Delete(memoDados->Lines->Count - 1);
      flagApagar = false;
    }
    btnSalvar->Enabled = true;
    flagSalvo = false;

    index = listCadastrados->ItemIndex;
    strTemp = listCadastrados->Items->Strings[index];
    chrPos = strTemp.Pos("-");
    strNome = strTemp.SubString(1,chrPos - 1);
    strNome = strNome.Trim();

    // Atualizando Memo Dados
    index = memoDados->Lines->IndexOf(strNome);
    intTemp = StrToInt(memoDados->Lines->Strings[index + 4]);
    for(line=0;line<5+intTemp;line++)
      memoDados->Lines->Delete(index);

    //Atualizando Memo Arquivo
    index = memoArquivo->Lines->IndexOf("Nome:\t" + strNome);
    for(line=0;line<7+intTemp;line++)
      memoArquivo->Lines->Delete(index);

    listCadastrados->Items->Delete(listCadastrados->ItemIndex);
    totalCadastro--;
    listCadastrados->ItemIndex = index;
    if(listCadastrados->ItemIndex == -1)
    {
      edtNome->Text = "";
      spinIdade->Value = 18;
```

```
      rdSexo->ItemIndex = -1;
      memoComentario->Clear();
      memoPasta->Clear();
    }
    else
      btnAtualizarClick(btnAtualizar);
    strTemp = GetCurrentDir() + "\\Dados\\" + strNome;
    unlink(strTemp.c_str());
    //DeleteFile(strTemp.c_str());
    CampoStat(false);
    AtualizaBotoes(false);
  }
}
//---------------------------------------------------------------------------
void __fastcall TfrmCadastro::btnProximoClick(TObject *Sender)
{
  if(listCadastrados->ItemIndex < listCadastrados->Items->Count-1)
    listCadastrados->ItemIndex++;
  btnAtualizarClick(btnAtualizar);
  AtualizaBotoes(false);
}
//---------------------------------------------------------------------------
void __fastcall TfrmCadastro::btnUltimoClick(TObject *Sender)
{
  if(listCadastrados->ItemIndex < listCadastrados->Items->Count-1)
    listCadastrados->ItemIndex = listCadastrados->Items->Count-1;
  btnAtualizarClick(btnAtualizar);
  AtualizaBotoes(false);
}
//---------------------------------------------------------------------------
void __fastcall TfrmCadastro::btnAnteriorClick(TObject *Sender)
{
  if(listCadastrados->ItemIndex > 0)
    listCadastrados->ItemIndex--;
  btnAtualizarClick(btnAtualizar);
  AtualizaBotoes(false);
}
//---------------------------------------------------------------------------
void __fastcall TfrmCadastro::btnPrimeiroClick(TObject *Sender)
{
  if(listCadastrados->ItemIndex > 0)
    listCadastrados->ItemIndex = 0;
  btnAtualizarClick(btnAtualizar);
  AtualizaBotoes(false);
}
//---------------------------------------------------------------------------
void __fastcall TfrmCadastro::FormShow(TObject *Sender)
{
  edtNome->Text ="";
  rdSexo->ItemIndex =-1;
  spinIdade->Value = 18;
  memoComentario->Clear();
  memoPasta->Clear();
  listCadastrados->ItemIndex = -1;
  flagEditando = false;
  flagSalvo = true;
  btnSalvar->Enabled = false;
  btnBuscar->Enabled = true;
  AtualizaBotoes(false);
}
//---------------------------------------------------------------------------
void __fastcall TfrmCadastro::btnSalvarClick(TObject *Sender)
{
  if(flagApagar)
    memoDados->Lines->Delete(memoDados->Lines->Count - 1);
  flagApagar = true;
  btnSalvar->Enabled = false;
  flagSalvo = true;
  memoDados->Lines->Add(IntToStr(totalCadastro));
  memoDados->Lines->SaveToFile(CaminhoArq);
  memoArquivo->Lines->SaveToFile(fileCadastro);

