CENTRO UNIVERSITÁRIO DE BRASÍLIA - UniCEUB
CURSO DE ENGENHARIA DE COMPUTAÇÃO

TOMÁS DA SILVA MARTINS DE GODOI

PRÓTESE MIOELÉTRICA CONTROLADA POR REDES NEURAIS

Orientador: Msc. Prof. Francisco Javier de Obaldía Díaz

Brasília
Junho, 2013

TOMÁS DA SILVA MARTINS DE GODOI

PRÓTESE MIOELÉTRICA CONTROLADA POR REDES NEURAIS

Trabalho apresentado ao Centro Universitário de Brasília (UniCEUB) como pré-requisito para a obtenção de Certificado de Conclusão de Curso de Engenharia de Computação.
Orientador: Msc. Prof. Francisco Javier de Obaldía Díaz

Brasília
Junho, 2013

TOMÁS DA SILVA MARTINS DE GODOI

PRÓTESE MIOELÉTRICA CONTROLADA POR REDES NEURAIS

Trabalho apresentado ao Centro Universitário de Brasília (UniCEUB) como pré-requisito para a obtenção de Certificado de Conclusão de Curso de Engenharia de Computação.
Orientador: Msc. Prof. Francisco Javier de Obaldía Díaz

Este Trabalho foi julgado adequado para a obtenção do Título de Engenheiro de Computação, e aprovado em sua forma final pela Faculdade de Tecnologia e Ciências Sociais Aplicadas - FATECS.

Msc. Prof. Abiezer Amarilia Fernandes
Coordenador do Curso

Banca examinadora:

Msc. Prof. Francisco Javier de Obaldía Díaz
Orientador

Msc. Prof. Luciano Henrique Duque
Co-orientador

Msc. Prof. Marco Antonio Araújo
Convidado 1

Msc. Prof. Fabiano Mariath D'Oliveira
Convidado 2

Dedico este trabalho à minha família, aos meus professores e amigos, por terem me apoiado nesses anos.

Agradecimentos

Agredeço primeiramente a Deus, que me deu força nos momentos de dificuldade para superar os obstáculos. Agradeço aos meus pais pelo incentivo ao estudo e pelo apoio financeiro em grande parte do curso. Agradeço também aos meus colegas de turma, pelo companheirismo durante todo o curso e cujas ideias também me ajudaram a finalizar este projeto. Agradeço aos meus professores, pela paciência e pela partilha do conhecimento. Em especial, agradeço ao Prof. Luciano Henrique Duque, que como orientador do Projeto de Iniciação Científica, ajudou-me com ideias para o Projeto Final e deu suporte em todas as etapas. Agradeço aos integrantes do Grupo de Engenharia de Reabilitação (GER-UniCEUB) cuja ajuda na pesquisa foi essencial para o desenvolvimento deste trabalho. Também agradeço aos Profs. Luis Claudio Lopes de Araujo e Fabiano Mariath D'Oliveira, pela orientação na escrita de artigos durante o curso. Agradeço ao José Carlos, pela ajuda em algumas questões técnicas do projeto. Agradeço a meu orientador, Prof. Francisco Javier de Obaldía, cuja orientação foi fundamental para o desenvolvimento deste projeto.

Sumário

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

Amp-op	Amplificador operacional
Ag	Prata
AR	Modelagem Auto-Regressiva
APF	Análise de Pontos de Função
ATP	Trifosfato de Adenosina
Ca	Cálcio
CI	Circuito Integrado
Cl	Cloro
DC	Corrente Contínua
DFT	Transformada Discreta de Fourier
DIP	*Dual In-Line Package*
DSP	Processador Digital de Sinais
EMG	Eletromiografia
FFT	Transformada Rápida de Fourier
IDFT	Transformada Discreta de Fourier Inversa
IEMG	EMG Integrado
IMAV	Inclinação do Valor Absoluto Médio
K	Potássio
MAV	Valor Absoluto Médio
MDF	Frequência Mediana
MFAP	Potencial de Ação da Fibra Muscular
MNF	Frequência Média
MUAP	Potencial de Ação da Unidade Motora

