UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

GUILHERME RAUTER CORSETTI

DESENVOLVIMENTO DE UM MOUSE VIRTUAL POR BIOSINAIS

Porto Alegre
2010

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

DESENVOLVIMENTO DE UM MOUSE VIRTUAL POR BIOSINAIS

Projeto de Diplomação apresentado ao Departamento de Engenharia Elétrica da Universidade Federal do Rio Grande do Sul, como parte dos requisitos para Graduação em Engenharia Elétrica.

ORIENTADOR: Prof. Dr. Alexandre Balbinot

Porto Alegre
2010

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

GUILHERME RAUTER CORSETTI

DESENVOLVIMENTO DE UM MOUSE VIRTUAL POR BIOSINAIS

Este projeto foi julgado adequado para fazer jus aos créditos da Disciplina de "Projeto de Diplomação", do Departamento de Engenharia Elétrica e aprovado em sua forma final pelo Orientador e pela Banca Examinadora.

Orientador: _____

Prof. Dr. Alexandre Balbinot, UFRGS

Doutor pela UFRGS – Porto Alegre, Brasil

Banca Examinadora:

Prof[a]. Dra. Adriane Parraga, UFRGS

Doutora pela UFRGS – Porto Alegre, Brasil

Prof. Dr. Luís Alberto Pereira, UFRGS

Doutor pela Universität Kaiserslautern – Kaiserslautern, Alemanha

Porto Alegre, julho de 2010.

DEDICATÓRIA

Dedico este trabalho aos meus pais por todo o apoio e incentivo para a minha formação pessoal e profissional.

AGRADECIMENTOS

Aos meus pais, meus irmãos e minha família por todo suporte e carinho dedicados para a minha formação.

À minha noiva pelo total apoio, carinho e compreensão durante a minha graduação.

Aos meus colegas da UFRGS e PUCRS que de alguma forma colaboraram com a minha graduação.

Aos colaboradores do laboratório IEE por todo o suporte prestado para o desenvolvimento desse projeto.

Ao meu orientador por me guiar durante o início, desenvolvimento e conclusão desse projeto.

À empresa Softinnovation pelo total apoio para com as minhas atividades acadêmicas.

RESUMO

Esse projeto teve como objetivo desenvolver um *software* para manipulação de um *mouse virtual* para computador controlado através de biosinais dos músculos masseter e temporal de ambas hemifaces do rosto humano. Os biosinais são capturados através de eletrodos não invasivos conectados a um eletromiógrafo. O eletromiógrafo é conectado a um conversor AD da *National Instruments*, que, por sua vez, é conectado via USB em um computador. O *software* desenvolvido coleta os biosinais através da interface USB, divide os biosinais no tempo em pacotes de dados e processa cada pacote para calcular a energia média do mesmo. O valor da energia dos pacotes de dados é comparado a dois limiares de energia calculados através de um processo de calibração feito pelo usuário no momento que o *software* é iniciado. Através do resultado da comparação da energia de um pacote com os limiares de energia que foram calibrados o *software* executa ou não alguma ação com o *mouse virtual*. Para a determinação dos limiares de energia foram desenvolvidas três técnicas, duas técnicas para determinação do limiar de forma adaptativa e uma técnica para determinação do limiar de forma estática. Ao final desse projeto se concluiu que as técnicas adaptativas não possuem diferença considerável entre elas para o funcionamento do *software*. A técnica adaptativa LED (*Linear Energy Based Detector*) apresentou 17% de falhas para detecção e execução de comandos, contra 15% da técnica adaptativa ALED (*Adaptive Linear Energy Based Detector*). Já a técnica que determina o limiar de forma estática apresentou uma taxa de falhas de 26% no momento da detecção de comandos, ou seja, bem superior que a das técnicas adaptativas.

Palavras-chaves: Tecnologia Assistiva. Processamento de Biosinais. Eletromiografia. Instrumentação Biomédica

ABSTRACT

The objective of this project is to show the development of a software that controls a computer virtual mouse through the biosignals of masseter and temporalis muscles. The biosignals are captured using non-invasive electrodes connected in a electromyograph. The electromyograph is connected to an AD converter of National Instruments manufacturer. This AD converter is connected to a computer using an USB interface. The developed software capture the biosignals through the USB interface and breaks the biosignal in data packets. Each single data packet is processed so is possible to calculate the mean energy of each one. The energy value of each packet is compared to two energy thresholds that are calculated using a calibration process that is realized in the moment that the software is started. Using the comparison results between the threshold and the packet energy the software executes or not an action with the virtual mouse. To determine the energy thresholds used in calibration, three techniques were implemented: two techniques are adaptive and the other one is static. At the end of this project it was concluded that there is no considerable difference between the adaptive techniques to the software functions. The LED (Linear Energy Based Detector) adaptive technique showed 17% of errors and the ALED (Adaptive Linear Energy Based Detector) adaptive technique showed 15% of failures in the software commands execution. The static technique showed a failure of 26% in the software functions, that means the adaptive techniques as superior of the static one.

Keywords: Assitive Technology. Biosignals Processing. Electromyography. Biomedical Instrumentation.

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE EQUAÇÕES

LISTA DE ABREVIATURAS

ADC: Conversor Analógico para Digital

AI: *Analogic Input*

AO: *Analogic Output*

ALED: *Adaptive Linear Energy Based Detector*

API: *Application programming interface*

ATM: Articulação Têmporo-Mandibular

CONADE: Conselho Nacional dos Direitos da Pessoa Portadora de Deficiência

CVM: Contração Voluntária Máxima

DAQ: *Data acquisition* (nome usado pela *National Instruments* para identificar suas placas de aquisição)

DELET: Departamento de Engenharia Elétrica

DIO: *Digital Input/Output*

DTA: Dispositivo de Tecnologia Assistiva

EEG: Eletroencefalograma

EMG: Eletromiografia

IEE: Laboratório de Instrumentação Eletro-Eletrônica

IEEE: *Institute of Electrical and Electronics Engineers*

LED: *Linear Energy Based Detector*

OMS: Organização Mundial da Saúde

RMS: *Root Mean Square*

SEDH: Secretaria Especial dos Direitos Humanos

STA: Serviço de Tecnologia Assistiva

TA: Tecnologia Assistiva

UFRGS: Universidade Federal do Rio Grande do Sul

USB: *Universal Serial Bus*

1. INTRODUÇÃO

O objetivo deste trabalho é desenvolver um programa para o processamento de sinais mioelétricos captados por um eletromiógrafo utilizando eletrodos não invasivos, posicionados nos músculos da região do masseter e temporal. A finalidade principal é caracterizar e utilizar esses biosinais na integração homem-máquina simulando e controlando um mouse virtual em um computador.

Além disso, o programa tem as seguintes funções: controlar a interface USB (*Universal Serial Bus*) com o eletromiógrafo utilizado para capturar os biosinais, processar os biosinais, controlar a lógica de decisão e de associação dos sinais processados ao movimento de um mouse virtual de um computador.

O programa foi desenvolvido linguagem de programação C#, utilizando a ferramenta *Microsoft Visual .NET C# 2008 Express Edition*, que é uma versão do *software* Visual Studio disponibilizada gratuitamente pela empresa *Microsoft*.

O trabalho está dividido em quatro etapas:

1. Revisão bibliográfica: onde é apresentado o estudo que foi realizado antes e durante o desenvolvimento desse projeto;
2. Definição e implementação da metodologia experimental: onde são definidos os equipamentos, tecnologias e procedimentos que foram implementados para que seja possível a realização desse trabalho. Além disso, é apresentado todo o desenvolvimento experimental deste trabalho;
3. Resultados alcançados: capítulo que apresenta os resultados e discussões relacionadas a esse projeto;
4. Conclusões: onde é realizado o fechamento e apresentação das conclusões desse trabalho.

Trabalhos nessa área se justificam pelo fato de que aproximadamente 10% da população mundial possui algum tipo de deficiência, segundo a Organização Mundial da Saúde (OMS). A Tecnologia Assistiva tem como objetivo aumentar ou restaurar a função humana podendo proporcionar uma vida independente e produtiva para pessoas portadoras de algum tipo de deficiência. Além disso, este trabalho faz parte da linha de pesquisa de Instrumentação Biomédica/Tecnologia Assistiva do Grupo de Pesquisas do Laboratório de Instrumentação Eletro-Eletrônica (IEE) do Departamento de Engenharia Elétrica (DELET) da Universidade Federal do Rio Grande do Sul (UFRGS), auxiliando o desenvolvimento de plataformas experimentais de interesse do grupo de pesquisa.

2. REVISÃO BIBLIOGRÁFICA

Para facilitar a compreensão deste trabalho é importante apresentar uma breve discussão dos seguintes assuntos:

- tecnologia assistiva;
- anatomia e fisiologia dos músculos masseter e temporal;
- eletromiografia.

2.1 TECNOLOGIA ASSISTIVA

Tecnologia Assistiva (TA) é um termo recente, utilizado para identificar todo o arsenal de recursos e serviços que contribuem para proporcionar ou ampliar habilidades funcionais de pessoas com deficiência e conseqüentemente promover vida independente e inclusão, conforme BERSCH (2008).

Segundo a lei pública 108-364, do Governo dos Estados Unidos da América, de 2004, TA e seus dispositivos podem ser definidos por:

- Tecnologia Assistiva: é a tecnologia utilizada através de um dispositivo de tecnologia assistiva ou um serviço de tecnologia assistiva;
- Dispositivo de Tecnologia Assistiva (DTA): é qualquer item, parte de equipamento ou sistema de um produto, que adquirido comercialmente, modificado ou customizado, cuja função é aumentar, manter ou melhorar as capacidades funcionais de um indivíduo deficiente;
- Serviço de Tecnologia Assistiva (STA): é qualquer serviço que diretamente provê assistência a um indivíduo deficiente na seleção, aquisição, ou uso de um DTA.

De acordo com o *website* da Presidência da República do Brasil, no Brasil existe a Secretaria Especial dos Direitos Humanos (SEDH), que é responsável pelo Conselho

Nacional dos Direitos da Pessoa Portadora de Deficiência (CONADE). Segundo o SEDH o CONADE é um órgão superior de deliberação colegiada criado para acompanhar e avaliar o desenvolvimento de uma política nacional para inclusão da pessoa portadora de deficiência e das políticas setoriais da educação, saúde, trabalho, assistência social, transporte, cultura, turismo, desporto, lazer e política urbana dirigidos a esse grupo social. O CONADE foi criado para que a população brasileira com algum tipo de deficiência ou incapacidade, que representa aproximadamente 14,5% da população brasileira, possa tomar parte do processo de definição, planejamento, e avaliação das políticas destinadas à pessoa com deficiência, por meio da articulação e diálogo com as demais instâncias de controle social e os gestores da administração pública direta e indireta. A partir das obrigações do CONADE, pode-se dizer que esse órgão é capaz de apontar quais as necessidades quanto ao desenvolvimento e implantação de tecnologias assistivas para a população brasileira.

Segundo MELO *et al* (2006) e BERSCH (2008), tecnologias assistivas são recursos e serviços que visam facilitar o desenvolvimento de atividades da vida diária por pessoas com deficiência. Procuram aumentar capacidades funcionais e assim promover a autonomia e a independência de quem as utiliza. Existem tecnologias assistivas para auxiliar na locomoção, no acesso à informação e na comunicação, no controle do ambiente e em diversas atividades do cotidiano como o estudo, o trabalho e o lazer. A tecnologia assistiva pode ser dividida nas seguintes categorias:

- tecnologias assistivas para auxiliar em atividades do dia-a-dia;
- tecnologias assistivas para auxiliar o uso de sistemas baseados em computador;
- comunicação aumentativa e alternativa;
- sistemas de controle de ambiente através de um controle remoto para gerenciar os mais variados aparelhos em uma residência;
- projetos arquitetônicos para acessibilidade;

- próteses artificiais que substituem partes ausentes do corpo;
- órteses colocadas junto a um segmento do corpo para melhor posicionamento, estabilização e/ou função de um determinado membro;
- dispositivos para adequação postural;
- dispositivos para pessoas portadoras de deficiência visual na realização de tarefas do dia a dia;
- dispositivos para pessoas portadoras de deficiência auditiva;
- dispositivos para adaptações em veículos, onde são criados acessórios e realizadas adaptações que possibilitam uma pessoa com deficiência física dirigir um automóvel e facilitadores de embarque e desembarque.

Em se tratando de produtos de Tecnologia Assistiva, a Norma Internacional ISO 9999:2007, "Ajudas Técnicas para Pessoas com Deficiência", apresenta a seguinte classificação dividida em 11 áreas apresentadas na Tabela 2.1.

Tabela 2.1 Classificação de produtos de Tecnologia Assistiva.

Classificação	Descrição
Classe 04	Ajudas para tratamento clínico individual
Classe 05	Ajudas para treino de capacidades
Classe 06	Órteses e próteses
Classe 09	Ajudas para cuidados pessoais e de proteção
Classe 12	Ajudas para mobilidade pessoal
Classe 15	Ajudas para cuidados domésticos
Classe 18	Mobiliário e adaptações para habitação e outros locais
Classe 21	Ajudas para a comunicação, informação e sinalização
Classe 24	Ajudas para o manejo e produtos e mercadorias
Classe 27	Ajudas e equipamentos para melhorar o ambiente, ferramentas e máquinas
Classe 30	Ajudas para a recreação

Fonte: ISO, 9999:2007.

2.2 UMA BREVE DESCRIÇÃO DA ANATOMIA DO MASSETER E DO TEMPORAL

A articulação Têmporo-Mandibular (ATM), que é a relação entre os ossos temporal e mandibular, caracteriza-se por ser peculiar e funcionalmente complexa. É um conjunto de

estruturas anatômicas que, com a participação de grupos musculares especiais, possibilitam à mandíbula executar vários movimentos durante a mastigação, segundo OLIVEIRA (2002).

Como o foco desse trabalho é a utilização de eletrodos nos músculos Masseter e Temporal, apenas eles são descritos nessa breve revisão bibliográfica. De acordo com CERQUEIRA (2005), os músculos Masseter e Temporal tem como função a elevação da mandíbula (abrir e fechar a boca) e, além disso, a porção posterior do Temporal é responsável pela retração da mandíbula. Na Figura 2.1, pode-se observar a localização do Temporal direito e do Masseter direito.

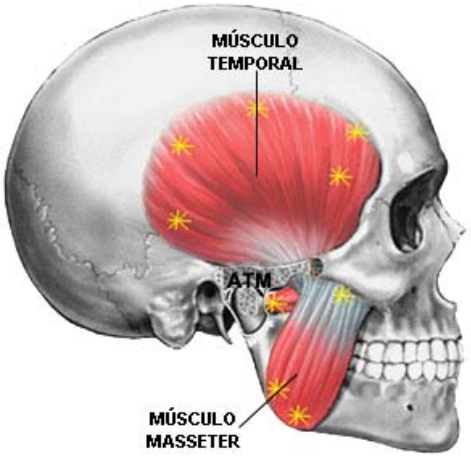

Figura 2.1 Localização dos músculos Masseter e Temporal.
Fonte: http://www.ortodontista.net/, 2010.

2.3 ELETROMIOGRAFIA

O eletromiógrafo (EMG) é o equipamento que capta o sinal mioelétrico resultante dos potenciais de ação das fibras musculares, que ocorrem antes da sua contração, portanto não é uma medida da força muscular. A origem do sinal mioelétrico é o potencial de ação, que é disparado por cada unidade motora ativada durante a contração muscular. Resumidamente, é a soma da atividade de todas as unidades motoras que constituem o sinal mioelétrico, que pode ser captado por eletrodos superficiais (eletrodos não invasivos) colocados na pele, ou por eletrodos invasivos. O estudo da morfologia deste sinal durante determinado movimento

permite avaliar e estudar a atividade muscular, a intensidade e a duração da contração (CHALUB *et al*.,2008).

Após a coleta do sinal mioelétrico, o mesmo necessita ser processado para ser interpretado. Para eliminar determinados artefatos, um eletrodo de referência pode ser utilizado em alguma superfície óssea. Em geral, os dados brutos passam por um processo de retificação, filtragem, e quantificação para descrever a quantidade de energia muscular gasta para a realização de uma determinada ação (CHALUB *et al*., *2008*).

Segundo DE LUCA (1995), existem as seguintes maneiras que são comumente utilizadas para processar o sinal do EMG. No domínio do tempo geralmente é utilizado o valor RMS (*root-mean-squared*) do sinal e a média do valor retificado. Ambos os métodos são apropriados e provêem medições úteis sobre a amplitude do sinal para determinadas aplicações. Para contrações provocadas voluntariamente (chamadas de Contração Voluntária Máxima (CVM)) detectadas pelo EMG, o valor RMS é mais apropriado, porque representa a potência do sinal e isso tem um significado físico claro nesta área.

As principais recomendações para uso da eletromiografia são:

1. Configuração diferencial dos eletrodos
    - superfícies de detecção com duas barras em paralelo: cada uma com 1cm de comprimento, 1 a 2mm de largura e 1 cm de distância;
    - largura de banda de 20 a 500 Hz com *roll-off* de pelo menos 12dB/oitava;
    - taxa de rejeição comum maior que 80dB;
    - ruído menor do que 2μV RMS (20 a 400Hz);
    - impedância de entrada maior que 100MΩ.
2. Posicionar o eletrodo na linha média do ventre muscular, entre a junção miotendínia e a zona de inervação mais próxima, com uma superfície de detecção orientada perpendicularmente ao comprimento das fibras musculares. Pode-se

utilizar estimulação elétrica ou mapeamento de superfície elétrica para localizar as zonas de inervação;

3. Utilizar o valor RMS do sinal para medir a amplitude de um sinal mioelétrico provocado voluntariamente.

3. METODOLOGIA EXPERIMENTAL

A metodologia experimental adotada foi dividida em três áreas: definição do arranjo experimental, onde são listados os itens que estão envolvidos nesse projeto, descrição do sistema de hardware que foi utilizado (eletromiógrafo e conversor AD) e o *software* desenvolvido.

3.1 ARRANJO EXPERIMENTAL PROPOSTO

O arranjo experimental utilizado foi dividido em três blocos, conforme mostrado na Figura 3.1.

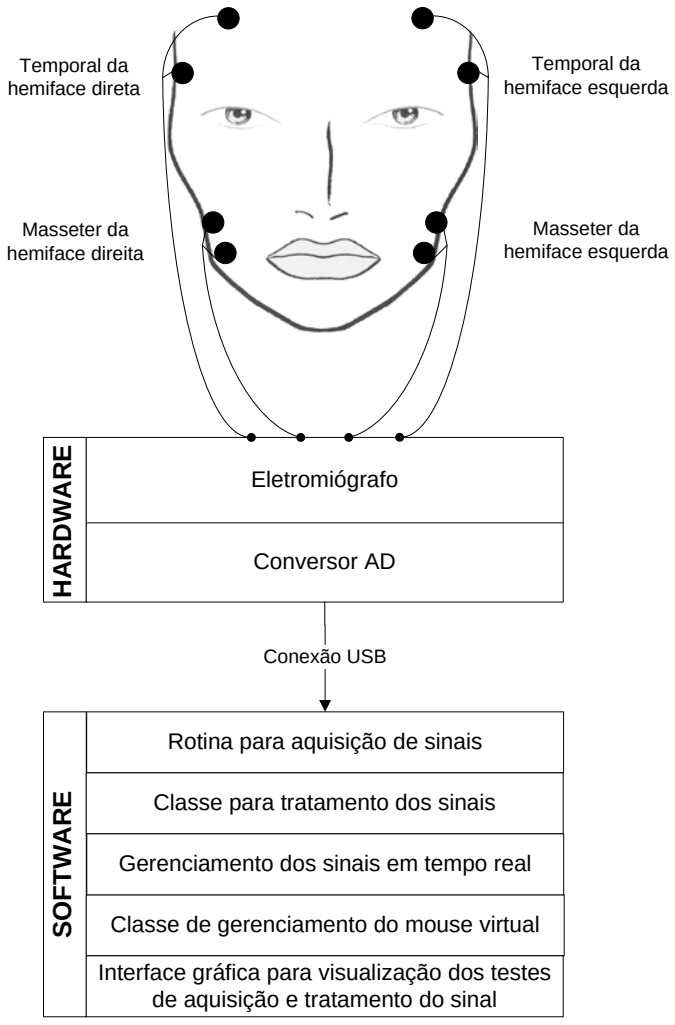

Figura 3.1 Modelo do arranjo experimental.

O primeiro bloco representa a face de um ser humano, onde são posicionados os oito eletrodos (quatro canais) que realizam o casamento de impedância entre a pele e o EMG permitindo a captura dos biosinais de interesse. Os eletrodos são posicionados nos músculos Masseter e Temporal de ambos os lados da face de um voluntário.

O segundo bloco representa a etapa de hardware utilizado neste trabalho. Basicamente é composto pelo eletromiógrafo e pelo conversor AD disponibilizados no laboratório IEE.

O terceiro bloco representa o *software* desenvolvido nesse projeto. O mesmo apresenta a característica modular de forma que cada uma de suas principais funções sejam independentes uma das outras. As principais funções são a coleta do sinal via interface USB, processamento do sinal, controle e decisão para o gerenciamento do mouse virtual implementado. Além disso, possui uma interface para testes de cada uma dessas funções.

Para acelerar o processo de desenvolvimento do trabalho foram realizadas coletas de dados utilizando o *software* LabVIEW, versão 8.2, da *National Instruments*, para permitir testes de processamento do sinal. Foram coletadas 10 amostras, com taxa de amostragem de 1kHz por canal e eletrodos não invasivos para cada uma das situações abaixo:

- sinal de resposta do músculo Masseter da hemiface esquerda para uma mordida com o lado esquerdo da face;
- sinal de resposta do músculo Temporal da hemiface esquerda para uma mordida com o lado esquerdo da face;
- sinal de resposta do músculo Masseter da hemiface direita para uma mordida com o lado direito da face;
- sinal de resposta do músculo Temporal da hemiface direita para uma mordida com o lado direito da face;

- sinal de resposta do músculo Masseter de ambas hemifaces para uma mordida com ambos os lados da boca;
- sinal de resposta do músculo Temporal de ambas hemifaces para uma mordida com ambos os lados da boca.