}
//---------------------------------------------------------------------------
void __fastcall TfrmCadastro::FormCreate(TObject *Sender)
{
  AnsiString strSexo,strNome,strIdade,strPathCadastro;
```

```
  unsigned int j=0,nComent,intSexo;
  FILE *pt1;

  CaminhoArq = GetCurrentDir() + "\\Dados.txt";
  fileCadastro = GetCurrentDir() + "\\Dados\\Cadastro.txt";

  if((pt1 = fopen(fileCadastro.c_str(),"r")) != NULL)
    memoArquivo->Lines->LoadFromFile(fileCadastro);
  fclose(pt1);

  //Dados
  if((pt1 = fopen(CaminhoArq.c_str(),"r")) != NULL)
  {
    memoDados->Lines->LoadFromFile(CaminhoArq);
    totalCadastro = StrToInt(memoDados->Lines->Strings[memoDados->Lines->Count - 1]);
    memoDados->Lines->Delete(memoDados->Lines->Count - 1);

    for(i=0;i<totalCadastro;i++)
    {
      strNome = memoDados->Lines->Strings[j++];
      intSexo = StrToInt(memoDados->Lines->Strings[j++]);
      strIdade = memoDados->Lines->Strings[j++];
      j++;    //Pula Arquivos;
      nComent =StrToInt(memoDados->Lines->Strings[j++]);
      while(nComent--)
        j++;

      if(intSexo)
        strSexo = "F";
      else
        strSexo = "M";
      listCadastrados->Items->Add(strNome+" - "+strSexo+" - "+strIdade);
    }
  }
  else  // Arquivo não existe
  {
    edtNome->Text ="";
    rdSexo->ItemIndex =-1;
    spinIdade->Value = 18;
    memoComentario->Clear();
    memoPasta->Clear();
    listCadastrados->Clear();
    memoDados->Clear();
    memoArquivo->Clear();
  }
  flagApagar = false;
  fclose(pt1);
}
//---------------------------------------------------------------------------
void __fastcall TfrmCadastro::btnAtualizarClick(TObject *Sender)
{
  int index,chrPos,intTemp;
  AnsiString strTemp,strIdade,strSexo,strNome;

  index = listCadastrados->ItemIndex;
  if(index == -1)
  {
    edtNome->Text = "";
    spinIdade->Value = 18;
    rdSexo->ItemIndex = -1;
    memoComentario->Clear();
    memoPasta->Clear();
    AtualizaBotoes(false);
    return;
  }

  strTemp = listCadastrados->Items->Strings[index];
  chrPos = strTemp.Pos("-");

  strNome = strTemp.SubString(1,chrPos - 1);
  strNome = strNome.Trim();
  edtNome->Text = strNome;

  strSexo = strTemp.SubString(chrPos + 2,1);
  if(strSexo == "M")
    intTemp = 0;
  else if (strSexo == "F")
```

```
    intTemp = 1;
  else
    intTemp = -1;
  rdSexo->ItemIndex = intTemp;
  strIdade = strTemp.SubString(chrPos+6,3);
  strIdade = strIdade.Trim();
  spinIdade->Value = StrToInt(strIdade);

  index = memoDados->Lines->IndexOf(strNome);
  memoPasta->Clear();
  memoPasta->Lines->Add(memoDados->Lines->Strings[index+3]);
  intTemp = StrToInt(memoDados->Lines->Strings[index + 4]);
  memoComentario->Clear();
  while(intTemp--)
    memoComentario->Lines->Add(memoDados->Lines->Strings[index++ + 5]);
  AtualizaBotoes(false);
}
//---------------------------------------------------------------------------
void __fastcall TfrmCadastro::btnEditarClick(TObject *Sender)
{
  btnAtualizarClick(btnEditar);
  CampoStat(true);
  AtualizaBotoes(true);
  listCadastrados->Enabled = false;
  flagEditando = true;
  btnEditar->Enabled = false;
  btnBuscar->Enabled = false;
  btnSalvar->Enabled = false;
}
//---------------------------------------------------------------------------
void __fastcall TfrmCadastro::dlgProcurarFind(TObject *Sender)
{
  AnsiString strNome,listNome,strTemp;
  int index,chrPos,indFind = -1;