Na	Sódio
RNA	Rede Neural Artificial
SEMG	Sinal eletromiográfico
SSC	Mudança no Sinal da Inclinação
RRMC	Razão de Rejeição de Modo Comum
WL	Tamanho da Forma de Onda
XML	*eXtensible Markup Language*
ZC	Cruzamento de Zero

Resumo

Os sinais eletromiográficos são sinais oriundos da atividade muscular e têm sido objeto de diversos estudos nos últimos anos. Diversas aplicações em áreas como fisioterapia, educação física, odontologia já foram documentadas. Uma das aplicações de grande interesse desses sinais consiste na sua utilização como entrada de um mecanismo de controle de próteses. Este trabalho continua uma pesquisa iniciada no Grupo de Engenharia de Reabilitação - UniCEUB. Ele trata do desenvolvimento de um sistema de aquisição, amplificação, filtragem e processamento digital dos sinais eletromiográficos com o objetivo de controle de uma prótese mioelétrica virtual. Na parte de *hardware*, utiliza-se circuitos eletrônicos com amplificadores operacionais e filtros para o condicionamento adequado desse sinal. Na parte de *software*, emprega-se a computação gráfica 3D para a animação de uma mão virtualizada e o processamento digital de sinais utilizando a Transformada Rápida de Fourier em conjunto com Redes Neurais Artificiais do tipo percéptron de múltipla camada para o controle de uma prótese mioelétrica virtualizada.

Palavras-chaves: sinais eletromiográficos. filtros. processamento digital de sinais. transformada rápida de Fourier. prótese virtual. redes neurais artificiais.

Abstract

The electromyographic signals are signals from the muscular activity, and they have been studied by several researchers in the last years. Several applications in fields like physiotherapy, physical education and odontology, have already been documented. One of these applications is the use of the electromyographic signal as the input of prosthetic control systems. This paper continues a research started in the group "Grupo de Engenharia de Reabilitação - UniCEUB". It aims to develop a system which will properly aquire, amplify, filter and digitally proccess the electromyographic signals to control a virtual myoelectric prosthesis. On its hardware components, it has been used electronic circuits with operational amplifiers and filters to properly aquire this signal. On its software components, it has been used 3D computer graphics to animate a virtual hand, and digital processing using the Fast Fourier Transform and Multilayer Perceptron Artificial Neural Networks to control the virtual myoelectric prosthesis.

Key-words: electromyographic signals. filters. digital signal processing. fast Fourier transform. virtual prosthesis. artificial neural networks.

1 Introdução

1.1 Apresentação do Problema

O processamento dos sinais biomédicos é um campo no qual diversas pesquisas têm sido realizadas. Várias técnicas têm sido utilizadas com sucesso para extrair informações úteis desses sinais. As possíveis aplicações são inúmeras, desde o diagnóstico de doenças até o desenvolvimento de próteses.

Entretanto, para conseguir extrair informações úteis desses sinais com sucesso, vários problemas de engenharia devem ser resolvidos. O primeiro problema é o da aquisição apropriada e precisa dos sinais biomédicos. Tais sinais, além de serem de baixa amplitude, sofrem das interferências presentes no ambiente e no próprio corpo humano.

O segundo problema é o do processamento do sinal biomédico para extração de informação útil. Para isso, diversas técnicas de processamento digital de sinais podem ser utilizadas. Os médicos podem ser capazes de chegar a conclusões sobre um paciente ao analisar um sinal biomédico.

Em particular, um sinal biomédico de grande interesse é o sinal eletromiográfico (SEMG), oriundo da atividade elétrica do músculo, já que a sua análise pode ter propósitos de diagnóstico ou desenvolvimento de próteses.

A proposta deste projeto é resolver esses dois problemas centrais e os problemas a eles correlatos, demonstrando por meio de um protótipo de prótese virtual uma aplicação que realiza o controle de um membro superior a partir de sinais biomédicos capturados do músculo de uma pessoa.

A questão de pesquisa formulada para este trabalho é: quais são as técnicas de processamento digital de sinais mais adequadas para processar SEMGs para o controle de uma prótese de membro superior?