Na Figura 3.2 é mostrado o diagrama de blocos construído no LabVIEW para efetuar a coleta de dados descrita acima.

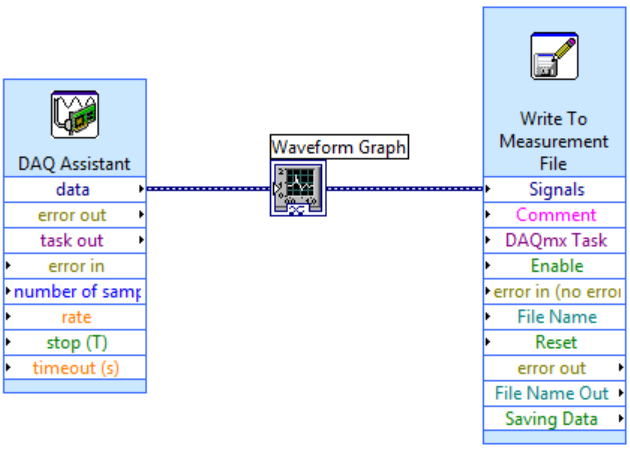

Figura 3.2 Diagrama de blocos criado no LabVIEW para a coleta de dados

3.2 PLATAFORMA DE HARWARE

Nesse tópico são apresentadas as características resumidas do eletromiógrafo e eletrodos utilizados, assim como, o modelo do conversor AD utilizado.

O eletromiógrafo utilizado foi desenvolvido no laboratório de Instrumentação Eletro-Eletrônica (IEE) para as pesquisas na área de Instrumentação Biomédica e Tecnologia Assistiva. As características básicas são:

- Ganho diferencial de 1000;
- Filtros *Butterworth*:
    - dois filtros passa-baixa de segunda ordem em cascata com freqüência de corte em 800 Hz;

- dois filtros passa-altas de segunda ordem em cascata com freqüência de corte em 20 Hz;

- Quantidade de canais: 8.

A Figura 3.3 mostra uma foto do eletromiógrafo utilizado. Para maiores detalhes consultar o Grupo de Pesquisa do IEE.

Figura 3.3 Foto do eletromiógrafo do IEE.

Os eletrodos utilizados são descartáveis do tipo não invasivos, modelo “Meditrace 200”, da marca Kendal. Na Figura 3.4 é mostrada uma foto de um eletrodo do modelo utilizado para esse projeto.

Figura 3.4 Foto do eletrodo utilizado.
Fonte: http://www.kendall-ltp.com/, 2010.

O conversor AD utilizado foi o DAQ (*Data acquisition*) modelo USB-6008 da *National Instruments* que possui 8 entradas analógicas (AI), dois canais analógicos de saída (AO), 12 canais de entrada/saída digital (DIO) e um contador de 32 bits com interface USB. Para esse trabalho foram utilizadas as entradas analógicas e a interface USB. As entradas analógicas do USB-6008 possuem uma resolução de 12 bits, com 11 bits *single-ended*, taxa de amostragem de até 10kS/s tanto para um único canal, como para múltiplos canais. A Figura 3.5 mostra uma foto da DAQ USB-6008 utilizada.

Figura 3.5 Foto do DAQ USB-6008 da *National Instruments*.
Fonte: http://sine.ni.com/, 2010.

3.3 PLATAFORMA DE *SOFTWARE*

A Figura 3.6 mostra as etapas de funcionamento do *software* desenvolvido. Abaixo encontra-se uma sucinta descrição de cada bloco:

1. Primeiramente o *software* recebe os dados através da interface USB. Essa aquisição ocorre de maneira contínua a partir do momento em que o *software* é inicializado até que o mesmo seja finalizado. A forma de aquisição desses dados é descrita na seção 3.3.3;

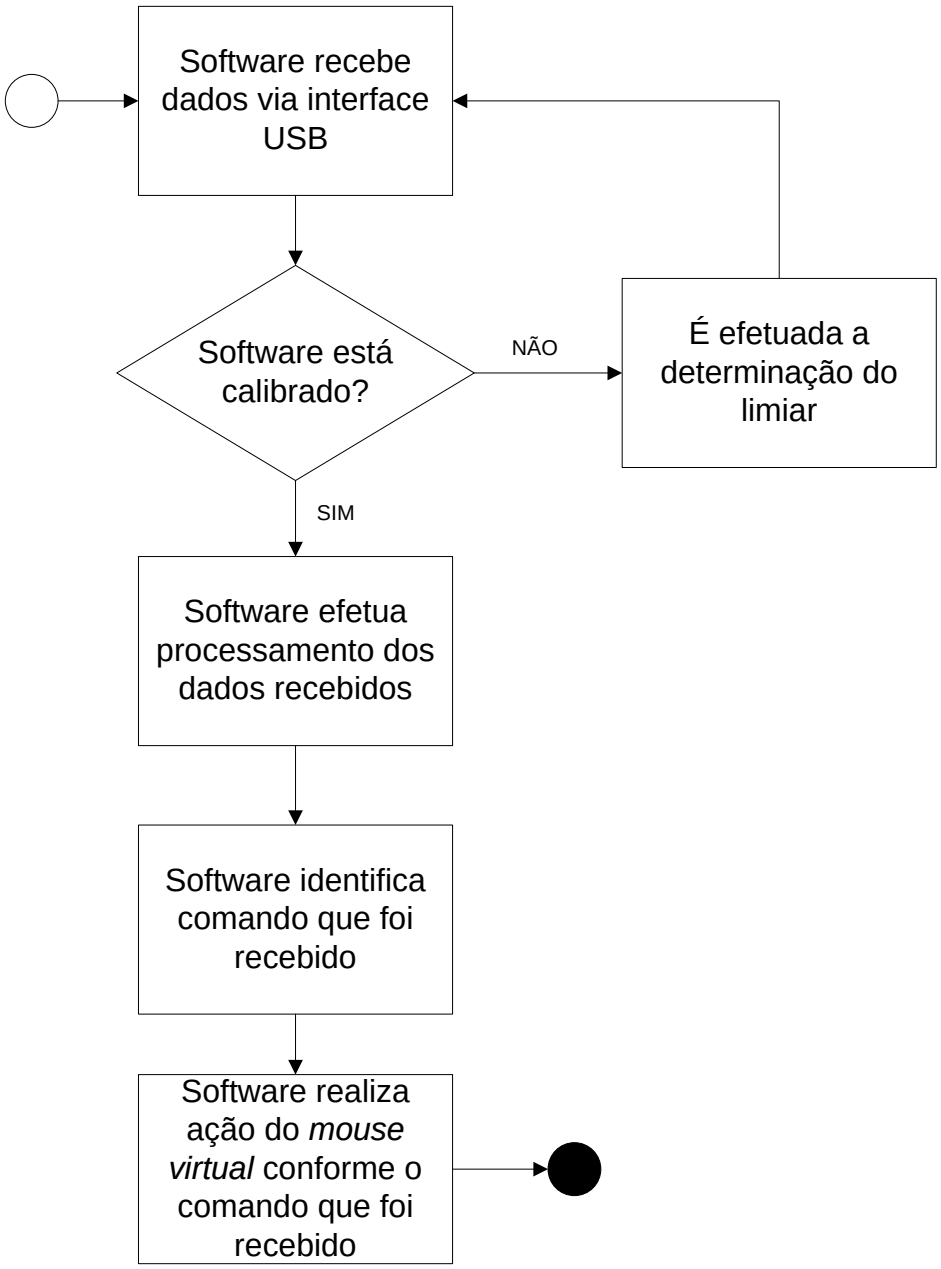

Figura 3.6 Fluxograma de funcionamento do *software*.

2. Toda a vez que o *software* é inicializado é realizada uma calibração do sistema. Essa calibração é necessária para que o *software* seja capaz de interpretar corretamente os biosinais que estão sendo capturados, pois cada pessoa possui características diferentes, portanto os sinais variam de pessoa para pessoa. Para a calibração é calculada a energia dos sinais capturados e são determinados limiares (detalhados na seção 3.3.1.6) que são utilizados para a ativação ou não

de um comando no *software*, além da identificação do comando que está sendo requisitado pelo usuário. Detalhes sobre o processo de calibração são descritos na seção 3.3.4;

3. Após a calibração do *software* é iniciado o processamento dos sinais capturados que é realizado através de pacotes de dados, ou seja, o sinal que é recebido continuamente é dividido em partes iguais no tempo e cada uma dessas partes é analisada individualmente pelo *software*. As técnicas de processamento de sinais são explicadas na seção 3.3.1;
4. Com o sinal processado é possível identificar um comando a ser realizado pelo *mouse virtual*. Essa identificação é feita através da energia do sinal dos pacotes que estão sendo analisados. Essa energia é comparada aos limiares que foram determinados durante o processo de calibração. Caso a energia do pacote seja maior do que a de calibração significa que o *software* deverá executar uma ação;
5. A realização de uma ação de um comando se dará após a identificação do comando a ser executado. Para isso é comparada a energia de cada um dos canais capturados com os limiares calibrados. Detalhes sobre essas ações e comandos do *mouse virtual* estão descritos no item 3.3.2.

3.3.1 TÉCNICAS DE PROCESSAMENTO DOS SINAIS

Para o funcionamento do *software* a aquisição de dados é continua, portanto, para que o sinal possa ser processado ele foi dividido no tempo em pacotes de dados. Cada pacote dados possui um tamanho fixo no tempo e é processado individualmente. Como são utilizados quatro canais de aquisição de dados o *software* efetua a divisão de pacotes e o processamento dos dados de cada um dos canais individualmente. Através da análise de cada pacote de cada

canal é possível coletar as informações necessárias para o funcionamento do *software*. Na Figura 3.7 é possível visualizar como é a divisão em pacotes de um sinal mioelétrico em pacotes.

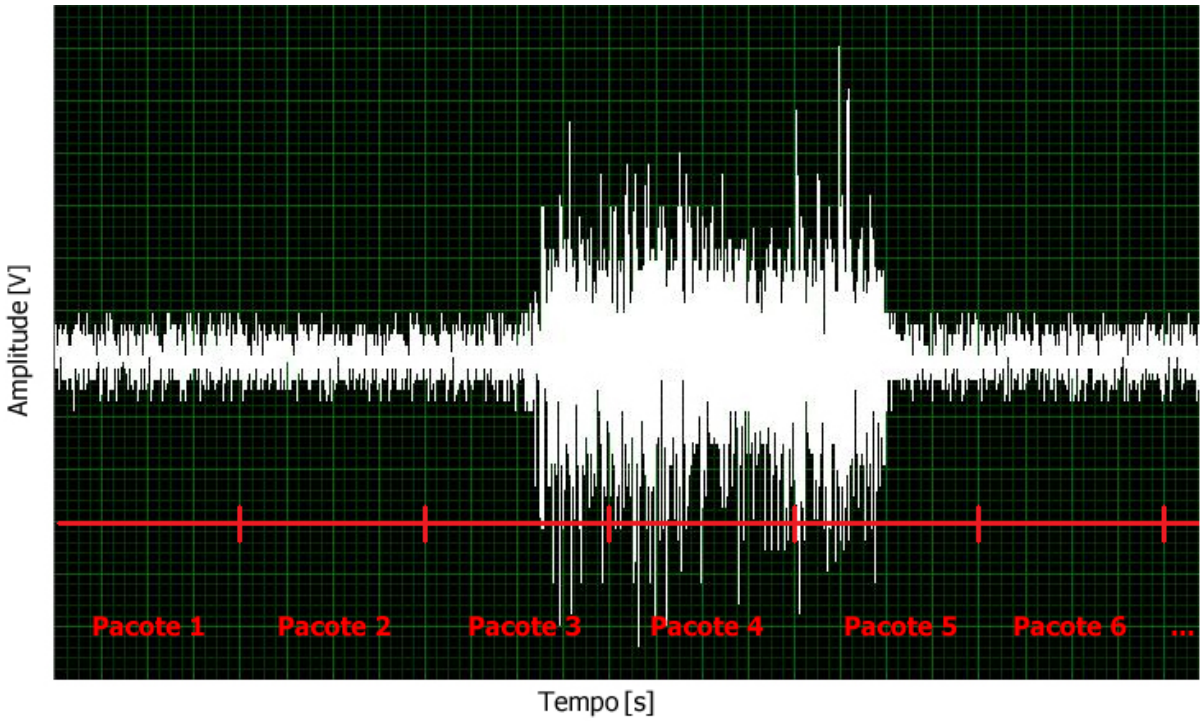

Figura 3.7 Divisão de pacotes de um sinal mioelétrico.

Para o desenvolvimento desse trabalho foi necessário uma revisão de técnicas de processamento de sinais. Foram estudados alguns procedimentos de análise do sinal para um pacote: determinação da quantidade de amostras coletadas, valor médio das amostras, variância das amostras, desvio padrão das amostras, energia média das amostras e técnicas de determinação de um limiar que usam como base de cálculo a energia do sinal. Foi implementada uma classe em C# chamada *IEESignalFunctions* que armazena os dados de um sinal e executa as funções de processamento de um pacote de dados para cada um dos itens citados acima.

A seguir são demonstradas em detalhe as técnicas de processamento de sinais que foram implementadas nesse trabalho.

3.3.1.1 DETERMINAÇÃO DA QUANTIDADE DE AMOSTRAS POR PACOTE

A Equação 3.1 apresenta a expressão utilizada para a determinação do número de amostras de pacote. Essa equação foi deduzida a partir do teorema da amostragem demonstrado no capítulo 3 do livro de SMITH, 1997. Através do número exato de amostras de um pacote é possível que o *software* desenvolvido efetue o processamento do sinal contido no mesmo sem que ocorra perda de informações.

$$N = \frac{t_{pacote}}{\frac{1}{f_{amostragem}}} \qquad (3.1)$$

Onde:

- N é o número de amostras de um pacote;
- t_{pacote} é o tamanho do pacote em segundos;
- $f_{amostragem}$ é a freqüência de amostragem utilizada pelo sistema.

Esse cálculo é feito pela função *CalculateNumberOfSamples()* da classe *IEESignalFunctions*. Além disso, o número de amostras de um pacote pode ser ajustado manualmente através da função *setNumberOfSamples(int nos)*, onde o parâmetro "nos" é o número de amostras desejado. Para obter o número de amostras de um pacote que foi calculado foi desenvolvida a função *getNumberOfSamples()*. Nos anexos desse trabalho encontram-se todas as funções desenvolvidas.

3.3.1.2 VALOR MÉDIO DAS AMOSTRAS

Para cada pacote é calculado o valor médio das amostras contidas no pacote sob análise. Esse dado é importante para o cálculo da variância das amostras (mostrado na seção

3.3.2.3). Na Equação 3.2 é mostrada a equação para o cálculo do valor médio das amostras de um pacote.

$$\bar{x} = \frac{\sum_{i=0}^{N-1}(x_i)}{N} \quad (3.2)$$

Onde:

- $\bar{x}$ é o valor médio das amostras;
- x_i é o valor da amostra do pacote;
- N é o número de amostras de um pacote.

O cálculo do valor médio das amostras de um pacote é feito pela função *CalculatePacketAverage()*da classe *IEESignalFunctions*. Para obter esse valor a partir da classe calculada foi implementada a função *getPacketAverage()*.

3.3.1.3 VARIÂNCIA DAS AMOSTRAS

A Equação 3.3 mostra a equação para o cálculo da variância. A variância é utilizada como passo de adaptação na técnica ALED (*Adaptive Linear Energy Detector*), detalhada no tópico 3.3.1.6.3, técnica que é utilizada para determinar o limiar de energia de um pacote. A variância é utilizada para detectar mudanças na característica de um sinal não linear, que é o caso de um sinal mioelétrico (STAUDE, 1995).

$$Var^2 = \frac{\left(x_0 - \bar{x}\right)^2 + \left(x_1 - \bar{x}\right)^2 + \left(x_2 - \bar{x}\right)^2 + ... + \left(x_{N-1} - \bar{x}\right)^2}{N} = \frac{\sum_{i=1}^{N}\left(x_i - \bar{x}\right)^2}{N-1} \quad (3.3)$$

Onde:

- Var^2 é o valor da variância de um pacote de dados;
- $\bar{x}$ é o valor médio das amostras;
- x_i é o valor da amostra do pacote;
- N é o número de amostras de um pacote.

O cálculo da variância de um pacote é feito pela função *CalculePacketVariance()*da classe *IEESignalFunctions*. Para obter esse valor a partir da classe calculada foi implementada a função *getPacketVariance().*

3.3.1.4 DESVIO PADRÃO

O desvio padrão foi implementado nesse projeto caso fosse necessário avaliar mudanças bruscas de energia dentro de um mesmo pacote, mas essa função não foi utilizada no *software*. A Equação 3.4 mostra o cálculo do desvio padrão.

$$\sigma = \sqrt{Var^2} \qquad (3.4)$$

Onde:

- σ é o valor do desvio padrão de um pacote de dados;
- Var^2 é o valor da variância de um pacote de dados.

O desvio padrão é calculado pela função *CalculePacketStandardDeviation()* da classe *IEESignalFunctions*. Para obter esse valor a partir da classe calculada foi implementada a função *getPacketStandardDeviation().*

3.3.1.5 ENERGIA MÉDIA DE UM PACOTE

Para determinar se um pacote deve ou não ser utilizado para que o mouse execute alguma ação é calculada a energia média do pacote. A energia é matematicamente definida como a multiplicação da potência no tempo (PAL, 2010). A Equação 3.5 mostra como é realizado esse cálculo.

$$E_m = \frac{1}{N}\sum_{0}^{N-1}\left(E_{amostra}\right)^2 \qquad (3.5)$$

Onde:

- E_m é o valor da energia média de um pacote;
- $E_{amostra}$ é o valor da energia de uma amostra;
- N é o número de amostras de um pacote.

A energia média é calculado pela função *CalculatePacketEnergy()* da classe *IEESignalFunctions*. Para obter esse valor a partir da classe calculada foi implementada a função *getPacketEnergy()*.

3.3.1.6 TÉCNICAS DE DETERMINAÇÃO DO LIMIAR BASEADAS NA ENERGIA

O uso de técnicas de determinação de um limiar baseado na energia é necessário para o *software*, pois é com base nesse limiar e na energia de um dos pacotes de dados dos quatro canais que o *software* irá ou não executar alguma ação. Se o valor da energia dos pacotes for maior do que o valor do limiar de energia estabelecido no momento da calibração do *software*, então é executada uma ação (a relação entre os canais que são utilizados para cada ação que o *software* irá executar estão descritas no item 3.3.3).

Para fins de pesquisa foram implementadas três técnicas de determinação de um limiar baseado na energia. Uma técnica implementada é uma simples determinação de um limiar estático, com base na energia média do ruído médio do sistema e as outras duas técnicas são técnicas adaptativas. A seguir são descritas em detalhes as técnicas implementadas, as informações sobre essas técnicas foram retiradas dos artigos de PRASAD (2002), SANGWAN (2002), RENEVEY (2001) e TANYER (2000).

3.3.1.6.1 DETERMINAÇÃO DE UM LIMIAR DE FORMA ESTÁTICA

A determinação de um limiar estático baseado na energia consiste basicamente em determinar o ruído do sistema e aplicar uma margem de segurança sobre o valor do ruído. Esta margem de segurança é determinada manualmente no sistema. A Equação 3.6 mostra como esse cálculo é efetuado.

$$L_{estático} > \mathrm{k} \cdot E_{ruído} \qquad (3.6)$$

Onde:

- $L_{estático}$ é o valor do limiar de energia estático;
- $E_{ruído}$ é o valor médio da energia do ruído do sistema;
- *k* é uma margem de segurança para que o sistema não fique instável. Esse valor deve ser maior do que 1.

A Figura 3.8 ilustra um pico de energia acima de um limiar, esse pico deve ser detectado pelo *software* para efetuar alguma ação.

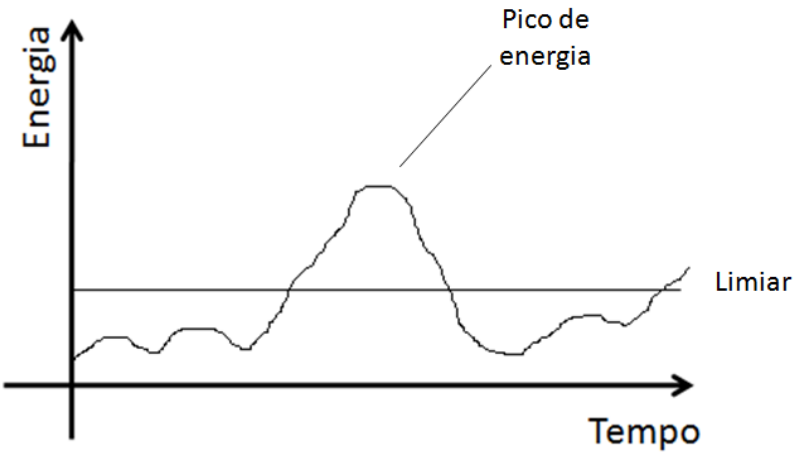

Figura 3.8 Exemplo de um limiar de energia estático.

O limiar estático é calculado pela função *setStaticThreshold(double noise, double margin)* da classe *IEESignalFunctions*, essa função recebe como parâmetros o valor da energia do ruído e a margem de segurança para efetuar os cálculos. A Figura 3.9 mostra o como essa função foi implementada. Para obter esse valor a partir da classe calculada foi implementada a função *getStaticThreshold()*.