  strNome = dlgProcurar->FindText;
  for(index=0;index<listCadastrados->Items->Count;index++)
  {
    strTemp = listCadastrados->Items->Strings[index];
    chrPos = strTemp.Pos("-");
    listNome = strTemp.SubString(1,chrPos - 1);
    listNome = listNome.Trim();
    if(listNome == strNome)
    {
      indFind = index;
      break;
    }
  }

  if(indFind == -1)
    ShowMessage("Pessoa não Cadastrada!");
  else
  {
    listCadastrados->ItemIndex = indFind;
    btnAtualizarClick(btnAtualizar);
  }
  dlgProcurar->CloseDialog();
}
//---------------------------------------------------------------------------

void __fastcall TfrmCadastro::listCadastradosKeyDown(TObject *Sender,
      WORD &Key, TShiftState Shift)
{
  btnAtualizarClick(btnAtualizar);
  AtualizaBotoes(false);
}
//---------------------------------------------------------------------------
```

ANEXO B – FIRMWARE DO MSP430

```
//-------------------------------------------------------------------------------------
# include "msp430x14x.h"
# include "iso646.h"
# include "stddef.h"

//-------------------------------------------------------------------------------------
//Funções
void Configurar_Clock(void); // Configura o clock.
void Configurar_UART(void);  // Configura a UART0 e UART1.
void Configurar_ConversorAD(void);  // configura o conversor AD.
void Configurar_TimerA(void); //Configura o TimerA para q ele comande a taxa de amostragem.
void Enviar(unsigned int result, int canal);  //Envia o valor da conversão.

//variáveis globais.
int flagForca,flagTrigger,count;

void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;    // Stop WDT

  Configurar_Clock();
  Configurar_ConversorAD();
  Configurar_UART();
  flagForca = flagTrigger =0;
  count = 0;
  P1DIR = 0x0F;
  Configurar_TimerA();
  _BIS_SR(GIE);               // habilita as interrupções.
  while(1);                   //loop infinito para parar o programa.
}

//-------------------------------------------------------------------------------------
void Configurar_Clock(void)
{
   unsigned int i;

  BCSCTL1 &= ~XT2OFF;                   // XT2on

  do
  {
  IFG1 &= ~OFIFG;                       // Clear OSCFault flag
  for (i = 0xFF; i > 0; i--);           // Time for flag to set
  }
  while ((IFG1 & OFIFG));              // OSCFault flag still set?

  BCSCTL2 |= SELM_2 + SELS+DIVS_0;    // MCLK = 8MHz, SMCLK =8MHz.
}
//-------------------------------------------------------------------------------------
void Configurar_UART(void)
{
  UCTL1 &= 0x00;      // Configuração da porta : 115200bps , 8bits , Paridade None, 2 stops
  UCTL1 &= ~ SWRST;   // Desabilita o reset por software
  P3SEL |= 0xFF;      // Seleciona os pinos da porta 3 como UART e como saída
  P3DIR |= 0xFF;
  UCTL1 |= CHAR;                 // 8-bit caracter com 2 bits de parada.
  UTCTL1 |= SSEL1+SSEL0;         // UCLK = SMCLK.
  UBR11 = 0;                     // 8MHz/115200=069 (aqui é posto o zero).
  UBR01 = 0x45;                  // 8MHz/115200=069 (aqui, o 69 em hexadecimal).
  UMCTL1 = 0;                    // o resto de 8MHz/115200 (modulation).
  ME2 |= UTXE1;                  // Enable USART1 TXD/RXD.
}
//-------------------------------------------------------------------------------------
void Configurar_ConversorAD(void)
{
  P6SEL = 0xF;                           // Seleciona a funçào de AD da porta P6
  P6DIR &=0x00;                           // P6 como entrada

  ADC12CTL0 = SHT0_8 + MSC + REF2_5V + REFON + ADC12ON;

 /*Usa 4 períodos de relógio para amostragem,
   referência de tensão interna ligada e em 2,5V,
```