1.2 Objetivos do Trabalho

1.2.1 Objetivo Geral

O objetivo deste trabalho é construir um protótipo de prótese mioelétrica de mão simulada em *software* e controlada por rede neural artificial.

1.2.2 Objetivos Específicos

Para atingir o objetivo geral, os seguintes objetivos específicos são identificados:

a) Desenvolver um protótipo de circuito que realize a captura, filtragem, amplificação e transmissão para meio digital dos sinais eletromiográficos. Tais operações deverão ser precisas o suficiente para não prejudicar o processamento digital desses sinais.

b) Desenvolver uma interface no computador que receba os sinais transmitidos pelo protótipo de circuito e construa uma visualização gráfica desse sinal, para fins de diagnóstico pelo médico (fisioterapeuta) de patologias e verificação do processo de reabilitação do paciente.

c) Desenvolver um *software* que processe digitalmente o sinal recebido no computador e modele uma mão virtual, buscando assim detectar os padrões de movimento feito pelo paciente e reproduzir no modelo virtual.

1.3 Justificativa e Importância do Trabalho

A captura e processamento dos sinais eletromiográficos tem uma série de aplicações que têm ganhado popularidade. Uma delas é a utilização do *biofeedback* eletromiográfico nos ambientes clínicos, que auxilia os pacientes a desenvolver um controle voluntário maior durante a reeducação muscular após uma lesão. (PRENTICE, 2008, Cap. 7)

Outra aplicação do processamento desses sinais é o desenvolvimento das denominadas próteses mioelétricas: próteses que utilizam o sinal eletromiográfico como entrada de um mecanismo de controle. O desenvolvimento de próteses cada vez mais precisas está sendo possível graças ao aumento do poder computacional e à descoberta de novos mecanismos de processamento desses sinais.

As próteses mioelétricas tem uma grande importância social, pois ajudam pessoas amputadas a se reintegrar na sociedade e voltar a realizar as tarefas do cotidiano que a amputação havia tornado quase impossível.

Tendo isso em consideração, este trabalho visa desenvolver um protótipo de prótese virtual com foco no processamento digital dos sinais eletromiográficos, tendo em vista explorar técnicas de inteligência artificial para o processamento desses sinais. As técnicas de inteligência artificial trazem uma abordagem diferente das tradicionais, de forma que a prótese aprende o padrão de cada pessoa ao invés de a pessoa aprender o padrão de movimento da prótese. Isso torna a fase de adaptação da pessoa com a prótese menos frustrante e os movimentos mais naturais.

1.4 Escopo do Trabalho

O protótipo desenvolvido neste trabalho tem como escopo a captura dos sinais eletromiográficos, o processamento desses sinais e o controle de uma mão virtualizada utilizando gráficos 3D. Não faz parte do escopo a confecção de uma prótese física, limitando-se portanto ao controle de um modelo 3D de uma prótese. Os movimentos da prótese virtual também serão limitados aos movimentos que os testes da rede neural desenvolvida mostrar-se capaz de predizer com um grau suficiente de precisão.

A Figura 1.1 mostra o diagrama macro do projeto. Nela podem ser observadas as quatro etapas em que está dividido o projeto: Captura, Transferência, Processamento e Controle. Na etapa da Captura, amplificação e filtragem do SEMG foi desenvolvido um protótipo de circuito condicionador, utilizando dispositivos eletrônicos como amplificadores operacionais para realizar tais funções. A Transferência do SEMG para o computador consistiu no uso de um conversor analógico-digital e uma interface de entrada para o computador (entrada da placa de som ou entrada USB) para transferir o SEMG para meio digital e permitir sua captura em uma interface de *software*. A etapa de Processamento Digital do SEMG envolveu a utilização da Transformada de Fourier para facilitar a extração de características do sinal, além de redes neurais para o reconhecimento de padrões no SEMG. O Controle da prótese de mão virtualizada tratou da modelagem 3D de uma mão e a programação de movimentos nessa mão, que são controlados pela saída do mecanismo de reconhecimento de padrão da rede neural.