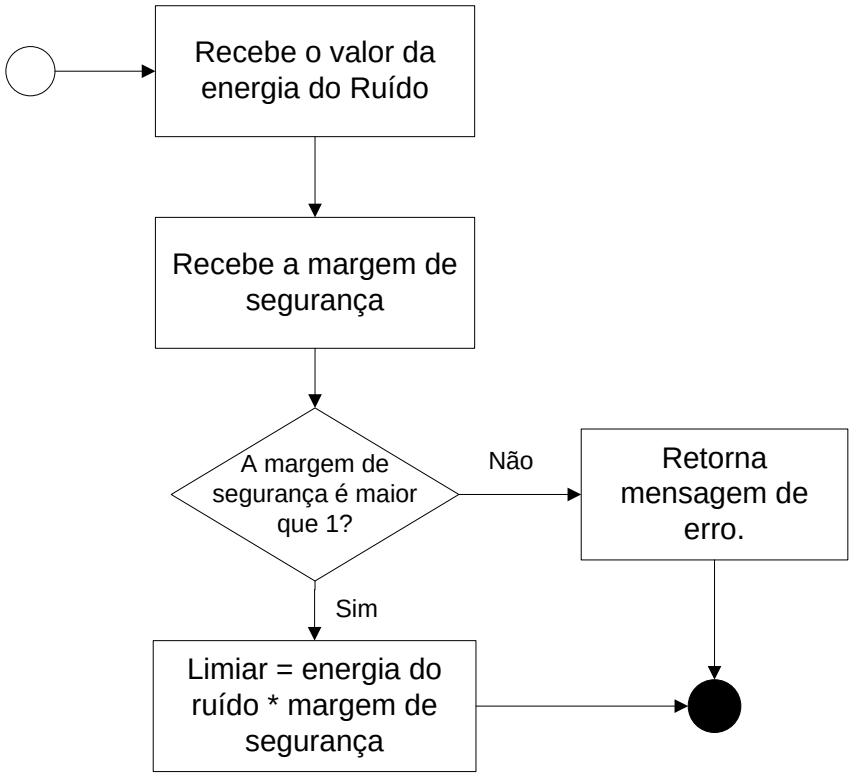

Figura 3.9 Determinação do limiar estático.

3.3.1.6.2 LED – *Linear Energy Based Detector*

A técnica LED é apresentada na Equação 3.7. Essa técnica adaptativa é utilizada para que seja determinado o limiar de energia de um pacote. Para isso é necessária a análise de pelo menos dois pacotes de dados.

$$E_{LED} = (1 - \mathrm{p}) \cdot E_{anterior} + \mathrm{p} \cdot E_{atual} \qquad (3.7)$$

Onde:

- L_{LED} é o valor do limiar de energia calculado;
- $E_{anterior}$ é o valor da energia do último pacote analisado;
- E_{atual} é o valor da energia do pacote que está sendo analisado;
- *p* é o índice que determina o passo de adaptação, podendo variar de 0 a 1. Nesta técnica o parâmetro p é determinado manualmente no momento da configuração do *software*.

O limiar LED é calculado pela função *ApplyLEDTechnique(double p)* da classe *IEESignalFunctions*, essa função recebe como parâmetros o valor do passo de adaptação. A Figura 3.10 mostra o como essa função foi implementada. Para obter esse valor a partir da classe calculada foi implementada a função *getLEDThreshold ()*.

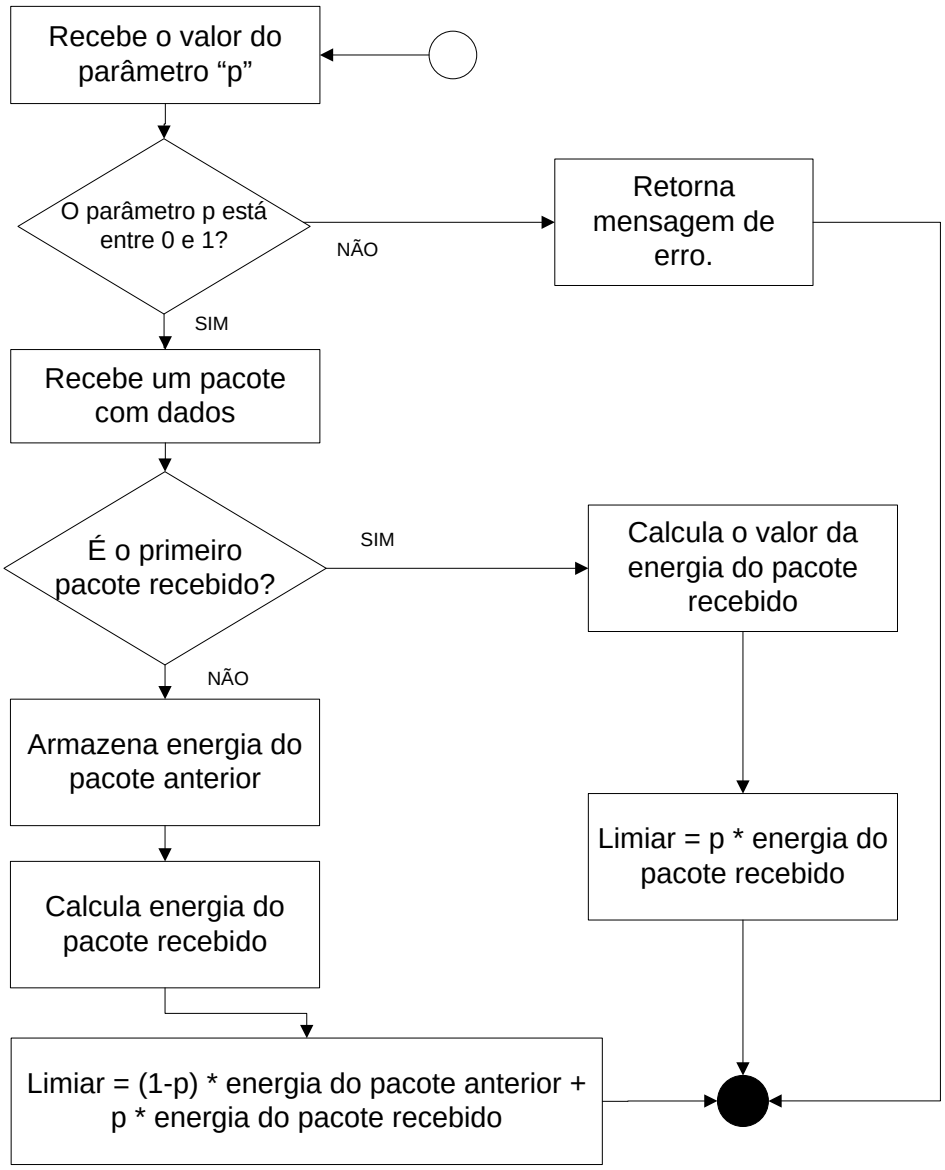

Figura 3.10 Determinação do limiar utilizando a técnica LED.

3.3.1.6.3 ALED – *Adaptive Linear Energy Based Detector*

O ALED é uma técnica que utiliza o mesmo algoritmo da técnica LED, exceto pela determinação do passo de adaptação (índice *p*). Ao invés do passo de adaptação (*p*) ser determinado por uma configuração manual no sistema, ele é determinado através do resultado da comparação da razão da variância da energia do pacote de dados sob análise (Var_{NEW}) com

a variância da energia do último pacote processado (σ_{OLD}), conforme valores apresentados na Tabela 3.1.

Tabela 3.1 Determinação do passo de adaptação da técnica ALED.

Classificação	Valor de *p*
$\frac{Var_{NEW}}{Var_{OLD}} \geq 1{,}25$	0,25
$1{,}25 \geq \frac{Var_{NEW}}{Var_{OLD}} \geq 1{,}10$	0,20
$1{,}10 \geq \frac{Var_{NEW}}{Var_{OLD}} \geq 1{,}00$	0,15
$1{,}00 \geq \frac{Var_{NEW}}{Var_{OLD}}$	0,10

Fonte: SANGWAN, 2002.

O limiar ALED é calculado pela função *ApplyALEDTechnique()* da classe *IEESignalFunctions*, essa função recebe como parâmetros o valor do passo de adaptação. A Figura 3.11 mostra o como essa função foi implementada. Para obter esse valor a partir da classe calculada foi implementada a função *getALEDThreshold ()*.

3.3.1.7 CLASSE *IEESignalFunctions*

No Anexo A desse trabalho é demonstrada toda a implementação da classe *IEESignalFunctions*, que é a classe responsável pelo processamento de sinais de um pacote.

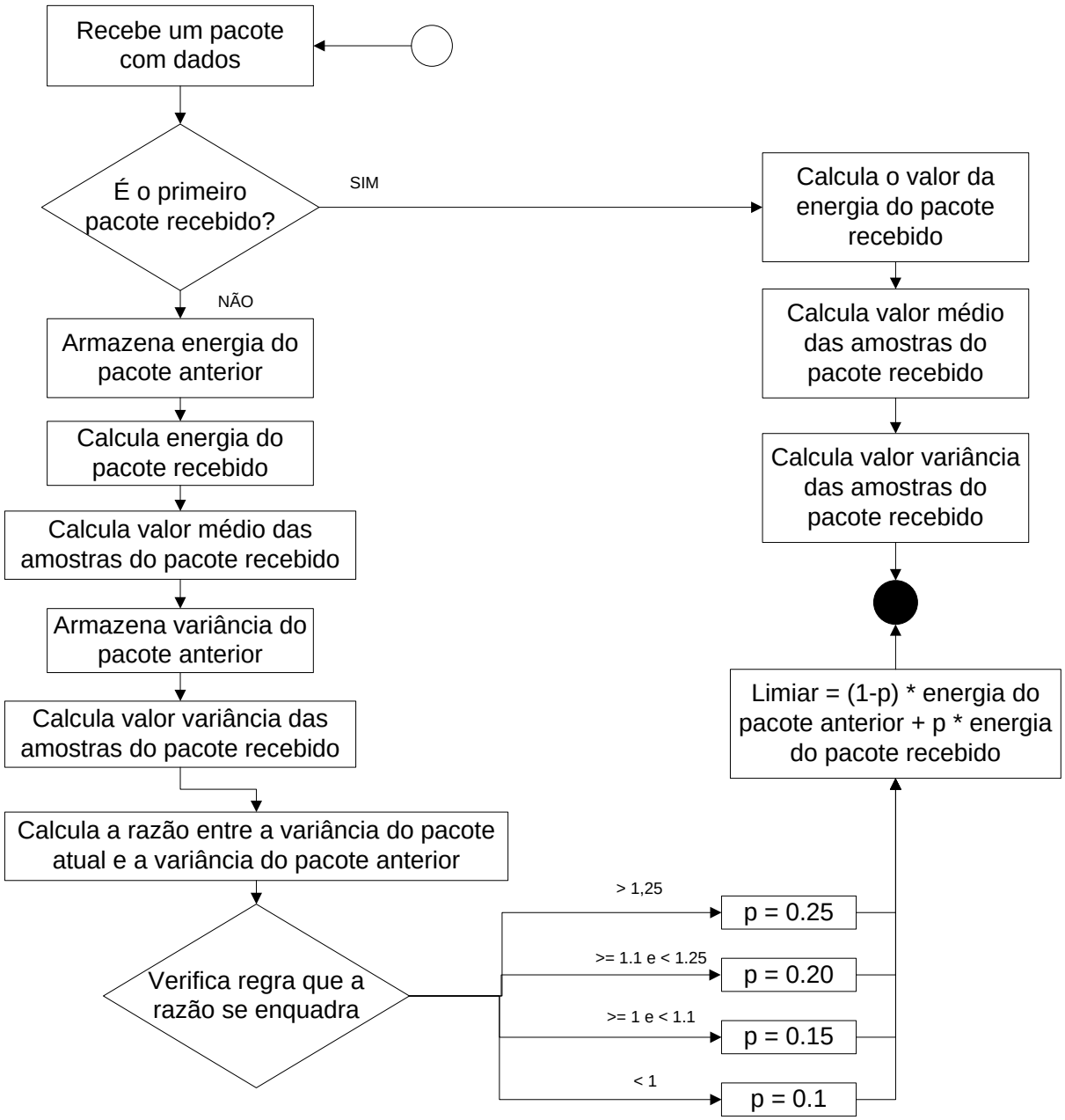

Figura 3.11 Determinação do limiar utilizando a técnica ALED.

3.3.1.8 PROGRAMA TESTE DESENVOLVIDO

Após o desenvolvimento da classe *IEESignalFunctions* foi desenvolvido um programa teste para verificar se a classe estava efetivamente funcionando, visto que seria muito difícil efetuar os testes com a classe com o *software* rodando em tempo real. Além disso esse programa teste serve como um exemplo de uso da classe.

O programa teste dessa classe foi desenvolvido em duas fases, a primeira fase consistiu em testar a classe utilizando apenas um pacote de dados com os dados desse pacote gerados aleatoriamente pelo sistema. Na Figura 3.12 é possível visualizar o programa teste desenvolvido na primeira fase.

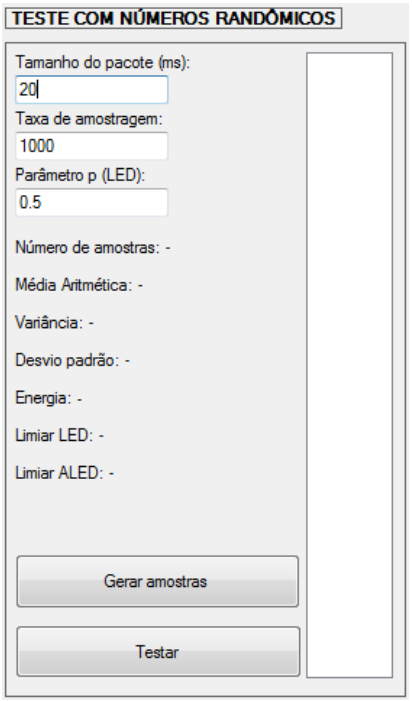

Figura 3.12 Programa teste para um pacote de dados com conteúdo aleatório.

Após constatar que a classe estava funcionando para o cálculo do número de amostras de um pacote, média, variância, desvio padrão e a determinação do limiar de energia com três técnicas implementadas, foi desenvolvida a segunda fase do programa. A segunda fase consistiu em utilizar a classe utilizando dados de biosinais reais dos músculos masseter e temporal, fazendo a divisão do sinal em pacotes de dados. Para isso, foram utilizados arquivos gerados pelo *LabVIEW* 8.2 com os dados coletados conforme descrito no item 3.1 desse projeto. Para utilização desses arquivos foi implementada uma função capaz de efetuar o *parser* dos dados contidos no arquivo. A Figura 3.13 mostra a interface do programa teste desenvolvido na segunda fase dos testes dessa classe.

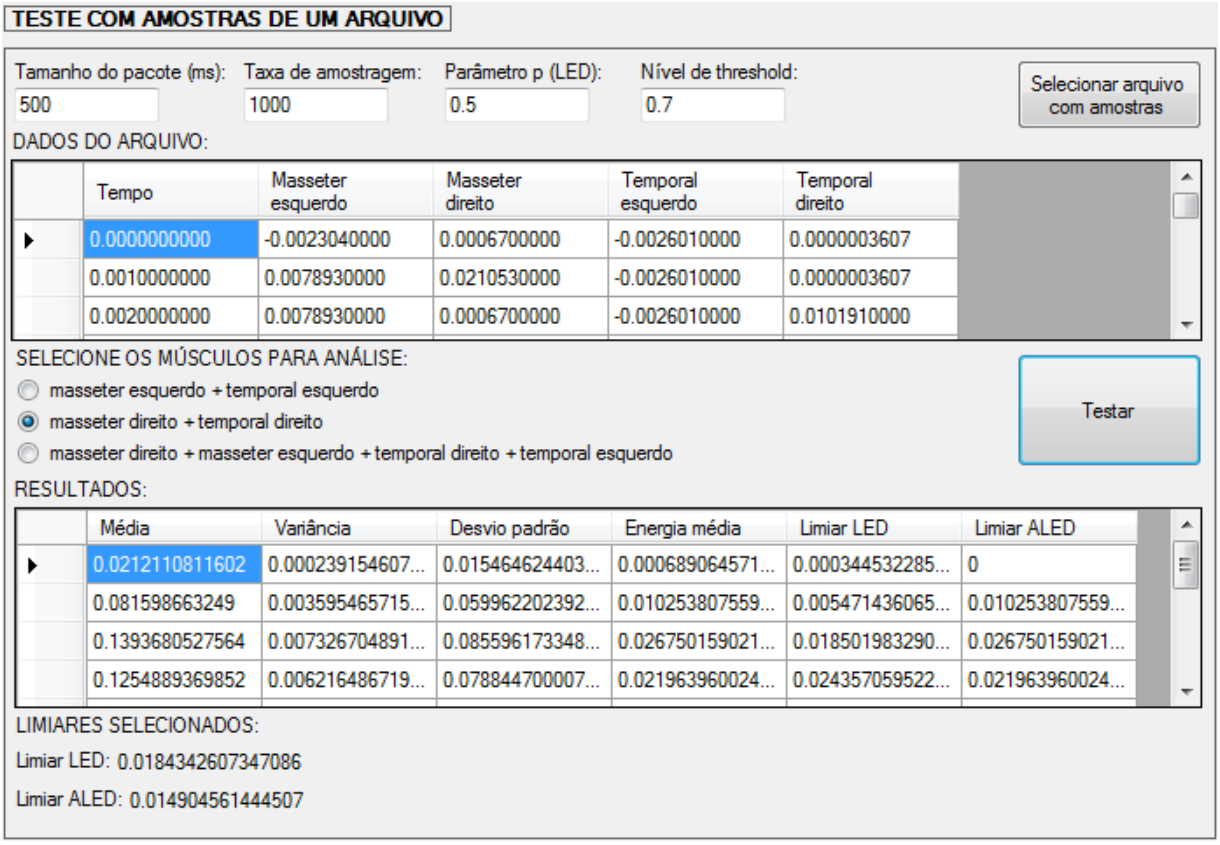

Figura 3.13 Programa teste para um arquivo com dados de biosinais.

Após ser constatado que a classe *IEESignalFunctions* funcionou com dados de biosinais dos arquivos de teste, então essa classe foi utilizada em conjunto com a rotina de aquisição de dados em tempo real.

3.3.2 ROTINA DE DECISÃO E CONTROLE

Na rotina implementada o usuário é responsável pelo processo de seleção de uma ação a ser realizada pelo *mouse* virtual. A interação do usuário com o *software* é realizada por três comandos, cabe ao *software* identificar se o comando foi enviado pelo usuário através das técnicas de Processamento de Sinais descritas no item 3.3.1 desse trabalho. Abaixo são descritos os três comandos:

- Comando 1: contração dos músculos Masseter e Temporal direito e esquerdo simultaneamente.
- Comando 2: contração dos músculos Masseter e Temporal direito;
- Comando 3: contração dos músculos Masseter e Temporal esquerdo.

Para que esses comandos sejam identificados corretamente pelo *software* será necessário que o usuário efetue uma calibração do *software* toda vez que o *software* for iniciado. Para o processo de calibração o usuário deverá efetuar as duas ações listados abaixo:

- Primeira ação: contração dos músculos Masseter e Temporal esquerdo;
- Segunda ação: contração dos músculos Masseter e Temporal direito.

Após a coleta dos dados de cada hemiface é possível determinar um limiar de energia para a hemiface esquerda (músculos Masseter e Temporal esquerdo) e um limiar de energia para a hemiface direita (músculos Masseter e Temporal direito). Através dos limiares calculados no momento da calibração o *software* é capaz de identificar quando é feita uma contração na hemiface esquerda, na hemiface direita e em ambas hemifaces.

Abaixo são listadas as ações realizadas por cada um dos comandos:

- Ação do comando 1: tem a função de seleção da ação que será emulada pelo *mouse* virtual. A Figura 3.14 mostra como é o processo de seleção de uma ação;

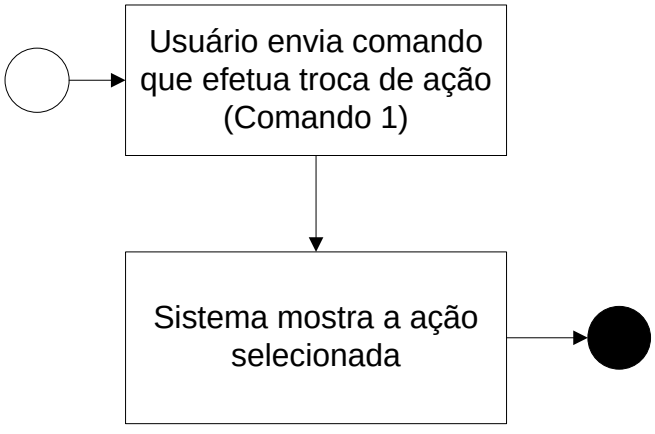

Figura 3.14 Rotina de seleção de uma ação a ser executada pelo *mouse virtual*.

- Ação do comando 2: tem a função de executar uma ação que foi selecionada no sistema através do Comando 1. Por exemplo, se a ação selecionada pelo comando um for “mover o mouse para direita”, então quando o comando 2 for detectado pelo *software* o *mouse* virtual irá se movimentar para a direita;
- Ação do comando 3: tem a função de emular a ação de clique com o botão esquerdo do mouse. A ação de clique com botão esquerdo terá um comando exclusivo por ser considerado, pelo autor desse trabalho, a ação mais utilizada de um *mouse* de computador.

Abaixo são listadas as ações que foram implementadas para o *mouse* virtual através da classe *IEEMouseFunctions*:

- Movimento para a esquerda. Realizada pela função *MoveMouseLeft* da classe;
- Movimento para a direita. Realizada pela função *MoveMouseRight* da classe;
- Movimento para cima. Realizada pela função *MoveMouseUp* da classe;
- Movimento para baixo. Realizada pela função *MoveMouseDown* da classe;
- Clique com o botão esquerdo. Realizada pela função *MouseLeftClick* da classe;
- Duplo clique com o botão esquerdo. Realizada pela função *MouseDoubleLeftClick* da classe;
- Clique com o botão direito. Realizada pela função *MouseRightClick* da classe;

Em cada função que executa um movimento (cima, baixo, direita e esquerda) é possível passar por parâmetro da função a quantidade de *pixels* desse movimento e o tempo, em milissegundos, que o mouse virtual irá levar para se mover de um *pixel* para outro.