```
   amostra o canal 0 uma vez, liga o conversor.*/

  ADC12CTL1 = CSTARTADD_0 + SHS_0 + SHP + ADC12DIV_0 + ADC12SSEL_3 +CONSEQ_1;

/*Salvar o resultado da conversão primeiro em ADC12MEM0, S/H controlado pelo bit
ADC12SC do registrador ADC12CTL0,SAMPCOM do timer de amostragem,clk dividido por 1 e
vindo do oscilador, modo de conversão (conversão repetida de um canal). */

  ADC12IE = 0x000F;                       // Habilita interrupção do canal 0 a 3 (ADC12MCTL0 a
ADC12MCTL3)
  ADC12MCTL0 = INCH_0 + SREF_0;
  ADC12MCTL1 = INCH_1 + SREF_0;
  ADC12MCTL2 = INCH_2 + SREF_0;
  ADC12MCTL3 = INCH_3 + SREF_0 + EOS;

  ADC12CTL0 |= ENC; // Conversion enabled
}

//-------------------------------------------------------------------------------------
void Configurar_TimerA(void)
{
/*A interrupção do TIMER_A serve para ligar e desligar o conversor. Cada vez q isso
é feito, uma amostra do canal 0 é feita.*/

  TACTL = TASSEL_2 + MC_1 +ID_3 + TAIE; // SMCLK, cont UP mode, CLK=8MHz/8, interrupt.
  TACCTL1 = CM_1+SCS; /*Modo de captura na subida do clk e sincronizado com o mesmo.*/
  CCR0 = 999;                          /* TimerA conta até 1000(o período é CCR0+1) e
                                          gera uma interrupção.*/
}

//-------------------------------------------------------------------------------------
void Enviar(unsigned int result, int canal)
{     int aux;

   aux = (result & 0xFF00) >> 8;           // Pega o byte mais significativo
   aux = aux | (canal << 4 );              // Coloca o número do canal  nos bits não usados
   while((UTXIFG1&IFG2) == 0);             // Espera até buffer de transmissão estar vazio
   TXBUF1 = aux;                           // Transmite byte mais significativo
   aux = result & 0x00FF;                  // Pega byte menos significativo
   while((UTXIFG1&IFG2) == 0);             // Espera até buffer de transmissão estar vazio
   TXBUF1 = aux;                           // Transmite byte menos significativo
}
//-------------------------------------------------------------------------------------
// Timer_A Interrupt Vector (TAIV) handler
#pragma vector=TIMERA1_VECTOR
__interrupt void Timer_A(void)
{
  if(count < 19)
    count++;
  else
  {
    count = 0;
    flagForca = 1;
    flagTrigger = 1;
  }

  switch( TAIV )
  {
    case 10:

        ADC12CTL1 |= CSTARTADD_0;
        ADC12CTL0 |= ADC12SC;                    // Inicia a conversão

    break;

    default: break;
  }
}

//-------------------------------------------------------------------------------------
#pragma vector=ADC_VECTOR
__interrupt void ADC12ISR (void)
{
  unsigned int result;
   switch(ADC12IV)                        //Determina qual canal gerou a interrupção
   {
     case(0x006):     //Canal Força
```

```
      ADC12IE &= 0x000E;
      result=ADC12MEM0;
      if(flagForca)
      {
        Enviar(result,0);
        flagForca = 0;
      }
    break;
    case(0x008):   //Canal EMG Agonista
      ADC12IE &= 0x000C;
      result=ADC12MEM1;
      Enviar(result,1);
    break;
    case(0x00A):   //Canal EMG Antagonista
      ADC12IE &= 0x0008;
      result=ADC12MEM2;
      Enviar(result,2);
    break;
    case(0x00C):   //Canal Trigger
      result=ADC12MEM3;
      ADC12IE |= 0x000F;
      if(flagTrigger)
      {
        Enviar(result,3);
        flagTrigger = 0;
      }
   break;

    default: break;
  }
}
```

ANEXO C: PROGRAMA DO MATLAB UTILIZADO PARA A VALIDAÇÃO