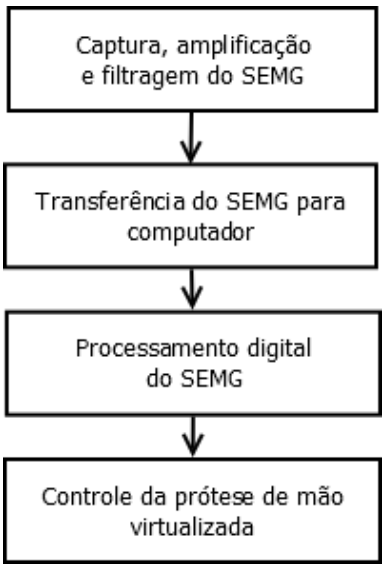


Figura 1.1 – Diagrama macro das etapas do projeto

1.5 Resultados Esperados

Com o desenvolvimento do protótipo proposto, pretende-se testar e validar a técnica de processamento digital de sinais baseada na transformada rápida de Fourier (FFT), utilizando uma rede neural artificial do tipo percéptron de múltiplas camadas como uma abordagem viável para a classificação dos sinais eletromiográficos.

Espera-se que o circuito de captura realize a amplificação e filtragem de forma a minimizar os possíveis ruídos, mantendo o sinal o mais fidedigno possível aos sinais eletromiográficos capturados em um eletromiógrafo profissional.

Do módulo de *software* que realizará o processamento dos sinais capturados, espera-se um processamento em tempo curto o suficiente para não atrasar o controle da prótese virtual. Além disso, o *software* deverá trazer o máximo de precisão no controle da prótese virtual, desde que a rede neural seja treinada de forma adequada e que os movimentos treinados sejam oriundos de músculos cuja atividade elétrica possa ser capturada pela eletromiografia de superfície.

1.6 Estrutura do Trabalho

Este trabalho está estruturado em 6 capítulos. O Capítulo 1 apresenta de forma sucinta o problema, descreve os objetivos gerais e específicos, traz a justificativa do trabalho, delimita o seu escopo e enuncia os resultados esperados. O Capítulo 2 se aprofunda no problema a ser resolvido, descrevendo o contexto no qual ele se insere e justifica a importância de sua resolução. O Capítulo 3 descreve as bases metodológicas para a resolução do problema, e se aprofunda nos aspectos técnicos que são relevantes para a resolução do problema. O Capítulo 4 apresenta o protótipo proposto para solução do problema, detalhando sua estrutura e componentes. O capítulo 5 mostra os resultados dos testes do protótipo e uma avaliação de seu desempenho. O Capítulo 6 traz as considerações finais sobre o trabalho, as dificuldades encontradas e as sugestões para trabalhos futuros.

2 Os Sinais Eletromiográficos

2.1 Histórico da medição dos sinais musculares e aplicações

O sinal eletromiográfico (SEMG) tem sua origem nos potenciais de ação das unidades motoras musculares. O primeiro relato desse sinal está no trabalho do médico italiano Francesco Redi, que em 1666 descobriu que um dos músculos do poraquê (*Electrophorus electricus*) produzia eletricidade (BASMAJIAN; DELUCA, 1985).

Uma outra evidência histórica da descoberta da relação entre a eletricidade e a contração muscular foi a pesquisa feita por Luigi Galvani que, em 1791, realizava a despolarização de pernas de rãs com a utilização de escalpelos metálicos (LUCA, 2006). A Figura 2.1 mostra o experimento de Luigi Galvani da despolarização de pernas de rãs. Seis décadas mais tarde, em 1849, Dubios-Raymond descobriu a possibilidade da gravação da atividade elétrica durante a contração muscular voluntária. Já no ano de 1890, Marey gravou com sucesso a atividade elétrica muscular e propôs o termo eletromiografia (EMG) para tal fenômeno. (IBRAHIM et al., 2008)

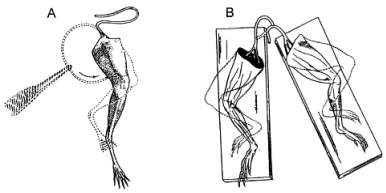


Figura 2.1 – Experimento de Luigi Galvani com pernas de rã

Fonte: Piccolino (1997)

A eletromiografia tem sido amplamente utilizada no diagnóstico clínico há mais de 45 anos. Por meio dela tem-se a origem de um valioso método de diagnóstico e de investigação, utilizado em diversas áreas como: neurologia, anátomo-fisiologia, traumato-ortopedia, odontologia, pneumologia, fisioterapia, dentre outras.