Para implementação dos três comandos, foi desenvolvida a classe *IEEMouseLogic*, que é uma classe que herda todas as funções da classe *IEEMouseFunctions* e, além disso, funciona como uma camada que executa os comandos do *mouse* virtual quando o usuário

enviar algum comando para o *software*. A classe *IEEMouseLogic* possui os seguintes funções para executar os comandos do *mouse* virtual:

- Define quantos pixels o mouse irá se mover em cada movimento e o tempo de movimento pixel a pixel através da função DefineMouseParameters;
- Define a ordem de troca de ações do mouse virtual através da função SelectMouseGenericAction. Essa função é utilizada quando o *software* identifica que o comando 1 (contração de ambas hemifaces) é enviado pelo usuário. Abaixo a ordem implementada:
    1. Mover *mouse* virtual para esquerda;
    2. Mover *mouse* virtual para direita;
    3. Mover *mouse* virtual para cima;
    4. Mover *mouse* virtual para baixo;
    5. *Mouse* virtual efetuar clique com botão esquerdo;
    6. *Mouse* virtual efetuar duplo clique com botão esquerdo;
    7. *Mouse* virtual efetuar clique com botão direito. Após essa última, retorna para a primeira ação;
- Executa a função selecionada pelo comando 1, ou seja, essa função de controle é utilizada quando o comando 2 (contração da hemiface esquerda) é detectado pelo *software* através da função *ExecuteGenericAction*;
- Executa a função principal do *mouse* virtual, ou seja, essa função de controle é utilizada quando o comando 3 (contração da hemiface direita) é detectado pelo *software* através da função *ExecuteMainAction*.

3.3.2.1 CLASSES DESENVOLVIDAS

No Anexo B desse trabalho é apresentada toda a classe desenvolvida *IEEMouseFunctions* que é responsável pelas ações do *mouse* virtual. No Anexo C é descrita toda a classe *IEEMouseLogic* que é responsável pela lógica de comandos do *mouse virtual*.

3.3.2.2 PROGRAMA TESTE DESENVOLVIDO

Ao finalizar o desenvolvimento das classes *IEEMouseFunctions* e *IEEMouseLogic* foi desenvolvido um programa teste para testar todas as funções de cada uma dessas classes. A interface desse programa é mostrada na Figura 3.15. Através desse programa teste foi possível constatar que ambas as classes funcionaram e que poderiam ser implementadas em conjunto com a as técnicas de processamento de sinais e com a rotina de aquisição de dados em tempo real.

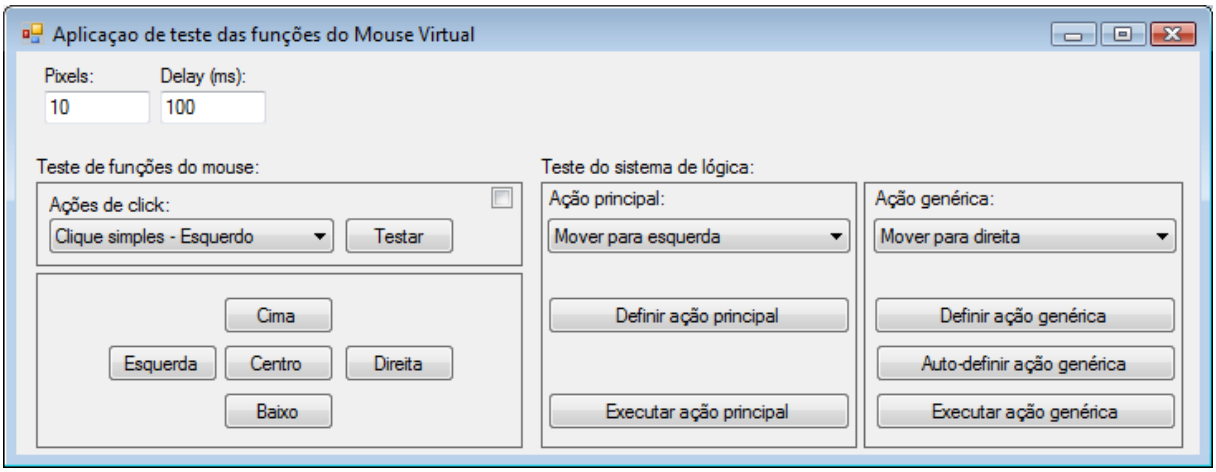

Figura 3.15 Programa teste das classes *IEEMouseFunctions* e *IEEMouseLogic*.

3.3.3 ROTINA DE AQUISIÇÃO DE DADOS EM TEMPO REAL

Para a aquisição de dados em tempo real foi utilizada uma API (*Application programming interface*) para acessar o DAQ USB-6008. Essa API chamada *Measurement Studio 2009 for Visual Studio* é disponibilizada no site da *National Instruments* para *download*. Ela possui uma série de classes que são disponiblizadas para os desenvolvedores em Microsoft Visual C#, Visual C++, ASP.NET e Visual Basic .NET que possuem integração com os hardwares da *National Instruments* (http://www.ni.com/mstudio/).

Esse projeto utilizou duas classes do *Measurement Studio 2009*. A classe *Task* é responsável por criar canais virtuais para a leitura da tensão que é capturada pelo DAQ USB 2008. Foram criados quatro canais virtuais para a leitura da tensão diferencial dos eletrodos localizados nos músculos masseteres e temporais esquerdo e direito, sendo assim possível coletar os *stream* de dados fornecidos pelo DAQ.

Para leitura dos dados dos quatro canais simultaneamente foi necessária a utilização da classe *AnalogMultiChannelReader*. Através dessa classe foi possível efetuar a leitura dos dados dos quatro canais virtuais criados para acessar o DAQ. O *software* envia um comando solicitando a quantidade de amostras desejadas para um dos canais desejados do DAQ e então o DAQ fornece essa informação. Uma limitação dessa classe é a possibilidade de ler no máximo 256 amostras a cada coleta de dados em cada canal. Então, dependendo do tamanho do pacote de dados do *software* e da taxa de amostragem utilizada pelo sistema, as amostras de cada canal são armazenadas em um *buffer* relativo ao canal até que se tenha o número de amostras suficiente para efetuar o processamento dos dados de um pacote.

A rotina de coleta de dados em tempo real utiliza um *timer*, que, a cada 250ms, coleta 250 amostras de cada canal do DAQ e armazena os dados no *buffer* de cada canal. O tempo de 250ms foi estabelecido, pois a taxa de amostragem do eletromiógrafo utilizado nesse

projeto é de 1000 amostras por segundo e a classe utilizada para coleta de dados permite que sejam coletadas no máximo 256 amostras por canal.

Apenas quando o *buffer* de cada um dos quatro canais contém o número de amostras correspondentes ao número de amostras do tamanho de pacote que foi configurado, então é realizado processamento dos pacotes de dados. Os quatro pacotes são processados pelo sistema a fim de obter o valor da energia de cada um deles e após o processamento dos dados dos pacotes, os quatro *buffers* têm seu conteúdo apagado e é efetuada uma nova coleta de dados no DAQ.

3.3.4 *SOFTWARE* FINAL

Com as classes de processamento de dados, manipulação do mouse e a rotina de aquisição de dados pronta, foi desenvolvido o *software* para que essas funções funcionem em conjunto de forma que se chegue ao objetivo desse trabalho, que é desenvolver um mouse virtual através de biosinais.

A interface foi criada com o objetivo de ter fácil compreensão por parte do usuário final e tamanho reduzido, para não ocupar espaço sobre outros programas abertos no computador. Ela possui dois dados, o primeiro mostra uma mensagem que informa o que o *software* irá executar (ação do *mouse virtual* ou momento do processo de calibração) e o segundo é um sistema de contagem regressiva para auxiliar no processo de calibração do *software*. Além disso, existe um menu com opção para abrir os programas de teste criados e com a opção de finalizar o programa, isso foi inserido na interface para facilitar o processo de desenvolvimento desse trabalho. Na Figura 3.16 é mostrada essa interface principal do *software*.

Foi implementada a característica "*Always on top*" para a interface do *mouse virtual*, ou seja, essa interface sempre estará visível na tela independentemente se algum outro programa for aberto.

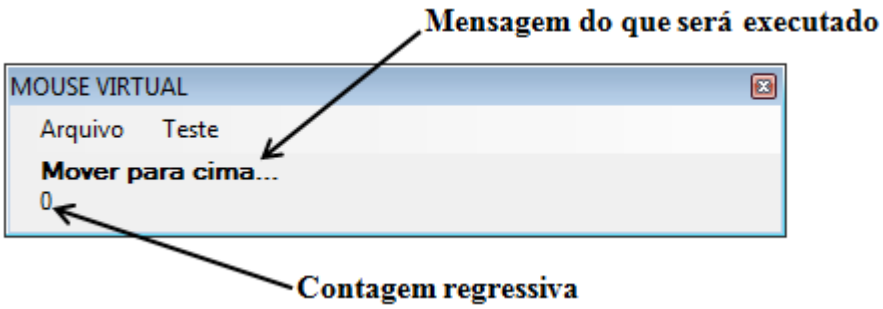

Figura 3.16 Interface do *software* principal.

Após a interface com o usuário ser desenvolvida, foi implementada a rotina de calibração do *software*. Essa rotina tem como função definir um limiar de energia para a hemiface (masseter + temporal) esquerda e um limiar de energia para a hemiface direita. Através desse limiar de energia é possível efetuar as comparações entre a energia dos pacotes coletados em tempo real e efetuar uma ação do *mouse virtual*. O processo de calibração funciona conforme descrito abaixo:

1. O *software* é inicializado e mostra imediatamente a mensagem "Prepare-se para calibrar o masseter esquerdo...", como é mostrado na Figura 3.17. Nesse momento o usuário tem cinco segundos (é iniciada uma contagem regressiva na interface) para ficar completamente relaxado. Durante esses cinco segundo o *software* não captura de nenhum dado através do DAQ;

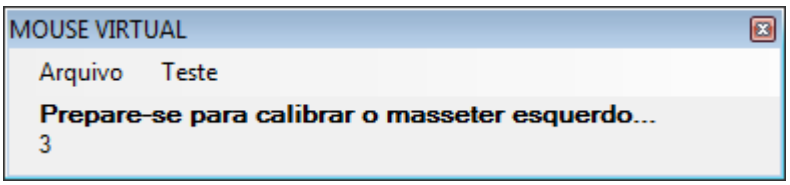

Figura 3.17 Interface do *software*: preparação para calibração da hemiface esquerda.

2. Após os cinco segundos de preparação, é iniciado o processo de calibração da hemiface esquerda. O *software* solicita ao usuário que faça uma contração com o masseter esquerdo (Figura 3.18) e inicia a coleta de dados do masseter esquerdo e

do temporal esquerdo. A cada 250ms são coletados os dados do masseter e do temporal esquerdos e os mesmos são armazenados em seus respectivos *buffers*. O usuário possui cinco segundos para efetuar uma contração com o masseter esquerdo, que conseqüentemente irá gerar algum tipo de contração no temporal esquerdo (esse comportamento foi verificado nos testes descritos no tópico 3.1 desse trabalho). Essa contração irá definir qual será o limiar de energia a ser utilizado para a hemiface esquerda;

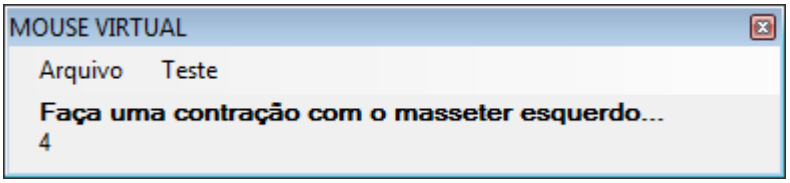

Figura 3.18 Interface do *software*: solicitando ao usuário que faça uma contração com o masseter esquerdo.

3. Após coletar os dados da hemiface esquerda, o *software* paralisa a coleta de dados através do DAQ e efetua o cálculo do limiar de energia que será utilizado essa hemiface. Esse cálculo tem as seguintes etapas:
    a. O *software* efetua a soma dos dados de um pacote do masseter e do temporal esquerdo, criando um pacote de dados da hemiface esquerda;
    b. Para o pacote de dados da hemiface esquerda é calculado um limiar de energia;
    c. Esse procedimento é repetido conforme a quantidade de pacotes coletados durante o período de cinco segundos de calibração, assim calculando um número de limiares de energia igual ao número de pacotes coletados durante o período de calibração. Exemplo: se o pacote de dados tem 250ms, então são calculados 20 limiares de energia para um período de 5 segundos.

d. Após calcular diversos limiares de energia, o *software* ordena os limiares do mais alto para o baixo e seleciona aquele que tiver na posição por volta 90% próxima do limiar mais alto (esse número foi determinado experimentalmente, isso será detalhado no tópico 4 desse trabalho). Exemplos: se existem 20 limiares, ele irá escolher o 18º limiar mais alto. Se existirem 21 ou 22 limiares, então o escolhido será o 19º mais alto.

4. Após a determinação do limiar da hemiface esquerda, o *software* mostra a seguinte mensagem durante cinco segundos na interface "Prepare-se para calibrar o masseter direto...", conforme é mostrado na Figura 3.19. Nesse momento o usuário deve relaxar os músculos para posteriormente efetuar a calibração da hemiface esquerda;

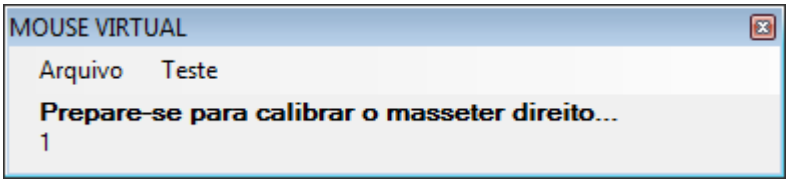

Figura 3.19 Interface do *software*: preparação para calibração da hemiface direita.

5. Após esses cinco segundos de relaxamento, é iniciado o processo de calibração da hemiface direita. O *software* solicita ao usuário que faça uma contração com o masseter direito (Figura 3.20) e inicia a coleta de dados do masseter direito e do masseter esquerdo, da mesma forma que foi descrita para a hemiface esquerda;

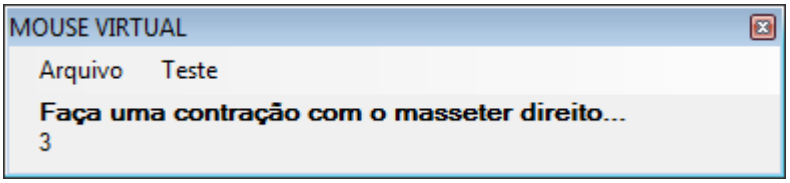

Figura 3.20 Interface do *software*: solicitando ao usuário que faça uma contração com o masseter direito.

6. Ao final dos cinco segundos de calibração da hemiface direita, o limiar para essa hemiface é calculado utilizando a mesma lógica explicada para o cálculo do limiar da hemiface esquerda. Após isso o processo de calibração dos limiares de energia é considerado concluído. Caso seja necessário efetuar uma nova calibração é necessário reiniciar o *software*.

No momento em que o processo de calibração é finalizado o *software* está pronto para ser utilizado pelo usuário e mostra a seguinte mensagem “Sistema calibrado. Contraia ambos masseteres...”, conforme é mostrado na Figura 3.21.

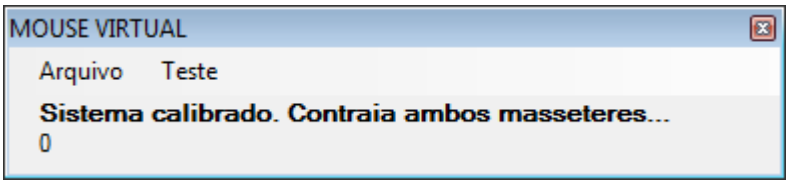

Figura 3.21 *Software* calibrado e pronto para ser utilizado.

A partir disso basta que o usuário efetue contrações com a hemiface direita ou hemiface esquerda ou ambas hemifaces, para que ele possa manipular o *mouse virtual*, de acordo com os comandos descritos no tópico 3.3.2. Os comandos são executados pelo *software* conforme descrito abaixo:

- Comando 1: é executado quando a energia dos pacotes com dados da hemiface (masseter + temporal) esquerda e direita possuem energia maior que os limiares calibrados para ambos lados;
- Comando 2: é executado quando a energia dos pacotes com dados da hemiface direita possui energia maior que o limiar calibrado para o lado direito;

- Comando 3: é executado quando a energia dos pacotes com dados da hemiface esquerda possui energia maior que o limiar calibrado para o lado esquerdo;

As mensagens selecionadas através do Comando 1 que aparecem na interface para que o usuário saiba qual ação ele irá executar são:

- “Mover para esquerda...”
- “Mover para direita...”
- “Mover para cima...”
- “Mover para baixo...”
- "Clique com o botão esquerdo..."
- "Duplo clique com o botão esquerdo..."
- "Clique com o botão direito..."

Quando uma função de movimento (esquerda, direita, cima e baixo) é executada, o *mouse virtual* se movimenta 15 pixels para a direção que foi selecionada e intervalo de movimento entre um pixel e outro é de 10ms. Esses valores foram determinados após testes práticos onde o movimento de 15 pixels foi determinado como tamanho de movimento adequado para o usuário conseguir navegar no monitor sem que o mouse ultrapasse do ponto desejado. Já o valor de 10ms de intervalo entre o movimento dos pixels, é para dar a sensação de mouse deslizando na tela, ao invés do mouse simplesmente pular de um ponto da tela para outro.

Mais detalhes sobre as rotinas descritas para o *software* final podem ser vistos no Anexo D desse trabalho.

4. RESULTADOS E DISCUSSÕES

Nesse tópico são demonstrados os resultados obtidos. Primeiramente são mostradas fotos dos ensaios para executar cada comando. Após são mostrados graficamente os biosinais que são reconhecidos após ser efetuado o processo de calibração. Depois são descritos os resultados com testes de acerto para cada tipo de comando utilizando as três técnicas de determinação do limiar. Por fim, são descritos alguns relatos de usuários do *software* sobre a utilização do mesmo e sugestões dadas pelos usuários para melhorias de usabilidade do *software*.

4.1 FOTOS DOS ENSAIOS

4.1.1 COMANDO 1 – CONTRAÇÃO DE AMBAS HEMIFACES

Para executar o comando 1 (troca de ação do *mouse virtual*) é necessária a contração de ambas hemifaces simultaneamente. Na Figura 4.1 é mostrada a foto de um usuário do sistema com ambas hemifaces relaxadas e a ação “Mover para cima” selecionada. Nesse momento o *software* não executa nenhuma ação.

A Figura 4.2 apresenta uma contração de ambas hemifaces, visualizando a ação “Mover para cima” continua selecionada, pois o *software* ainda não efetuou o processamento dos biosinais.

Já na Figura 4.3, ambas hemifaces estão relaxadas novamente e o *software* já executou o processamento dos biosinais, como se pode notar a ação do *mouse virtual* foi alterada para “Mover para baixo”, ou seja, o comando 1 foi executado com sucesso.

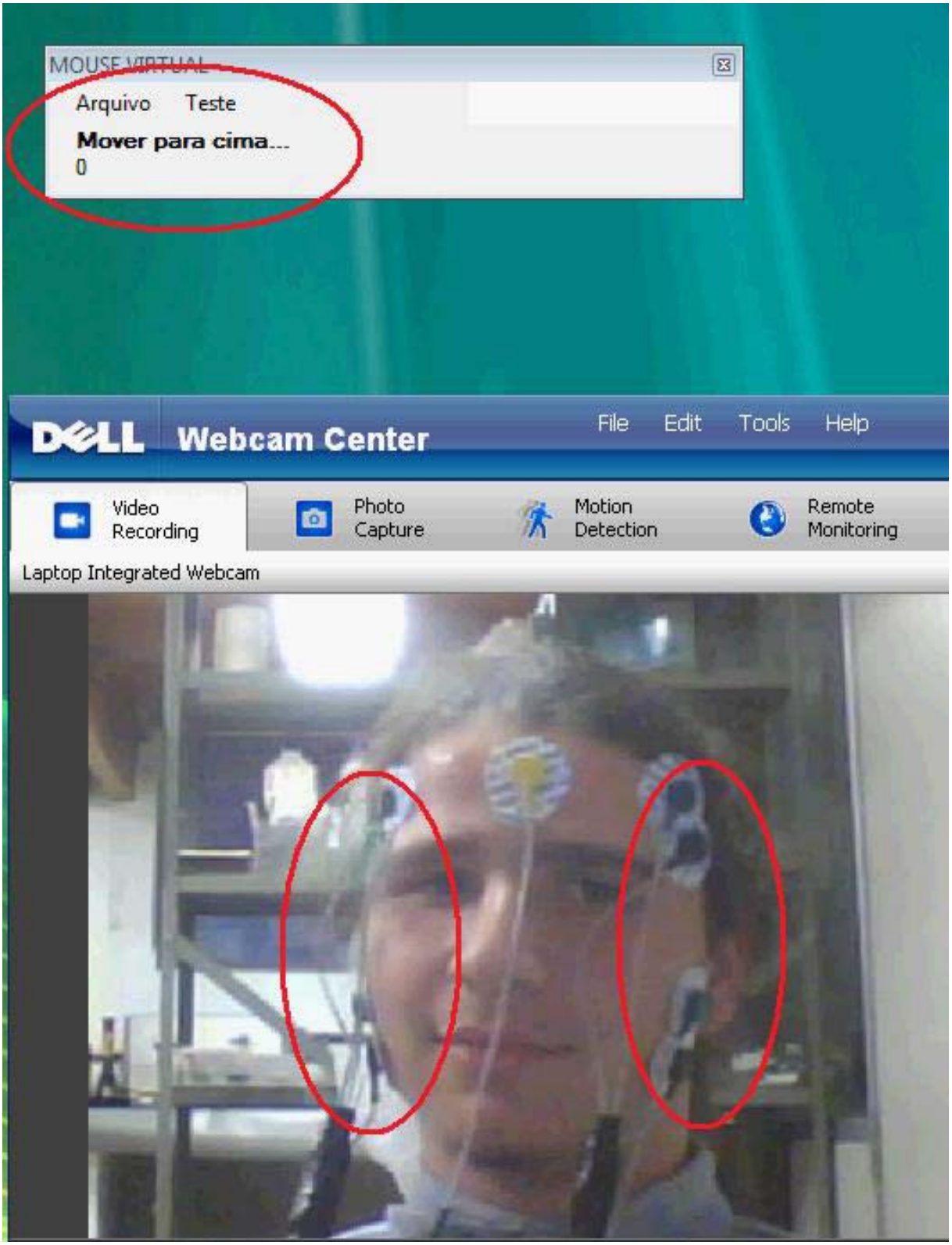

Figura 4.1 Ambas as hemifaces relaxadas e a ação “Mover para cima” selecionada.