```
clear all; close all; clc;

% Carrega os Dados
Forca5Hz  = load('50Hz-Ago.txt ');
Forca10Hz = load('100Hz-Ago.txt');
Forca25Hz = load('250Hz-Ago.txt');
Forca50Hz = load('450Hz-Ago.txt');

% Determina, entre os arquivos abertos,
% qual o vetor de menor tamanho
len5 = length(Forca5Hz);
len10 = length(Forca10Hz);
len25 = length(Forca25Hz);
len50 = length(Forca50Hz);

minLen = min([len5,len10,len25,len50]); % menor tamanho

%Faz com que os vetores possuem o mesmo tamanho
Forca5Hz  = Forca5Hz(1:minLen);
Forca10Hz = Forca10Hz(1:minLen);
Forca25Hz = Forca25Hz(1:minLen);
Forca50Hz = Forca50Hz(1:minLen);

%Retira o DC
DC5  = mean(Forca5Hz);
DC10 = mean(Forca10Hz);
DC25 = mean(Forca25Hz);
DC50 = mean(Forca50Hz);

Forca5Hz  = Forca5Hz - DC5;
Forca10Hz = Forca10Hz - DC10;
Forca25Hz = Forca25Hz - DC25;
Forca50Hz = Forca50Hz - DC50;

%Plota os sinais
fs = 1000;       % Taxa de Amostragem.
t=(0:minLen - 1)*1/fs; % Eixo do Tempo

% Figura: Tensao vs Tempo
figure, subplot(2,2,1),plot(t,Forca5Hz);
 axis([0 0.1 min(Forca5Hz) max(Forca5Hz)]);
 title('Frequencia = 50 Hz');
 ylabel('Tensao (V)'); xlabel('Tempo (s)');
 subplot(2,2,2),plot(t,Forca10Hz);
  axis([0 0.1 min(Forca10Hz) max(Forca10Hz)]);
  title('Frequencia = 100 Hz');
  ylabel('Tensao (V)'); xlabel('Tempo (s)');
 subplot(2,2,3), plot(t,Forca25Hz);
  axis([0 0.1 min(Forca25Hz) max(Forca25Hz)]);
  title('Frequencia = 250 Hz');
  ylabel('Tensao (V)'); xlabel('Tempo (s)');
 subplot(2,2,4), plot(t,Forca50Hz);
  axis([0 0.1 min(Forca50Hz) max(Forca50Hz)]);
  title('Frequencia = 450 Hz');
  ylabel('Tensao (V)'); xlabel('Tempo (s)');
```

```
 set(gcf,'Color','white')


% Definindo eixo Frequencia
f = (0:minLen - 1)*fs/minLen;

% Calcula a FFT dos sinais
fft5 = abs(fft(Forca5Hz));
fft10 = abs(fft(Forca10Hz));
fft25 = abs(fft(Forca25Hz));
fft50 = abs(fft(Forca50Hz));

% Normaliza para visualizarmos melhor
% o posicionamento espectral
maxF5 = max(fft5);
maxF10 = max(fft10);
maxF25 = max(fft25);
maxF50 = max(fft50);

fft5 = fft5 / maxF5;
fft10 =fft10 / maxF10;
fft25 =fft25 / maxF25;
fft50 =fft50 / maxF50;

% Figura: Espectro de Frequencia
 figure,subplot(4,1,1),plot(f,fft5);
        title('Transformada de Fourier dos Sinais Adquiridos');
        ylabel('Frequencia 50Hz');
        axis([0 fs/2 0 1]);
 subplot(4,1,2),plot(f,fft10);
         ylabel('Frequencia 100Hz');
         axis([0 fs/2 0 1]);
 subplot(4,1,3),plot(f,fft25);
         ylabel('Frequencia 250Hz');
         axis([0 fs/2 0 1]);
 subplot(4,1,4),plot(f,fft50);
        ylabel('Frequencia 450Hz');
        axis([0 fs/2 0 1]);
    xlabel('Frequencia (Hz)');
set(gcf,'Color','white');
```