Os fisioterapeutas utilizam a eletromiografia como método de avaliação da função e disfunção do sistema neuromuscular. A eletromiografia cinestológica tem sido utilizada para o exame da função muscular durante tarefas intencionais específicas, ou regimes terapêuticos. Também tem sido empregada na avaliação da eficácia de técnicas de recuperação funcional das mais variadas patologias. (FORTI, 2005)

A eletromiografia encontra aplicações na educação física e ciências do desporto, pois permite estudar a atividade muscular em ações dinâmicas, sendo aplicável à análise biomecânica de

gestos, à análise da marcha, em estudos da fadiga muscular e do rendimento esportivo. Também incluem-se nesse rol de aplicações áreas como medicina laboral e ergonomia. (MASSó et al., 2010)

Na odontologia, uma das principais aplicações é no tratamento das Disfunções Temporomandibulares. Neste contexto, a eletromiografia colabora na análise dos músculos da mastigação e um dos objetivos é mostrar ao paciente o estado atual do músculo em análise e sua evolução durante o tratamento, até que se chegue a uma função satisfatória. Além disso, (MALTA et al., 2006) prevê outros usos possíveis da eletromiografia nas áreas de Ortodontia e Cirurgia e Traumatologia Bucomaxilofacial, ambas envolvendo a análise do SEMG como forma de diagnóstico das funções musculares.

2.1.1 SEMG como mecanismo de controle de próteses

Uma outra aplicação dos sinais eletromiográficos tem sido o seu uso como mecanismo de controle de próteses. Tais próteses são chamadas de próteses mioelétricas e utilizam pequenos circuitos que realizam a captura, filtragem e processamento dos SEMGs. Entretanto, o problema do processamento desses sinais para o controle preciso de um membro é complexo, devido à natureza estocástica dos SEMGs, e tem sido objeto de estudo de vários pesquisadores ao redor do globo nas últimas décadas.

Reiter foi o primeiro a propor, em 1948, o uso dos SEMGs para o controle de uma mão mecânica, porém tal estudo não rendeu uma implementação clínica. (CAMARGO, 2008) Durante a década de 1970, (GRAUPE; CLINE, 1975) conseguiu a partir do SEMG distinguir com sucesso 6 classes diferentes de movimento da mão, utilizando modelos autoregressivos e bayesianos, com uma taxa de sucesso próxima de 99%. Entretanto tais resultados eram obtidos apenas depois de 12 horas de treinamento e degradavam-se com o tempo, devido a modificações dos SEMGs. (ZECCA et al., 2002)

Em 1993, (HUDGINS; PARKER; SCOTT, 1993) propôs uma nova estratégia de controle para próteses mioelétricas. Em sua pesquisa, constatou-se que existe uma estrutura considerável no SEMG durante o início de uma contração. Tal estrutura é diferente para movimentos de diferentes membros e poderia ser utilizada como informação de classificação do SEMG. Eles conseguiram discriminar 4 movimentos diferentes com apenas um eletrodo bipolar e a extração das seguintes características: Valor Absoluto Médio (MAV), Inclinação do Valor Absoluto Médio (IMAV), Cruzamento de Zero (ZC), Mudança no Sinal da Inclinação (SSC), Tamanho da Forma de Onda (WL). Para a classificação, uma rede neural artificial de 2 camadas foi utilizada, entretanto a taxa de erro ainda era maior do que 10%.