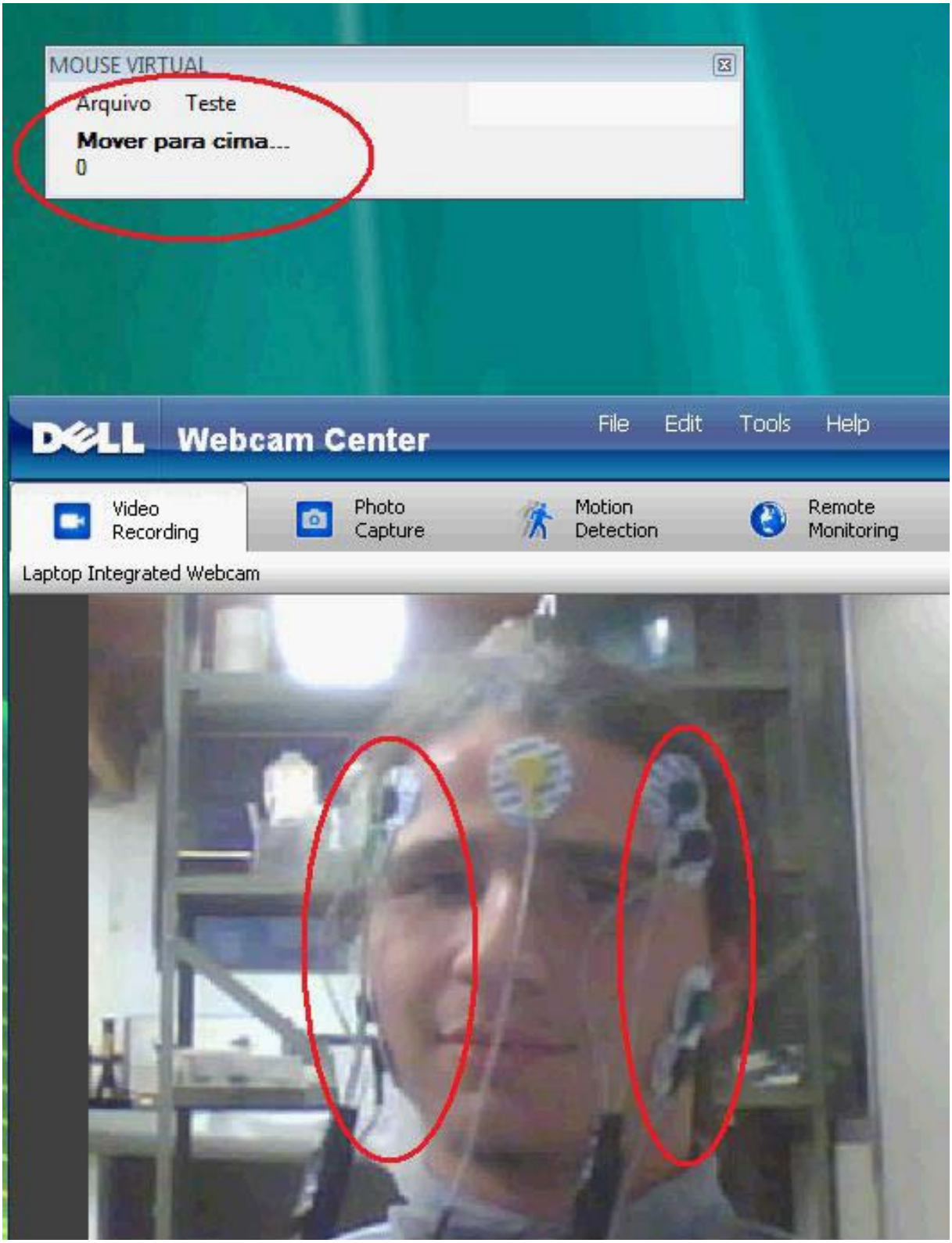

Figura 4.2 Ambas as hemifaces contraídas e a ação “Mover para cima” selecionada.

Figura 4.3 Ambas as hemifaces relaxadas e a ação “Mover para baixo” selecionada.

4.1.2 COMANDO 2 – CONTRAÇÃO DA HEMIFACE DIREITA

Para executar o comando 2 (executar ação do *mouse virtual* que está selecionada) é necessária a contração apenas da hemiface direita. Na Figura 4.4 é mostrada a foto de um usuário do sistema com ambas hemifaces relaxadas e a ação “Mover para baixo” selecionada. Nesse momento o *software* não executa nenhuma ação.

A Figura 4.5 apresenta uma contração apenas da hemiface direita, nesse momento o *software* ainda não efetuou o processamento dos biosinais. Na Figura 4.6, ambas hemifaces estão relaxadas e o *software* já executou o processamento dos biosinais, como se pode notar a o *mouse virtual* se deslocou para baixo, conforme a ação que estava selecionada no *software*, ou seja, o comando 2 foi executado com sucesso.

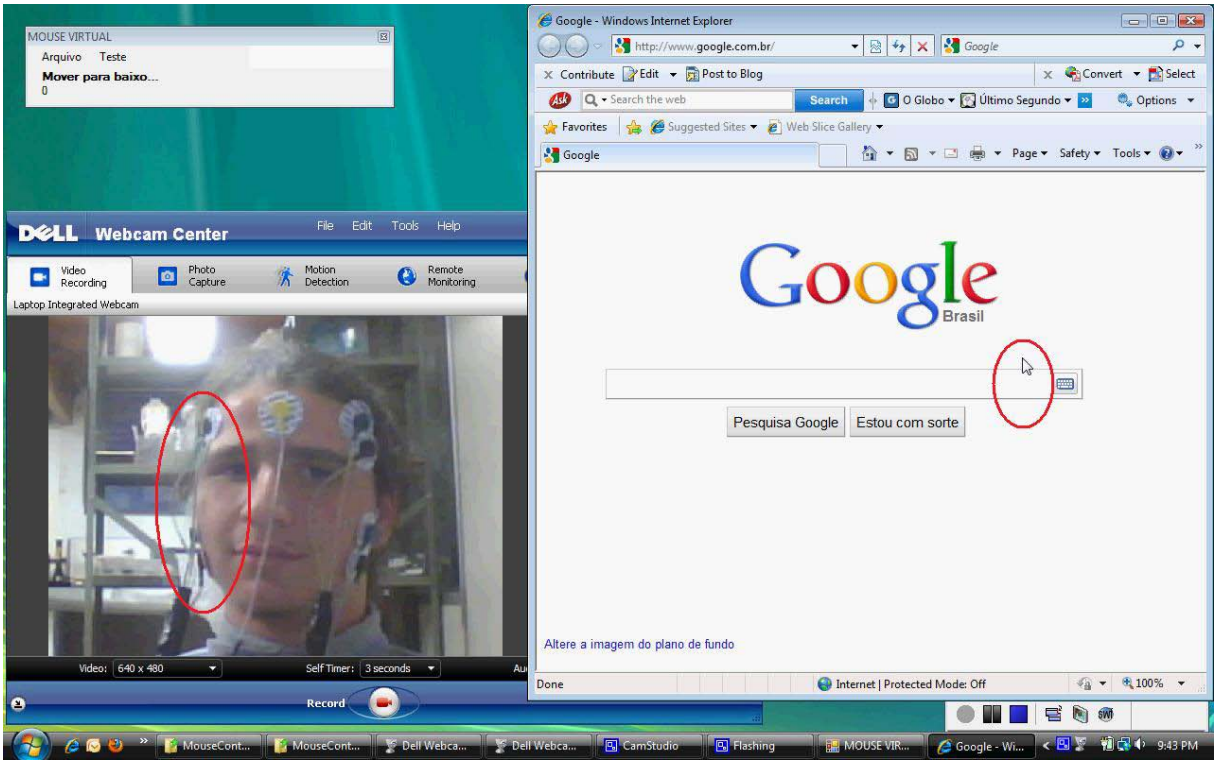

Figura 4.4 Ambas as hemifaces relaxadas e a posição do *mouse virtual*.

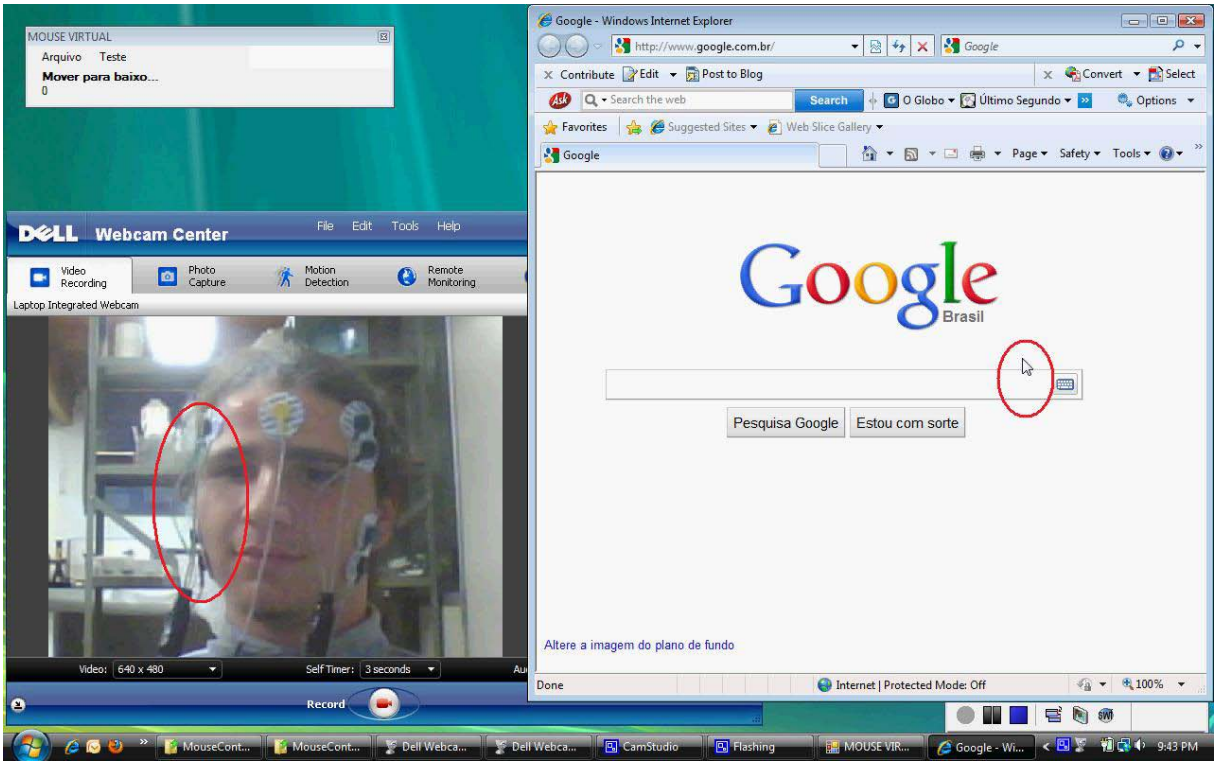

Figura 4.5 Hemiface direita contraída e a posição do *mouse virtual*.

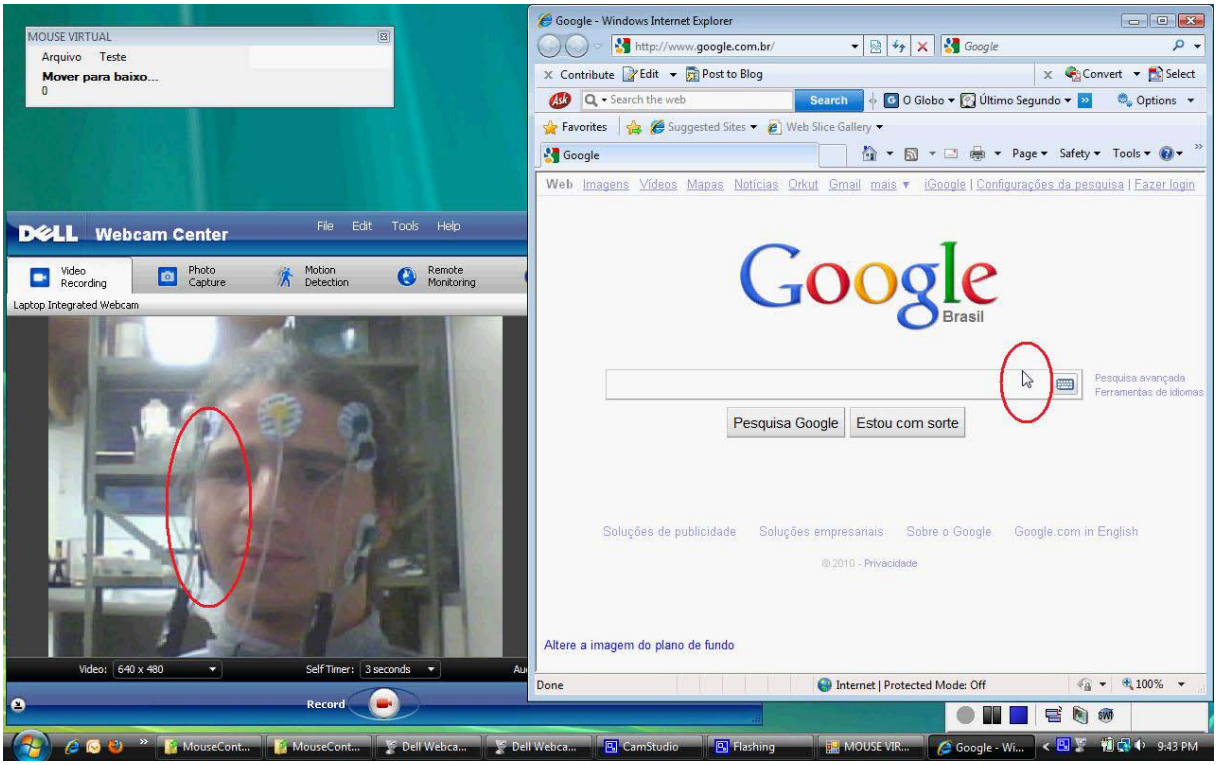

Figura 4.6 Ambas hemifaces relaxadas e a nova posição do *mouse virtual*.

4.1.3 COMANDO 3 – CONTRAÇÃO DA HEMIFACE ESQUERDA

Para executar o comando 3 (executar clique com botão esquerdo do *mouse virtual*) é necessária a contração apenas da hemiface esquerda. Na Figura 4.7 é mostrada a foto de um usuário do sistema com ambas hemifaces relaxadas e o *mouse virtual* posicionado sobre a letra “o”, do teclado do *Google*. Nesse momento o *software* não executa nenhuma ação.

A Figura 4.8 apresenta uma contração apenas da hemiface esquerda, nesse momento o *software* ainda não efetuou o processamento dos biosinais. Na Figura 4.9, ambas as hemifaces estão relaxadas e o *software* já executou o processamento dos biosinais, como se pode notar a o *mouse virtual* “clicou” na letra “o” e a mesma apareceu no campo de texto do *Google*, ou seja, o comando 3 foi executado com sucesso.

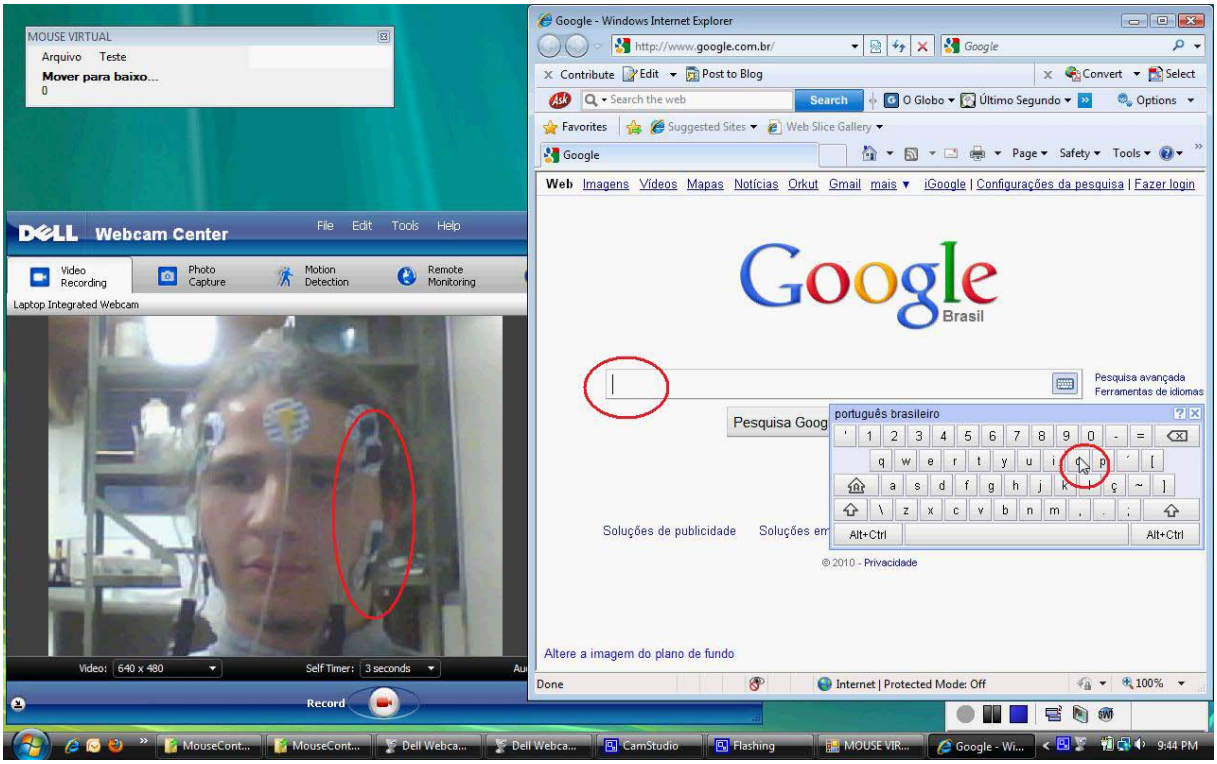

Figura 4.7 Ambas as hemifaces relaxadas e o *mouse virtual* sobre a letra “o”.

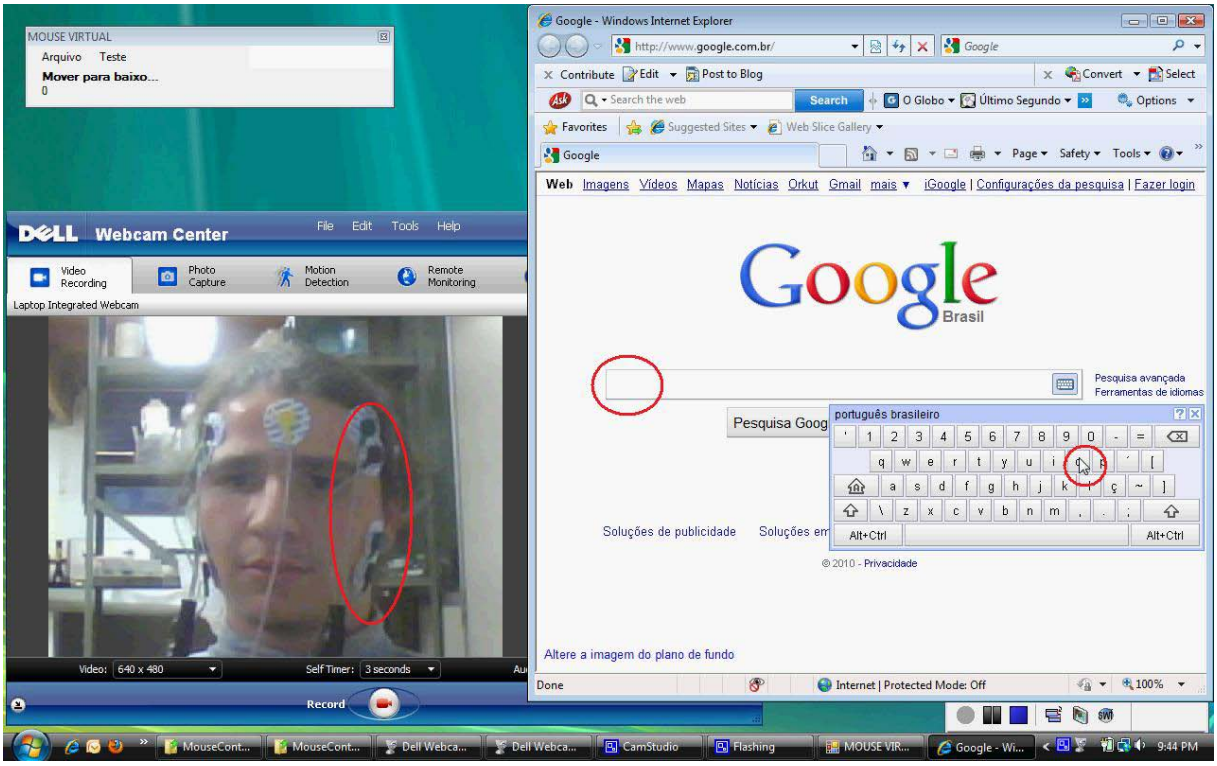

Figura 4.8 Hemiface esquerda contraída e nenhum comando foi executado.

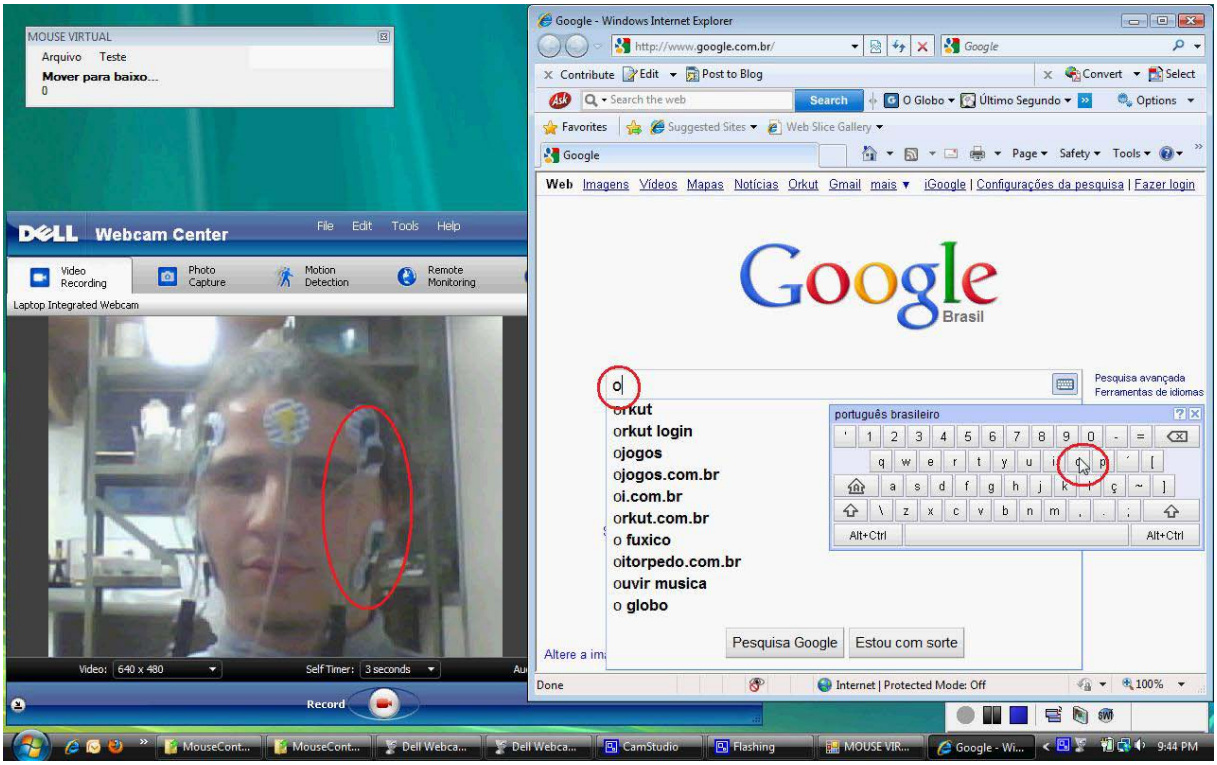

Figura 4.9 Ambas hemifaces relaxadas e clique executado.

4.2 TESTES DE RECONHECIMENTO DE SINAIS APÓS CALIBRAÇÃO

A Figura 4.10 mostra o tipo de sinal utilizado pelo *software* para a execução do Comando 1 (troca de função do *mouse virtual*). Pode-se notar que no mesmo instante de tempo existe atividade nos quatro canais. Foi verificado que sinais com esse mesmo padrão e que possuem energia superior aos limiares calibrados são reconhecidos com sucesso pelo *software* e o Comando 1 é executado corretamente.

Já a Figura 4.11 mostra o tipo de sinal utilizado pelo *software* para a execução do Comando 2 (executar a função selecionada no Comando 1). Em um mesmo instante de tempo que existe atividade nos dois canais relativos à hemiface direita e que nos canais relativos à hemiface esquerda existe apenas ruído. Nos testes foi constatado que sinais com esse mesmo padrão e que possuem energia superior ao limiar calibrado para a hemiface direita são reconhecidos pelo *software*, com isso o Comando 2 é executado com sucesso.

Na Figura 4.12 é mostrado o tipo de sinal utilizado pelo *software* para a execução do Comando 3 (simular clique com o botão esquerdo do *mouse*). Verifica-se que no mesmo instante de tempo que existe atividade nos dois canais relativos à hemiface esquerda, no canal relativo masseter direito existe um pequeno aumento de tensão e no canal relativo ao temporal direito, apenas ruído. Nos testes foi possível confirmar o *software* executou o Comando 3 com sucesso, mesmo com um leve aumento de tensão no masseter direito. Como esse leve aumento de tensão masseter direito não ultrapassou o limiar calibrado para a hemiface direita, o *software* descartou esse dado.

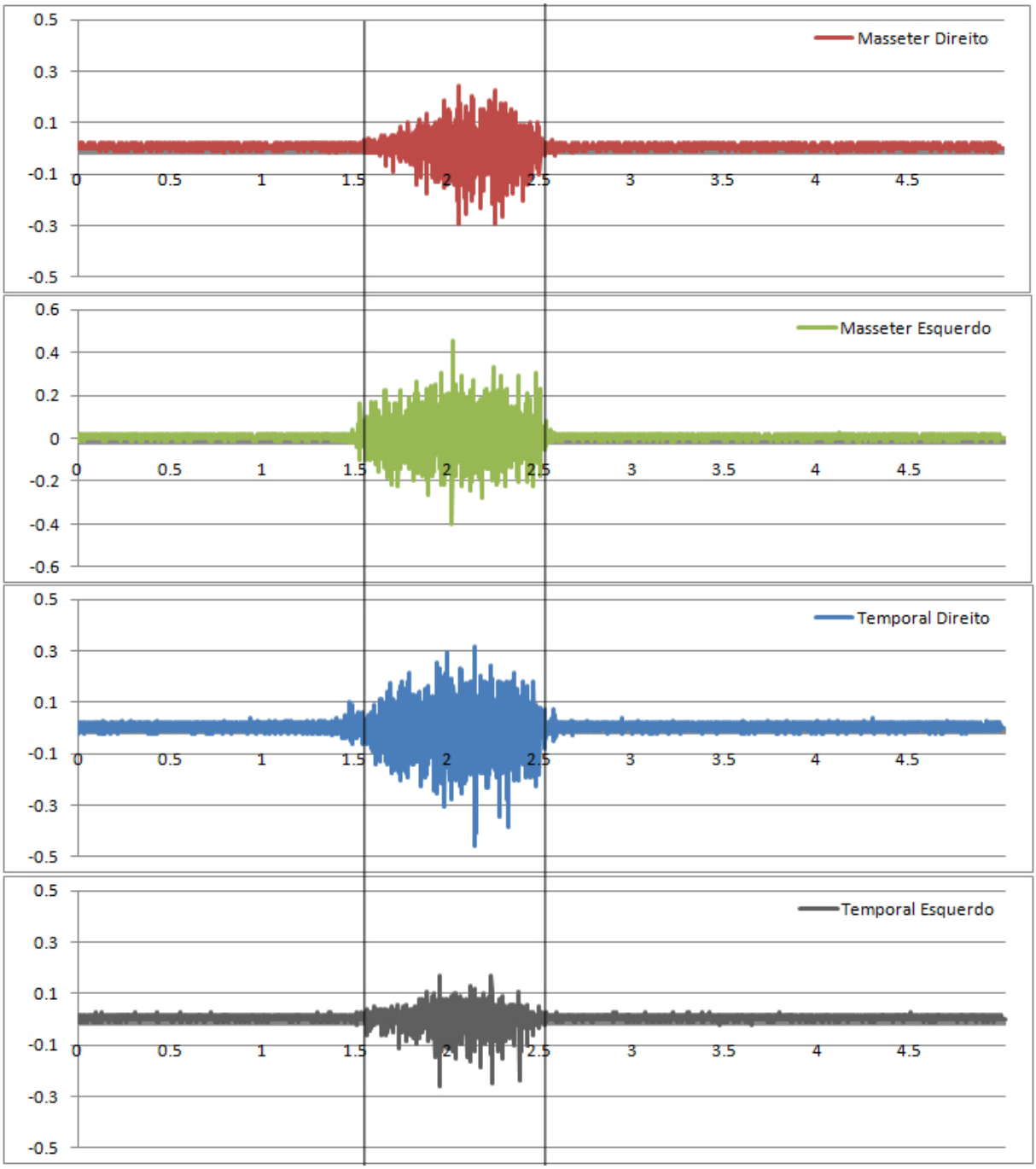

Figura 4.10 Tipo de biosinal necessário para executar o Comando 1.

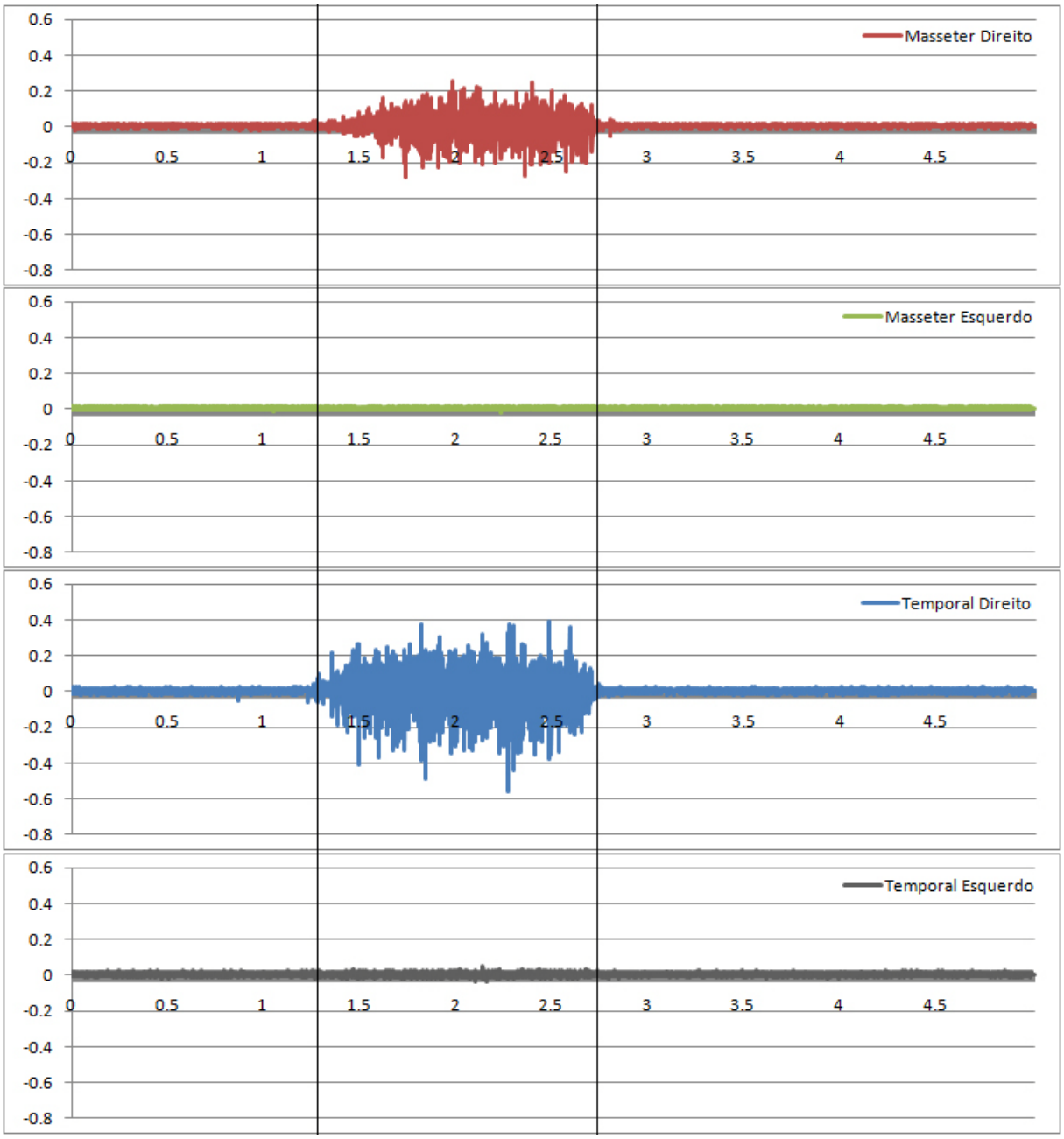

Figura 4.11 Tipo de biosinal necessário para executar o Comando 2.

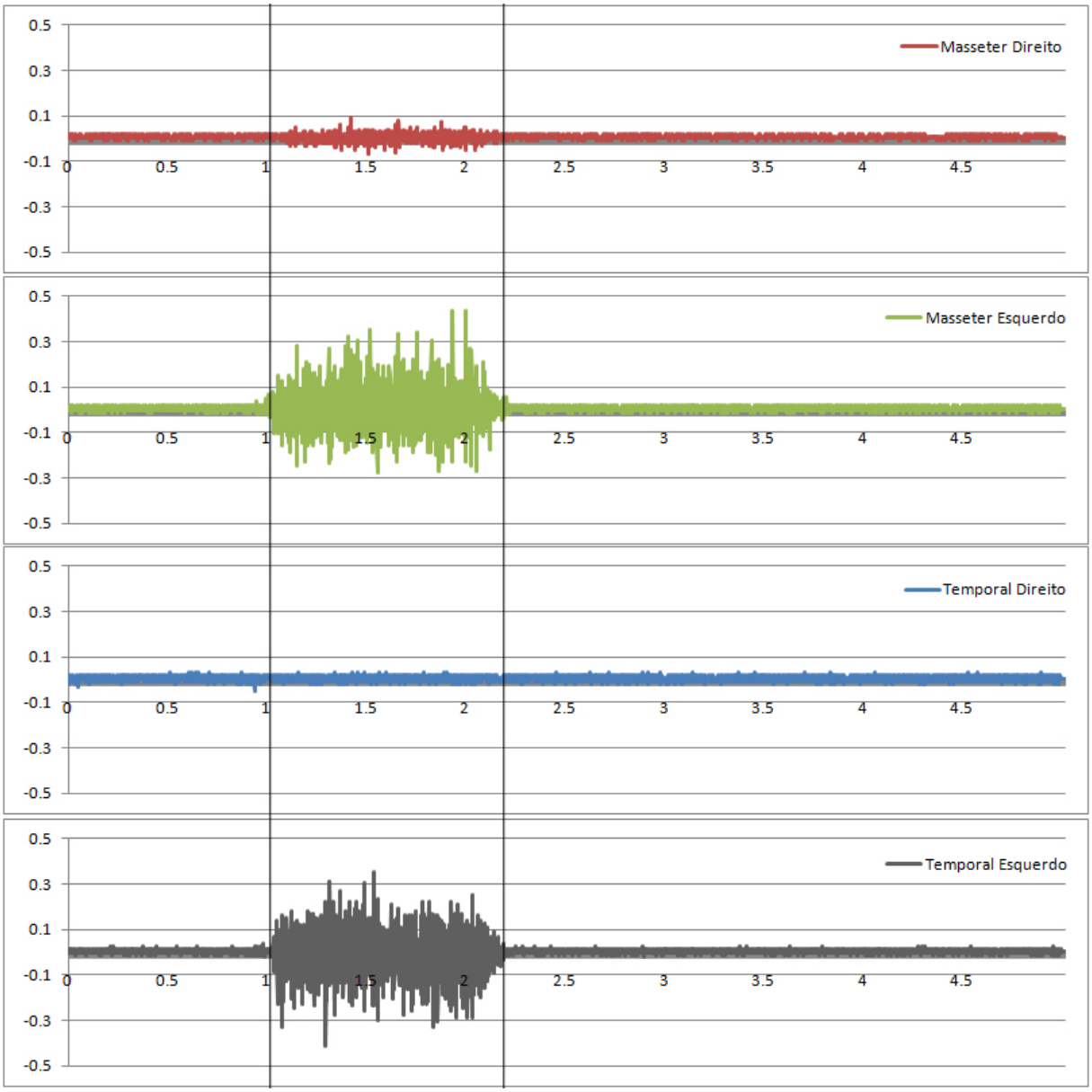

Figura 4.12 Tipo de biosinal necessário para executar o Comando 3.

4.3 TESTES DE ASSERTIVIDADE COM AS TÉCNICAS IMPLEMENTADAS

Foram realizados testes de assertividade de comando com as três técnicas de determinação de limiar de energia que foram implementadas (Estática, LED e ALED). Esses testes visam determinar qual das três técnicas é mais eficiente para determinar um limiar de energia o mais próximo do ideal para um usuário.

4.3.1 METODOLOGIA DOS TESTES

Para os testes de assertividade por técnica foram utilizados dois voluntários. Cada voluntário teve que efetuar 30 contrações de cada tipo (contração com hemiface esquerda, contração com hemiface direita e contração com ambas hemifaces simultâneamente) para cada uma das três técnicas implementadas, ou seja, para cada técnica foram realizadas 90 contrações. Para cada contração foi anotado se o comando esperado foi executado com sucesso ou se ocorreu alguma falha. As falhas podem ser de três tipos:

1. O *sofware* não executa nenhum comando;
2. O comando executado não era o comando esperado;
3. O comando foi executado duas vezes seguidas;

O voluntário 01 fez os testes primeiro com a técnica LED, depois ALED e por último com a técnica estática. Já o voluntário 02 os testes foram realizados primeiro com a técnica ALED, após com a estática e por último com a LED.

Antes dos testes serem iniciados, cada um dos voluntários utilizou o *software* por 40 minutos para que o mesmo pudesse se adaptar aos comandos e ao uso do mesmo.

4.3.2 ACERTOS E FALHAS DOS ENSAIOS

Os resultados para a técnica LED, utilizando o passo de adaptação (p) igual a 0,5, são apresentados na Tabela 4.1.

Tabela 4.1 Testes com a técnica LED.

Voluntário	Comando 1		Comando 2		Comando 3	
	Acertos	Falhas	Acertos	Falhas	Acertos	Falhas
Voluntário 01	26	4	23	7	27	3
Voluntário 02	25	5	28	2	21	9

Para a técnica LED em 180 testes realizados o total de acertos foi de 150, contra 30 erros, ou seja, um percentual total de acerto de 83,3%.

Na Tabela 4.2 são mostrados os de resultados dos testes para a técnica ALED.

Tabela 4.2 Testes com a técnica ALED.

Voluntário	Comando 1		Comando 2		Comando 3	
	Acertos	Falhas	Acertos	Falhas	Acertos	Falhas
Voluntário 01	27	3	25	5	28	2
Voluntário 02	24	6	26	4	23	7

A técnica ALED apresentou 153 acertos contra 27 erros no total dos 180 testes realizados, ou seja, um percentual total de acerto de 85%.

Para os testes com a técnica estática foram utilizadas margens de segurança diferentes para cada voluntário. Para o Voluntário 01 foi utilizado 10% de margem de segurança, enquanto para o Voluntário 02 foi utilizado 30% de margem de segurança. Essa diferença ocorreu porque utilizando essa técnica com 10% e depois com 20% de margem de segurança para o Voluntário 02 o *software* se mostrou completamente instável. A Tabela 4.3 apresenta os resultados dos testes para a técnica estática.

Tabela 4.3 Testes com a técnica estática.

Voluntário	Comando 1		Comando 2		Comando 3	
	Acertos	Falhas	Acertos	Falhas	Acertos	Falhas
Voluntário 01	25	5	16	14	28	2
Voluntário 02	23	7	27	3	15	15

Por fim, a técnica estática teve 134 acertos, mediante 46 erros nos 180 testes com os dois voluntários, ou seja, um percentual total de acerto de 74,4%.

O voluntário 01 revelou que possui mais facilidade em controlar os músculos da sua hemiface esquerda. Já o voluntário 02 revelou que para ele é mais fácil controlar os músculos da hemiface direita. Para os testes com as três técnicas ambos acertaram mais comandos com a hemiface que possuem mais facilidade em controlar, enquanto o comando que exige o

controle muscular das duas hemifaces simultaneamente, ambos os voluntários tiveram alta quantidade de acertos, como pode-ser ver nas Tabelas 4.1, 4.2 e 4.3.

A Tabela 4.4 mostra um comparativo percentual de falhas e acertos para cada uma das técnicas

Tabela 4.4 Comparativo entre as técnicas.

	LED	ALED	ESTÁTICA
ACERTOS	83%	85%	74%
FALHAS	17%	15%	26%

Analisando a Tabela 4.4, nota-se que não existe uma diferença considerável entre a taxa de acertos das técnicas que são adaptativas (LED e ALED), já para a técnica estática mais de um quarto das tentativas de efetuar um comando falharam.

Outro fator relevante é que a técnica estática teve ter sua margem de segurança alterada de 10% para 30% para que funcionasse com o voluntário 02. Além disso, a técnica estática apresentou 50% de falhas para o comando 3 nos testes do voluntário 02, comando esse que é executado com a hemiface que o voluntário 02 relatou ter mais dificuldade em controlar. Para o voluntário 01 o comando 2 teve 47% de falhas utilizando a técnica estática, essa hemiface é a que o voluntário 01 relatou ter mais dificuldade de controlar. Com as técnicas adaptativas a maior taxa de falhas para uma das hemifaces foi de 30% para uma hemiface que um voluntário relatou ter mais dificuldade de controle, ou seja, as técnicas adaptativas ajudam consideravelmente a melhorar o desempenho do *software* para uma hemiface que o usuário relata ter dificuldades de controle.

4.3 TRABALHOS FUTUROS

Abaixo uma lista de sugestões de trabalhos futuros para continuidade do que foi desenvolvido nesse trabalho:

- Analisar se a taxa de falhas está relacionada à redução da impedância dos eletrodos ao longo de um ensaio;
- Analisar o tempo ótimo de uso do sistema em função da fadiga muscular ao longo de um ensaio;
- Analisar a taxa de assertividade da técnica LED com diferentes passos de adaptação (parâmetro *p*);
- Desenvolver uma maneira de determinar o tamanho dos pacotes de dados dinamicamente, a fim de diminuir a taxa de erros.