Na década de 2000, os SEMGs foram estudados com os objetivos de construção de próteses mioelétricas multifuncionais e de melhoria na operação remota de dispostivos robóticos, mas ainda assim tais sistemas ainda não são capazes de controlar uma mão multifuncional. O principal problema é a variação no tempo das características do SEMG, devido a mudanças fisiológicas no músculo e mudanças na interface pele-eletrodo. Um outro poblema relevante é

o da natureza estocástica do SEMG, que dificulta a estimação precisa de parâmetros, o que causa dificuldades no controle. Além disso, também ocorrem os erros de operador, que são os erros introduzidos pela dificuldade do paciente de gerar as contrações corretas para o controle. (ZECCA et al., 2002)

A Figura 2.2 lista cronologicamente algumas pesquisas importantes referentes ao uso do SEMG como mecanismo de controle de próteses.

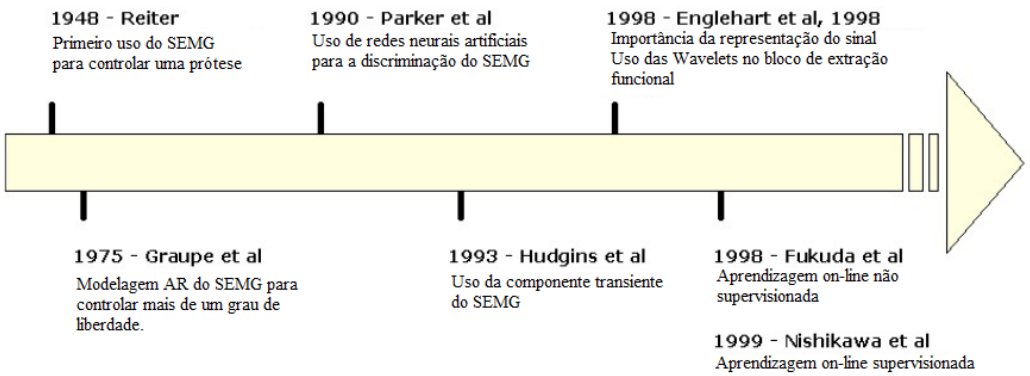


Figura 2.2 – Evolução da utilização do SEMG para controle de próteses

Fonte: Adaptado de Zecca et al. (2002)

2.2 Fisiologia dos músculos e os sinais biomédicos

O SEMG tem origem fisiológica nas fibras individuais ou grupos de fibras musculares. O entendimento das características anatômicas dessas fibras e da origem fisiológica dos potenciais de ação é necessário para a correta captura, análise e interpretação do SEMG.

Algumas dessas características são: comprimento da fibra muscular, composição da fibra, particionamento do músculo e variações na distribuição dos receptores sensoriais. Elas variam de pessoa para pessoa e portanto devem ser consideradas para uma interpretação correta do SEMG. As fibras musculares normalmente percorrem a região do tendão distal ao proximal, entretanto, algumas fibras musculares são curtas e podem estar concentradas na região distal, proximal ou central de um músculo.

O músculo é um tecido constantemente banhado em meio iônico e como todas as células, ele é cercado por uma membrana, o sarcolema, que tem uma espessura de aproximadamente 75 angstroms. Essa membrana é interrompida pelo Sistema de Túbulo T. Em alguns locais, os Túbulos T percorrem longitudinalmente, conectando-se com outros Túbulos T e com o Retículo Sarcoplasmático. Os Túbulos T são estruturas importantes para a transmissão dos potenciais de ação transversalmente até às miofibrilas para ativação de todas as partes da fibra muscular. (KAMEN; GABRIEL, 2010)

2.2.1 Condução nervosa

O neurônio motor é o responsável por comandar a contração das fibras musculares. Os neurônios são compostos de três partes: o corpo celular, os dendritos e o axônio. O corpo celular contém o citoplasma, o núcleo e as organelas e em um neurônio motor ele está entre os dendritos e o axônio. Os dendritos são prolongamentos do neurônio, altamente ramificados, de forma a oferecer amplas áreas de contato para a recepção da informação. Eles são responsáveis pelo recebimento de informações e pelo envio de estímulos para o corpo celular. Os axônios são responsáveis pela condução dos impulsos nervosos emitidos pelo corpo celular para outros neurônios ou glândulas. Na sua extremidade, os axônios ramificam-se, formando os botões sinápticos. A Figura 2.3 mostra os componentes de um neurônio. (ANDRADE, 2007)