5. CONCLUSÕES

O objetivo traçado no início do desenvolvimento desse projeto foi alcançado com êxito. Foi provado que é possível desenvolver um *mouse virtual* experimental para computador manipulado através de biosinais da face de uma pessoa. Para que isso fosse possível o laboratório IEE, da UFRGS, forneceu todos os recursos necessários para que esse trabalho fosse desenvolvido. Além dos recursos do laboratório IEE, o curso de Engenharia Elétrica forma seus alunos para serem capazes de projetar e desenvolver utilizando diversos tipos de tecnologias.

Esse projeto me propiciou um grande aprendizado tanto na área de instrumentação biomédica quanto na área de processamento de sinais em tempo real. A pesquisa e leitura de artigos foram essenciais para as etapas de revisão bibliográfica e metodologia experimental do projeto, para isso o *website* do IEEE foi fundamental, além da colaboração do meu orientador para nortear o projeto.

O trabalho foi modelado para que as classes de processamento de sinais e manipulação do mouse possam ser reaproveitadas pelo laboratório IEE para futuras pesquisas, pois as mesmas foram desenvolvidas de forma a serem módulos externos do *software*. Todo o código desenvolvido foi organizado e documentado de forma a facilitar desenvolvimentos futuros e manutenção de código.

Para transformar esse *software* em um produto comercial alguns itens teriam que ser melhorados, como a diminuição da quantidade de erros de detecção de comandos. Isso pode ser feito através da implementação de pacotes de dados com tamanho variável, ao invés de tamanho fixo como foi implementado nesse trabalho. Além disso, como são utilizados 9 eletrodos para uso do *software*, a quantidade de fios se torna inconveniente para um usuário doméstico. Para a redução de fios, poderia ser utilizado apenas o sinal dos masseteres direito e

esquerda além do uso de um conversor AD e eletromiógrafo que utilizem alguma tecnologia sem fio.

Ao total o *software* teve três usuários, foram coletados os seguintes relatos:

- Os três usuários notaram que quanto mais tempo utilizavam o *software* mais fácil ficava de controlar o mesmo, pois eles não possuem o hábito de utilizar os músculos masseter e temporal para o controle de um mouse;
- Dois usuários relataram que efetuar uma nova calibração após utilizar o *software* por alguns minutos melhora a usabilidade, pois após alguns minutos de uso eles já estão mais habituados a contrair os músculos das hemifaces, isso faz com que a calibração fique mais adequada ao uso;
- Os três usuários relataram cansaço nos músculos das hemifaces durante o uso do *software*.

Por fim, foi possível determinar que as técnicas LED e ALED são melhores do que a técnica estática para este tipo de aplicação, visto que as suas taxas de falhas, 17% e 15%, respectivamente, são consideravelmente menores que os 26% da técnica estática.

REFERÊNCIAS

BERSCH, R. *Introdução a Tecnologia Assistiva*. Centro Especializado em desenvolvimento Infantil, Artigo, Porto Alegre, 2008.

Assistive Technology Act of 2004, Lei Pública 108-364, Estados Unidos, 2004.

Secretaria Especial dos Direitos Humanos - SEDH, Governo Brasileiro. Disponível em: <http://www.presidencia.gov.br/estrutura_presidencia/sedh/>. Acesso em 04 de abril de 2010.

MELO, A. M., COSTA, J. B., SOARES, S. M. *Acessibilidade, Discurso e Prática no Cotidiano das Bibliotecas*, cap. 8, Unicamp, Campinas, 2006.

ISO 9999:2007. Norma Internacional; classificação. Disponível em <http://www.unit.org.uy/misc/catalogo/9999.pdf>. Acesso em 04 de abril de 2010.

OLIVEIRA, M.G., *Manual da Anatomia da Cabeça e do Pescoço*, 4ª Edição, Porto Alegre, EDIPUCRS, 2002. IBSN: 85-7430-295-3.

CERQUEIRA, C., *Anatomia da ATM*, Hospital Geral de Santo Antonio, Serviço de Estomatologia e Cirurgia Maxilo-Facial, 2005.

Ortodontista.net, <http://www.ortodontista.net/dtm/bruxismo.jpg>. Acesso em 11 de abril de 2010.

DE LUCA, C. J., *The use of Surface Electromyography in Biomechanics*, The International Society for Biomechanics, Neuromuscular Research Center, Boston, 1995.

VEIGL, C., *An Open-Source System for Biosignal- and Camera-Mouse Applications*, Young Researchers Consortium of the ICCHP, 2006, Linz

BARRETO, A. B., SCARGLE, S. D., ADJOUADI, M., *A Real-Time Assistive Computer Interface for Users with Motor Disabilities*, Department of Electrical and Computer Engineering, Florida International University

CHALUB, A. A., ALMEIDA, J. J., DIAS, N. L., MORAIS S. G., *O Uso da Eletromiografia de Superfície como Recurso de Avaliação e Tratamento Fisioterapêutico em Indivíduos Hemiparéticos Crônicos*, UNIVALE, Governador Valadares, 2008

RADTKE, C.A., *Protótipo de um sistema de aquisição e processamento de sinais mioelétricos para caracterização de fadiga muscular*, ULBRA, Canoas, 2007

SMITH, S., *The Scientist and Engineer's Guide to Digital Signal Processing*, capítulo 3, 1997, ISBN 0-9660176-3-3

STAUDE, G., WOLF, W., APPEL, U., *Automatic Event Detection in Surface EMG of Rhythmically Activated Muscles*, 1995 IEEE-EMBC and CMBEC Theme 5: Neuromuscular SystemdBiamechanics

PAL, P. R., PANDA, R., *Classification Of EEG Signals For Epileptic Seizure Evaluation*, Proceedings of the 2010 IEEE Students' Technology Symposium 3-4 April 2010, IIT Kharagpur

PRASAD, R.V., SANGWAN, A., CHIRANTH, M. C. SAB., *Comparation of Voice Activity Detection Algoritms for VoIP*, IEEE, Bangalore ntár, 2002

SANGWAN, A., CHIRANTH, M. C., JAMADAGNI, H. S., SAH, R., PRASAD, R. V., VAD *Techniques for Real-Time Speech Transmission on the Internet*, High Speed Networks and Multimedia Communications 5th IEEE International Conference on, Jeju Island, S. KOREA, 2002

RENEVEY, P., DRYGAJLO, A., *Entropy Based Voice Activity Detection in Very Noisy Conditions*, European Conference on Speech Communication and Technology, Aalborg, Denmark, September 2001, vol. 3, pp. 1883–1886

TANYER, S.G., ÖZER, H., *Voice activity Detection in Nonstationary Gaussian Noise*, IEEE Transactions on Speech and Audio Processing. Vol. 8, NO. 4, July 2000

National Instruments Measurement Studio 2009. Disponível em <http://www.ni.com/mstudio/>. Acesso em 27 de maio de 2010

ANEXO A – CLASSE IEESIGNALFUNCTIONS

```
using System;
using System.Collections;
using System.Linq;
using System.Text;

namespace MouseControl
{
    class IEESignalFunctions
    {
        private double PacketEnergy; //energia do pacote atual
        private ArrayList PacketContent = new ArrayList(); //conteudo de
cada amostra do pacote atual
        private double LastPacketEnergy; //energia do ultimo pacote
calculado
        private double StaticThreshold; //limiar
        private double LEDThreshold; //limiar
        private double ALEDThreshold; //limiar
        private int SamplesRate; //taxa de amostragem
        private int NumberOfSamples; //numero de amostras de um pacote
        private int SizeOfPacket; //tempo do pacote em ms
        private double PacketAverage; //media aritmetica dos dados de um
pacote
        private double PacketVariance; //variancia dos dados de um pacote
        private double LastPacketVariance; //variancia dos dados do ultimo
pacote
        private double PacketStandardDeviation; //desvio padrao dos dados
de um pacote


        //funcao que retorna a energia do pacote atual
        public double getPacketEnergy()
        {
            CalculatePacketEnergy();
            return PacketEnergy;
        }

        //funcao que seta a energia do pacote atual
        public void setPacketEnergy(double en)
        {
            PacketEnergy = en;
        }

        //funcao que retorna a taxa de amostragem
        public double getSamplesRate()
        {
            return SamplesRate;
        }

        //funcao que seta a frequencia de amostragem
        public void setSamplesRate(int sr)
        {
            SamplesRate = sr;
        }

        //funcao que retorna o numero de amostras
        public int getNumberOfSamples()
        {
            return NumberOfSamples;
```

```
        }

        //funcao que o seta o numero de amostras
        public void setNumberOfSamples(int nos)
        {
            NumberOfSamples = nos;
        }

        //calcula o numero de amostras de um pacote com base na taxa de
amostragem e no tamanho do pacote
        public void CalculateNumberOfSamples()
        {
            NumberOfSamples = SizeOfPacket * SamplesRate / 1000;
        }

        //funcao que retorna o tamanho de um pacote
        public int getSizeOfPacket()
        {
            return SizeOfPacket;
        }

        //funcao que seta o tamanho de um pacote
        public void setSizeOfPacket(int sop)
        {
            SizeOfPacket = sop;
        }

        //funcao que seta os dados de um pacote
        public void setPacketContent(ArrayList content)
        {
            int i;

            PacketContent.Clear();

            for (i = 0; i < NumberOfSamples; i++)
            {
                PacketContent.Add(content[i]);
            }

        }

        //funcao que retorna os dados de um pacote
        public ArrayList getPacketContent()
        {
            return PacketContent;
        }

        //funcao que limpa os dados de um pacote
        public void ClearPacketContent()
        {
            PacketContent.Clear();
        }

        //funcao que calcula a energia de um pacote
        public void CalculatePacketEnergy()
        {
            int i;

            LastPacketEnergy = PacketEnergy;
            PacketEnergy = 0;
```

```
            for (i = 0; i < NumberOfSamples; i++)
            {
                PacketEnergy = System.Convert.ToDouble(PacketContent[i]) *
System.Convert.ToDouble(PacketContent[i]) + PacketEnergy;
            }

            PacketEnergy = PacketEnergy / NumberOfSamples;
        }

        //funcao que calcula a media aritmetica dos dados um pacote
        public void CalculatePacketAverage()
        {
            int i;

            PacketAverage = 0;

            for (i = 0; i < NumberOfSamples; i++)
            {
                PacketAverage = System.Convert.ToDouble(PacketContent[i]) +
PacketAverage;
            }

            PacketAverage = PacketAverage / NumberOfSamples;
        }

        //funcao que retorna a media aritmetica dos dados de um pacote
        public double getPacketAverage()
        {
            return PacketAverage;
        }

        //funcao que calcula a variancia dos dados de um pacote
        public void CalculePacketVariance()
        {
            int i;

            LastPacketVariance = PacketVariance;
            PacketVariance = 0;

            CalculatePacketAverage();

            for (i = 0; i < NumberOfSamples; i++)
            {
                PacketVariance =
Math.Pow(System.Convert.ToDouble(PacketContent[i]) - PacketAverage, 2) +
PacketVariance;
            }

            PacketVariance = PacketVariance / NumberOfSamples;
        }

        //funcao que retorna a variancia dos dados de um pacote
        public double getPacketVariance()
        {
            return PacketVariance;
        }

        //funcao que calcula o desvio padrao dos dados de um pacote
        public void CalculePacketStandardDeviation()
        {
            PacketStandardDeviation = 0;
```

```
            CalculePacketVariance();
            PacketStandardDeviation = Math.Sqrt(PacketVariance);
        }

        //funcao que retorna o desvio padrao dos dados de um pacote
        public double getPacketStandardDeviation()
        {
            return PacketStandardDeviation;
        }

        //funcao que define um limiar estatico
        //Parametros:   noise = valor da energia do ruido
        //              margin = margem de seguranca, deve ser igual ou
maior que 1
        public void setStaticThreshold(double noise, double margin)
        {
            if (margin<1)
                return;

            StaticThreshold = noise * margin;
        }

        //funcao que retorna o limiar
        public double getStaticThreshold()
        {
            return StaticThreshold;
        }

        //funcao que aplica a tecnica LED para determinacao do limiar
        public void ApplyLEDTechnique(double p)
        {
            CalculatePacketEnergy();
            LEDThreshold = (1 - p) * LastPacketEnergy + p * PacketEnergy;
        }

        //retorna o limiar calculado pela tecnica LED
        public double getLEDThreshold()
        {
            return LEDThreshold;
        }

        //funcao que aplica a tecnica ALED para determinacao do limiar
        public void ApplyALEDTechnique()
        {
            double VarianceIndex;


            CalculatePacketEnergy();
            CalculePacketVariance();

            if (LastPacketVariance != 0)
            {
                VarianceIndex = PacketVariance / LastPacketVariance;

                if (VarianceIndex >= 1.25)
                    ALEDThreshold = (1 - 0.25) * LastPacketEnergy + 0.25 *
PacketEnergy;

                if ((VarianceIndex >= 1.1) && (VarianceIndex < 1.25))
                    ALEDThreshold = (1 - 0.2) * LastPacketEnergy + 0.2 *
PacketEnergy;
```

```
                if ((VarianceIndex >= 1) && (VarianceIndex < 1.1))
                    ALEDThreshold = (1 - 0.15) * LastPacketEnergy + 0.15 *
PacketEnergy;

                if (VarianceIndex < 1)
                    ALEDThreshold = (1 - 0.10) * LastPacketEnergy + 0.10 *
PacketEnergy;
            }
        }

        //retorna o limiar calculado pela tecnica ALED
        public double getALEDThreshold()
        {
            return ALEDThreshold;
        }
    }
      }
```

ANEXO B – CLASSE QUE IMPLEMENTA AS FUNÇÕES DO MOUSE

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Runtime.InteropServices;


namespace MouseControl
{
    class IEEMouseFunctions : Control
    {

        Point Pointer = new Point();

        [DllImport("user32.dll", CharSet = CharSet.Auto, CallingConvention
= CallingConvention.StdCall)]
        public static extern void mouse_event(long dwFlags,
                                              long dx,
                                              long dy,
                                              long cButtons,
                                              long dwExtraInfo);

        private const int MOUSEEVENTF_LEFTDOWN = 0x02;
        private const int MOUSEEVENTF_LEFTUP = 0x04;
        private const int MOUSEEVENTF_RIGHTDOWN = 0x08;
        private const int MOUSEEVENTF_RIGHTUP = 0x10;

        //função que centraliza o mouse
        public void CenterMouse()
        {
            int ScreenHeight;
            int ScreenWidth;

            ScreenHeight = Screen.PrimaryScreen.Bounds.Height;
            ScreenWidth = Screen.PrimaryScreen.Bounds.Width;

            Pointer.X = ScreenWidth / 2;
            Pointer.Y = ScreenHeight / 2;

            Cursor.Position = Pointer;
        }

        //função que retorna a horizontal do mouse
        private int GetMousePositionX()
        {
            int x;
            x = MousePosition.X;
            return x;
        }

        //função que retorna a vertical do mouse
        private int GetMousePositionY()
        {
            int y;
```

```
            y = MousePosition.Y;
            return y;
        }

        //função que move o mouse para a esquerda
        //Parâmetros:   PixelsDistance = quantidade de pixels que o mouse
irá se deslocar
        //              DelayPerStep = tempo em milissegundos para o
movimento de cada pixel
        public void MoveMouseLeft(int PixelsDistance, int DelayPerStep)
        {
            int i;


            for (i = 0; i < PixelsDistance; i++)
            {
                Pointer.X = GetMousePositionX() - 1;
                Pointer.Y = GetMousePositionY();
                Cursor.Position = Pointer;
                System.Threading.Thread.Sleep(DelayPerStep);
            }
        }

        //função que move o mouse para a direita
        //Parâmetros:   PixelsDistance = quantidade de pixels que o mouse
irá se deslocar
        //              DelayPerStep = tempo em milissegundos para o
movimento de cada pixel
        public void MoveMouseRight(int PixelsDistance, int DelayPerStep)
        {
            int i;

            for (i = 0; i < PixelsDistance; i++)
            {
                Pointer.X = GetMousePositionX() + 1;
                Pointer.Y = GetMousePositionY();
                Cursor.Position = Pointer;
                System.Threading.Thread.Sleep(DelayPerStep);
            }
        }

        //função que move o mouse para a cima.
        //Parâmetros:   PixelsDistance = quantidade de pixels que o mouse
irá se deslocar
        //              DelayPerStep = tempo em milissegundos para o
movimento de cada pixel
        public void MoveMouseUp(int PixelsDistance, int DelayPerStep)
        {
            int i;

            for (i = 0; i < PixelsDistance; i++)
            {
                Pointer.X = GetMousePositionX();
                Pointer.Y = GetMousePositionY() - 1;
                Cursor.Position = Pointer;
                System.Threading.Thread.Sleep(DelayPerStep);
            }
        }

        //função que move o mouse para a baixo.
```

```
        //Parâmetros:   PixelsDistance = quantidade de pixels que o mouse
irá se deslocar
        //              DelayPerStep = tempo em milissegundos para o
movimento de cada pixel
        public void MoveMouseDown(int PixelsDistance, int DelayPerStep)
        {
            int i;

            for (i = 0; i < PixelsDistance; i++)
            {
                Pointer.X = GetMousePositionX();
                Pointer.Y = GetMousePositionY() + 1;
                Cursor.Position = Pointer;
                System.Threading.Thread.Sleep(DelayPerStep);
            }
        }

        //função que emula um clique com o botão esquerdo do mouse.
        public void MouseLeftClick()
        {
            Pointer.X = GetMousePositionX();
            Pointer.Y = GetMousePositionY();
            mouse_event(MOUSEEVENTF_LEFTDOWN | MOUSEEVENTF_LEFTUP,
Pointer.X, Pointer.Y, 0, 0);
        }

        //função que emula um duplo clique com o botão esquerdo do mouse
        public void MouseDoubleLeftClick()
        {
            Pointer.X = GetMousePositionX();
            Pointer.Y = GetMousePositionY();
            mouse_event(MOUSEEVENTF_LEFTDOWN | MOUSEEVENTF_LEFTUP,
Pointer.X, Pointer.Y, 0, 0);
            mouse_event(MOUSEEVENTF_LEFTDOWN | MOUSEEVENTF_LEFTUP,
Pointer.X, Pointer.Y, 0, 0);
        }

        //função que emula um clique com o botão direito
        public void MouseRightClick()
        {
            Pointer.X = GetMousePositionX();
            Pointer.Y = GetMousePositionY();
            mouse_event(MOUSEEVENTF_RIGHTDOWN | MOUSEEVENTF_RIGHTUP,
Pointer.X, Pointer.Y, 0, 0);
        }
    }
}
```

ANEXO C – CLASSE QUE IMPLEMENTA A LÓGICA DE COMANDOS

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace MouseControl
{
    class IEEMouseLogic : IEEMouseFunctions
    {
        private string MouseMainAction;
        private string MouseGenericAction;
        private int Pixels;
        private int Delay;

        //Constantes que definem os códigos da ação a ser tomada pelo mouse
virtual
        private const string MoveLeft = "ML";
        private const string MoveRight = "MR";
        private const string MoveUp = "MU";
        private const string MoveDown = "MD";
        private const string LeftClick = "LC";
        private const string DoubleLeftClick = "2LC";
        private const string RightClick = "RC";

        //função de define as configurações de funcionamento do mouse
        //Parâmetros:   PixelsDistance = quantidade de pixels que o mouse
irá se deslocar
        //              DelayPerStep = tempo em milissegundos para o
movimento de cada pixel
        public void DefineMouseParameters(int PixelsDistance, int
DelayPerStep)
        {
            Pixels = PixelsDistance;
            Delay = DelayPerStep;
        }

        //retorna código da ação principal atual
        public string GetMainAction()
        {
            return MouseMainAction;
        }

        //retorna código da ação genérica atual
        public string GetGenericAction()
        {
            return MouseGenericAction;
        }

        //função de define a função genérica de funcionamento do mouse
        //Parâmetros:   MouseAction = código da função a ser utilizada
        public void SelectMouseGenericAction(string MouseAction)
        {
            switch (MouseAction)
            {
                case MoveLeft:
                    MouseGenericAction = MoveLeft;
                    break;

                case MoveRight:
```

```
                    MouseGenericAction = MoveRight;
                    break;

                case MoveUp:
                    MouseGenericAction = MoveUp;
                    break;

                case MoveDown:
                    MouseGenericAction = MoveDown;
                    break;

                case LeftClick:
                    MouseGenericAction = LeftClick;
                    break;

                case DoubleLeftClick:
                    MouseGenericAction = DoubleLeftClick;
                    break;

                case RightClick:
                    MouseGenericAction = RightClick;
                    break;

                default:
                    break;
            }
        }

        //função que define ação genérica do mouse. Não permite que a ação
genérica seja igual a ação principal
        public void SelectMouseGenericAction()
        {
            if (MouseGenericAction == null)
            {
                MouseGenericAction = LeftClick;
            }
            else
            {
                switch (MouseGenericAction)
                {
                    case MoveLeft:
                        if (MouseMainAction != MoveRight)
                            MouseGenericAction = MoveRight;
                        else
                            MouseGenericAction = MoveUp;
                        break;

                    case MoveRight:
                        if (MouseMainAction != MoveUp)
                            MouseGenericAction = MoveUp;
                        else
                            MouseGenericAction = MoveDown;
                        break;

                    case MoveUp:
                        if (MouseMainAction != MoveDown)
                            MouseGenericAction = MoveDown;
                        else
                            MouseGenericAction = LeftClick;
                        break;
```

```
                case MoveDown:
                    if (MouseMainAction != LeftClick)
                        MouseGenericAction = LeftClick;
                    else
                        MouseGenericAction = DoubleLeftClick;
                    break;

                case LeftClick:
                    if (MouseMainAction != DoubleLeftClick)
                        MouseGenericAction = DoubleLeftClick;
                    else
                        MouseGenericAction = RightClick;
                    break;

                case DoubleLeftClick:
                    if (MouseMainAction != RightClick)
                        MouseGenericAction = RightClick;
                    else
                        MouseGenericAction = MoveLeft;
                    break;

                case RightClick:
                    if (MouseMainAction != MoveLeft)
                        MouseGenericAction = MoveLeft;
                    else
                        MouseGenericAction = MoveRight;
                    break;

                default:
                    break;
            }
        }
    }

    //função que executa a função genérica de funcionamento do mouse
    public void ExecuteGenericAction()
    {
        switch (MouseGenericAction)
        {
            case MoveLeft:
                MoveMouseLeft(Pixels,Delay);
                break;

            case MoveRight:
                MoveMouseRight(Pixels,Delay);
                break;

            case MoveUp:
                MoveMouseUp(Pixels,Delay);
                break;

            case MoveDown:
                MoveMouseDown(Pixels, Delay);
                break;

            case LeftClick:
                MouseLeftClick();
                break;

            case DoubleLeftClick:
                MouseDoubleLeftClick();
```

```
            break;

        case RightClick:
            MouseRightClick();
            break;

        default:
            break;
    }
}

//função de define a função principal de funcionamento do mouse
//Parâmetros:   MouseAction = código da função a ser utilizada
public void DefineMainAction(string MouseAction)
{
    switch (MouseAction)
    {
        case MoveLeft:
            MouseMainAction = MoveLeft;
            break;

        case MoveRight:
            MouseMainAction = MoveRight;
            break;

        case MoveUp:
            MouseMainAction = MoveUp;
            break;

        case MoveDown:
            MouseMainAction = MoveDown;
            break;

        case LeftClick:
            MouseMainAction = LeftClick;
            break;

        case DoubleLeftClick:
            MouseMainAction = DoubleLeftClick;
            break;

        case RightClick:
            MouseMainAction = RightClick;
            break;

        default:
            break;
    }
}

//função de executa a função principal de funcionamento do mouse
public void ExecuteMainAction()
{
    switch (MouseMainAction)
    {
        case MoveLeft:
            MoveMouseLeft(Pixels, Delay);
            break;

        case MoveRight:
            MoveMouseRight(Pixels, Delay);
```

```
                    break;

                case MoveUp:
                    MoveMouseUp(Pixels, Delay);
                    break;

                case MoveDown:
                    MoveMouseDown(Pixels, Delay);
                    break;

                case LeftClick:
                    MouseLeftClick();
                    break;

                case DoubleLeftClick:
                    MouseDoubleLeftClick();
                    break;

                case RightClick:
                    MouseRightClick();
                    break;

                default:
                    break;
            }
        }
    }
}
```

ANEXO D – ROTINA PRINCIPAL DO *SOFTWARE*

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using NationalInstruments.DAQmx;

namespace MouseControl
{
    public partial class MainAppForm : Form
    {
        public MainAppForm()
        {
            InitializeComponent();
            InitializeDataReader();
            FormConfiguration();
            SetupSystem();
        }


////////////////////////////////////////////////////////////////////////////
//////////////////////VARIAVEIS DE CONTROLE DO SISTEMA//////////////////////
////////////////////////////////////////////////////////////////////////////

        //config padrão do sistema
        private const int NumberOfDaqSamples = 250;
        private const int SamplesRate = 1000;
        private const int SizeOfPacket = 250;
        private const double p = 0.5;
        private const double ThresholdLevel = 0.9;
        private int SetupTime = NumberOfDaqSamples;
        private const string ThresholdType = "LED"; //LED, ALED ou STATIC
        private const double StaticThresholdMargin = 1.5;
        private const int NumberOfTests = 1;
        private const int MousePx = 15;
        private const int MouseDelay = 10;
        private const string MouseMainAction = "LC"; //ML, MR, MU, MD, LC,
2LC ou RC

        //calibracao
        private int SetupTimeTimer = 1000;
        private int SetupTimeInterface = 5;
        double offsetRM = 0;
        double offsetRT = 0;
        double offsetLM = 1.4;
        double offsetLT = -1.4;

        private string SetupStatus;
        private const string SetupLeftSideWaitStatus = "LSW";
        private const string SetupLeftSideStartStatus = "LSS";
        private const string SetupRightSideWaitStatus = "RSW";
        private const string SetupRightSideStartStatus = "RSS";
        private const string SetupCompletedStatus = "SC";
        ArrayList LMSetupData = new ArrayList();
        ArrayList RMSetupData = new ArrayList();
```

```
        ArrayList LTSetupData = new ArrayList();
        ArrayList RTSetupData = new ArrayList();
        ArrayList SetupThresholds = new ArrayList();

        //sinal
        IEESignalFunctions DataSignal = new IEESignalFunctions();
        ArrayList LMData = new ArrayList();
        ArrayList RMData = new ArrayList();
        ArrayList LTData = new ArrayList();
        ArrayList RTData = new ArrayList();
        ArrayList EncapsulatedData = new ArrayList();
        private const string LeftMasseterSignal = "LMS";
        private const string RightMasseterSignal = "RMS";
        private const string LeftTemporalSignal = "LTS";
        private const string RightTemporalSignal = "RTS";

        //coleta de pacotes
        private Task ReadDataTask;
        private AnalogMultiChannelReader DataReader;
        private string[] AvailableChannels; //Conexões -> AI0 = Masseter
esquerdo, AI1 = masseter direito, AI4 = temporal esquerdo, AI5 = temporal
direito
        private double VoltageMinValue = -5;
        private double VoltageMaxValue = 5;
        private bool DaqStatus;

        //mouse
        IEEMouseLogic VirtualMouse = new IEEMouseLogic();

        //limiares
        private double LeftSideThreshold;
        private double RightSideThreshold;
        private bool LeftSideStatus;
        private bool RightSideStatus;


////////////////////////////////////////////////////////////////////////
//////////////////FUNCOES DE INICIALIZAÇÃO DO SISTEMA//////////////////
////////////////////////////////////////////////////////////////////////

        // funcao que ajusta a posição inicial do software e ajuste ele
como sempre visivel
        private void FormConfiguration()
        {
            this.TopMost = true;
            int ScreenHeight;
            int ScreenWidth;
            ScreenHeight = Screen.PrimaryScreen.Bounds.Height;
            ScreenWidth = Screen.PrimaryScreen.Bounds.Width;
        }

        //funcao que configura o DAQ para que os sinais possam ser
coletados
        public void InitializeDataReader()
        {
            try
            {

                //cria nova task para aquisicao dos dados
                ReadDataTask = new Task();
```

```
                //verifica canais existentes no DAQ
                AvailableChannels =
DaqSystem.Local.GetPhysicalChannels(PhysicalChannelTypes.AI,
PhysicalChannelAccess.External);

                //Cria os canais a serem lidos

ReadDataTask.AIChannels.CreateVoltageChannel(AvailableChannels[0],
"Masseter Esquerdo", (AITerminalConfiguration)(-1), VoltageMinValue,
VoltageMaxValue, AIVoltageUnits.Volts);

ReadDataTask.AIChannels.CreateVoltageChannel(AvailableChannels[1],
"Masseter Direito", (AITerminalConfiguration)(-1), VoltageMinValue,
VoltageMaxValue, AIVoltageUnits.Volts);

ReadDataTask.AIChannels.CreateVoltageChannel(AvailableChannels[4],
"Temporal Esquerdo", (AITerminalConfiguration)(-1), VoltageMinValue,
VoltageMaxValue, AIVoltageUnits.Volts);

ReadDataTask.AIChannels.CreateVoltageChannel(AvailableChannels[5],
"Temporal Direito", (AITerminalConfiguration)(-1), VoltageMinValue,
VoltageMaxValue, AIVoltageUnits.Volts);

                //cria um leitor de N canais virtuais
                DataReader = new
AnalogMultiChannelReader(ReadDataTask.Stream);

                //verifica se todos os parametros da task sao validos
                ReadDataTask.Control(TaskAction.Verify);
                DaqStatus = true;
            }
            catch (DaqException e)
            {
                MessageBox.Show(e.Message);
                DaqStatus = false;
            }
        }

        //funcao configura as classes de mouse e dsp
        private void SetupSystem()
        {
            DataSignal.setSamplesRate(SamplesRate);
            DataSignal.setSizeOfPacket(SizeOfPacket);
            DataSignal.CalculateNumberOfSamples();

            VirtualMouse.DefineMouseParameters(MousePx, MouseDelay);
            VirtualMouse.DefineMainAction(MouseMainAction);

            SetupTimer.Interval = SetupTime;

            SetupStatus = SetupLeftSideWaitStatus;
            SetupLeftSideWait();
        }


////////////////////////////////////////////////////////////////////////
//////////////////////////FUNCOES DA INTERFACE COM USUARIO//////////////////
////////////////////////////////////////////////////////////////////////

        private void SetupLeftSideWait()
        {
```

```
            lblMouseFunction.Text = "Se prepare para calibrar o masseter
esquerdo...";
            RunInterfaceTimer();
        }

        private void SetupLeftSideStart()
        {
            lblMouseFunction.Text = "Faça uma contração com o masseter
esquerdo...";
            RunInterfaceTimer();
            SetupTimer.Start();
        }

        private void SetupRightSideWait()
        {
            lblMouseFunction.Text = "Se prepare para calibrar o masseter
direito...";
            RunInterfaceTimer();
            SetupLeftSide(); //calcula o limiar do lado esquerdo
        }

        private void SetupRightSideStart()
        {
            lblMouseFunction.Text = "Faça uma contração com o masseter
direito...";
            RunInterfaceTimer();
            SetupTimer.Start();
        }

        private void SetupCompleted()
        {
            lblMouseFunction.Text = "Sistema calibrado. Aguarde...";
            //RunInterfaceTimer();
            SetupRightSide();
            //SetupBothSides(); //calcula o limiar de ambos os lados
        }

        private void SetupDataClear()
        {
            LMSetupData.Clear();
            RMSetupData.Clear();
            LTSetupData.Clear();
            RTSetupData.Clear();
            SetupThresholds.Clear();
        }

        //timer para controle das mensagens e do cronometro na interface
        private void RunInterfaceTimer()
        {
            lblTimer.Text = SetupTimeInterface.ToString();
            InterfaceTimer.Interval = SetupTimeTimer;
            InterfaceTimer.Start();
        }

        private void timer_Tick(object sender, EventArgs e)
        {
            int CurrentTime;
            CurrentTime = System.Convert.ToInt32(lblTimer.Text);
            CurrentTime = CurrentTime - 1;
            if (CurrentTime >= 0)
                lblTimer.Text = CurrentTime.ToString();
```

```
            else
            {
                InterfaceTimer.Stop();
                SetupChangeStatus();
            }
        }

        private void SetupChangeStatus()
        {
            switch (SetupStatus)
            {
                case SetupLeftSideWaitStatus:
                    SetupStatus = SetupLeftSideStartStatus;
                    SetupLeftSideStart();
                    break;

                case SetupLeftSideStartStatus:
                    SetupTimer.Stop();
                    SetupStatus = SetupRightSideWaitStatus;
                    SetupRightSideWait();
                    break;

                case SetupRightSideWaitStatus:
                    SetupStatus = SetupRightSideStartStatus;
                    SetupRightSideStart();
                    break;

                case SetupRightSideStartStatus:
                    SetupTimer.Stop();
                    SetupStatus = SetupCompletedStatus;
                    SetupCompleted();
                    break;
            }
        }



////////////////////////////////////////////////////////////////////////
////////////////////FUNCOES PARA CALIBRACAO DO SISTEMA/////////////////
////////////////////////////////////////////////////////////////////////

        private void SetupLeftSide()
        {
            int i;
            int j;
            int Step = 0;
            int k;
            int NumberOfIterations;
            double LT;
            double LM;

            NumberOfIterations = LMSetupData.Count /
(DataSignal.getNumberOfSamples());

            for (i = 0; i < NumberOfIterations; i++)
            {
                LMData.Clear();
                LTData.Clear();
                EncapsulatedData.Clear();
```

```
                k = 0;
                for (j = Step; j < ((LMSetupData.Count /
NumberOfIterations) + Step); j++)
                {
                    LM = System.Convert.ToDouble(LMSetupData[j]) +
offsetLM;
                    LT = System.Convert.ToDouble(LTSetupData[j]) +
offsetLT;

                    LMData.Add(LM);
                    LTData.Add(LT);

EncapsulatedData.Add(Math.Abs(System.Convert.ToDouble(LMData[k])) +
Math.Abs(System.Convert.ToDouble(LTData[k])));
                    k++;
                }
                CalculatePacketThreshold();
                Step = Step + LMSetupData.Count / NumberOfIterations;
            }

            LeftSideThreshold = SelectThreshold(NumberOfTests,
LMSetupData.Count, SetupThresholds, ThresholdLevel);
            SetupDataClear();
        }

        private void SetupRightSide()
        {
            int i;
            int j;
            int Step = 0;
            int k;
            int NumberOfIterations;
            double RT;
            double RM;

            NumberOfIterations = RMSetupData.Count /
(DataSignal.getNumberOfSamples());

            for (i = 0; i < NumberOfIterations; i++)
            {
                RMData.Clear();
                RTData.Clear();
                EncapsulatedData.Clear();

                k = 0;
                for (j = Step; j < ((RMSetupData.Count /
NumberOfIterations) + Step); j++)
                {
                    RM = System.Convert.ToDouble(RMSetupData[j]) +
offsetRM;
                    //RMData.Add(RMSetupData[j]);
                    RT = System.Convert.ToDouble(RTSetupData[j]) +
offsetRT;

                    RMData.Add(RM);
                    RTData.Add(RT);

EncapsulatedData.Add(Math.Abs(System.Convert.ToDouble(RMData[k])) +
Math.Abs(System.Convert.ToDouble(RTData[k])));
```

```
                    k++;
                }
                CalculatePacketThreshold();
                Step = Step + LMSetupData.Count / NumberOfIterations;
            }

            RightSideThreshold = SelectThreshold(NumberOfTests,
LMSetupData.Count, SetupThresholds, ThresholdLevel);
            SetupDataClear();
            RunApplication();
        }

        //funcao que seleciona o limiar
        public double SelectThreshold(int NumberOfTests, int
NumberOfSamplesPerTest, ArrayList Samples, double ThresholdLevel)
        {
            int i = 0;
            int NumberOfSamples;
            int StartPosition;
            double Threshold = new double();

            NumberOfSamples = NumberOfTests * NumberOfSamplesPerTest;

            Samples.Sort();

            StartPosition =
System.Convert.ToInt32(Math.Truncate(ThresholdLevel * Samples.Count));

            while (i < NumberOfTests)
            {
                Threshold = System.Convert.ToDouble(Samples[StartPosition])
+ Threshold;
                StartPosition++;
                i++;
            }

            Threshold = Threshold / NumberOfTests;

            return Threshold;
        }

        //funcao que calcula o limiar para cada pacote no processo de
calibracao
        private void CalculatePacketThreshold()
        {
            DataSignal.setPacketContent(EncapsulatedData);

            switch (ThresholdType)
            {
                case "LED":
                    DataSignal.ApplyLEDTechnique(p);
                    SetupThresholds.Add(DataSignal.getLEDThreshold());
                    break;
                case "ALED":
                    DataSignal.ApplyALEDTechnique();
                    SetupThresholds.Add(DataSignal.getALEDThreshold());
                    break;
                case "STATIC":

DataSignal.setStaticThreshold(DataSignal.getPacketEnergy(),
StaticThresholdMargin);
```

```
                SetupThresholds.Add(DataSignal.getStaticThreshold());
                break;
            }
        }

        //funcao que le dados dos canais do DAQ
        private void SetupTimer_Tick(object sender, EventArgs e)
        {
            try
            {
                double[,] DaqData;
                int i;

                DaqData = DataReader.ReadMultiSample(NumberOfDaqSamples);

                for (i = 0; i < NumberOfDaqSamples; i++)
                {
                    LMSetupData.Add(DaqData[0, i]);
                    RMSetupData.Add(DaqData[1, i]);
                    LTSetupData.Add(DaqData[2, i]);
                    RTSetupData.Add(DaqData[3, i]);
                }

            }
            catch (DaqException exception)
            {
                GetPacketsTimer.Enabled = false;
                ReadDataTask.Dispose();
                MessageBox.Show(exception.Message);
            }
        }


////////////////////////////////////////////////////////////////////////////
///////////////FUNCOES PARA ANALISE DO SINAL EM TEMPO REAL//////////////////
////////////////////////////////////////////////////////////////////////////

        //funcao que inicializa a coleta de dados apos a calibracao
        private void RunApplication()
        {
            RealTimeDataClear();
            GetPacketsTimer.Interval = SetupTime;
            GetPacketsTimer.Start();
        }

        private void RealTimeDataClear()
        {
            LMData.Clear();
            RMData.Clear();
            LTData.Clear();
            RTData.Clear();
        }

        //funcao que compara a energia do masseter e temporal direito com o
limiar
        private void LeftSideAnalysis()
        {
            int i;
            EncapsulatedData.Clear();

            for (i = 0; i < DataSignal.getNumberOfSamples(); i++)
```

```
            {
EncapsulatedData.Add(Math.Abs(System.Convert.ToDouble(LMData[i])) +
Math.Abs(System.Convert.ToDouble(LTData[i])));
            }

            DataSignal.setPacketContent(EncapsulatedData);

            if (DataSignal.getPacketEnergy() >= LeftSideThreshold)
                LeftSideStatus = true;
            else
                LeftSideStatus = false;
        }

        //funcao que compara a energia do masseter e temporal direito com o
limiar
        private void RightSideAnalysis()
        {
            int i;
            EncapsulatedData.Clear();

            for (i = 0; i < DataSignal.getNumberOfSamples(); i++)
            {
EncapsulatedData.Add(Math.Abs(System.Convert.ToDouble(RMData[i])) +
Math.Abs(System.Convert.ToDouble(RTData[i])));
            }

            DataSignal.setPacketContent(EncapsulatedData);

            if (DataSignal.getPacketEnergy() >= RightSideThreshold)
                RightSideStatus = true;
            else
                RightSideStatus = false;
        }

        //funcao de analise em tempo real do sinal
        private void GetPacketsTimer_Tick(object sender, EventArgs e)
        {
            double RT;
            double RM;
            double LT;
            double LM;


            try
            {
                double[,] DaqData;
                int i;

                DaqData = DataReader.ReadMultiSample(NumberOfDaqSamples);

                for (i = 0; i < NumberOfDaqSamples; i++)
                {
                    LM = System.Convert.ToDouble(DaqData[0, i]) + offsetLM;
                    RM = System.Convert.ToDouble(DaqData[1, i]) + offsetRM;
                    LT = System.Convert.ToDouble(DaqData[2, i]) + offsetLT;
                    RT = System.Convert.ToDouble(DaqData[3, i]) + offsetRT;

                    LMData.Add(LM);
                    RMData.Add(RM);
```

```
                    LTData.Add(LT);
                    RTData.Add(RT);
                }

                if (LMData.Count == DataSignal.getNumberOfSamples())
                {
                    LeftSideAnalysis();
                    RightSideAnalysis();
                    SelectMouseAction();
                    RealTimeDataClear();
                }

            }
            catch (DaqException exception)
            {
                GetPacketsTimer.Enabled = false;
                ReadDataTask.Dispose();
                MessageBox.Show(exception.Message);
            }
        }


////////////////////////////////////////////////////////////////////////
//////////////////////FUNCOES DE CONTROLE DO MOUSE//////////////////////
////////////////////////////////////////////////////////////////////////

        //funcao que define a acao do mouse a ser executada
        private void SelectMouseAction()
        {
            if ((LeftSideStatus == true) && (RightSideStatus == true))
            {
                ExecuteMouseAction(1);
                return;
            }

            if ((LeftSideStatus == false) && (RightSideStatus == true))
            {
                ExecuteMouseAction(2);
                return;
            }

            if ((LeftSideStatus == true) && (RightSideStatus == false))
            {
                ExecuteMouseAction(3);
                return;
            }

        }

        //funcao que executa a acao do mouse a selecionada
        private void ExecuteMouseAction(int command)
        {
            switch (command)
            {
                case 1:
                    ExecuteCommand1();
                    break;
                case 2:
                    ExecuteCommand2();
                    break;
                case 3:
```

```
                    ExecuteCommand3();
                    break;
            }
        }

        //funcao que seleciona uma acao
        private void ExecuteCommand1()
        {
            VirtualMouse.SelectMouseGenericAction();
            switch (VirtualMouse.GetGenericAction())
            {
                case "ML":
                    lblMouseFunction.Text = "Mover para esquerda...";
                    break;
                case "MR":
                    lblMouseFunction.Text = "Mover para direita...";
                    break;
                case "MU":
                    lblMouseFunction.Text = "Mover para cima...";
                    break;
                case "MD":
                    lblMouseFunction.Text = "Mover para baixo...";
                    break;
                case "LC":
                    lblMouseFunction.Text = "Clique com o botão
esquerdo...";
                    break;
                case "2LC":
                    lblMouseFunction.Text = "Duplo clique com botão
esquerdo...";
                    break;
                case "RC":
                    lblMouseFunction.Text = "Clique com botão direito...";
                    break;
            }
        }

        //funcao que executa a acao selecionada
        private void ExecuteCommand2()
        {
            VirtualMouse.ExecuteGenericAction();
        }

        //funcao que executa a acao principal do mouse
        private void ExecuteCommand3()
        {
            VirtualMouse.ExecuteMainAction();
        }


////////////////////////////////////////////////////////////////////////////
//////////////////////////FUNCOES DO MENU DA APLICACAO//////////////////////
////////////////////////////////////////////////////////////////////////////

        //funcao que abre a aplicacao para teste do mouse
        private void mouseToolStripMenuItem_Click(object sender, EventArgs
e)
        {
            MouseTest MouseTester = new MouseTest();
            MouseTester.Show();
        }
```

```
        //funcao que abre a aplicacao para teste do tratamento do sinal
        private void tratamentoDoSinalToolStripMenuItem_Click(object
sender, EventArgs e)
        {
            SignalFunctionTest SignalTester = new SignalFunctionTest();
            SignalTester.Show();
        }

        //funcao que abre a tela de opcoes
        private void sairToolStripMenuItem1_Click(object sender, EventArgs
e)
        {
            Options OptionsForm = new Options();
            OptionsForm.Show();
        }

        //funcao que fecha o programa
        private void sairToolStripMenuItem_Click(object sender, EventArgs
e)
        {
            Application.Exit();
        }


    }
}
```