Ao atingir os botões sinápticos, o impulso nervoso deve ser transmitido a uma outra célula, geralmente a um outro neurônio. Entretanto, as extremidades axônicas de um neurônio não tocam a superfície da outra célula. Entre a extremidade axônica e a superfície da célula vizinha existe um espaço estreito, com cerca de 10 e 50 nm, denominado sinapse nervosa, no qual são liberadas substâncias químicas denominadas neurotransmissores. Esses neurotransmissores liberados na sinapse geram um novo impulso nervoso na célula vizinha, o qual se propaga até a sinapse seguinte. Já foram identificadas mais de dez substâncias que atuam como neurotransmissores, dentre os quais se destacam: a acetilcolina, a adrenalina, a noradrenalina, a dopamina e a serotonina. (AMABIS; MARTHO, 2002)

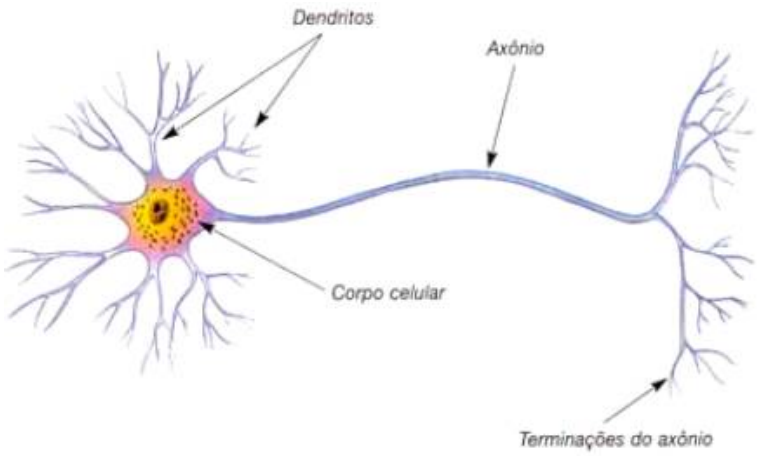


Figura 2.3 – Composição de um neurônio

Fonte: Packter (2011)

Os potenciais de ação da unidade motora são reações eletroquímicas que conduzem os sinais nervosos através dos neurônios. Essa condução é realizada na forma de pulsos regenerativos, ou seja, sem atenuação. Eles podem ser caracterizados como variações rápidas dos potenciais externo e interno da membrana da célula nervosa que se deslocam ao longo da fibra nervosa até atingirem a extremidade do axônio. Cada potencial de ação inicia com uma mudança abrupta de

um potencial negativo de repouso para um positivo, e em seguida acaba com um rápido retorno para o potencial negativo.

Durante o repouso a membrana celular é relativamente impermeável aos íos de Sódio (Na^+), ao passo que é bastante permeável aos íons de Potássio (K^+). Como durante essa etapa a concentração de K^+ em seu interior é alta, eles difundem-se para o exterior da membrana, causando a passagem de cargas positivas para fora da fibra, mas deixando muitos íons negativos em seu interior, gerando assim um potencial de membrana negativo da ordem de -90 mV em seu interior. (HALL, 2011)

Na despolarização, algum estímulo aumenta subitamente a permeabilidade da membrana aos íons Na^+, o que causa seu movimento rápido para o interior da célula, gerando cargas positivas que tornam a polaridade no interior positiva. Na terceira fase, a repolarização, a membrana torna-se novamente impermeável aos íons Na^+, apesar de ainda continuar permeável aos íons K^+. Isso torna a concentração de íons positivos no interior da célula nervosa elevada, de modo que os íons de potássio (K^+) difundem-se novamente para o exterior da membrana, tornando a região interior novamente negativa. Após essa fase, o neurônio torna-se apto a transmitir um novo impulso nervoso. A Figura 2.4 mostra o gráfico do potencial da membrana celular em cada estágio da condução nervosa. (ANDRADE, 2007)

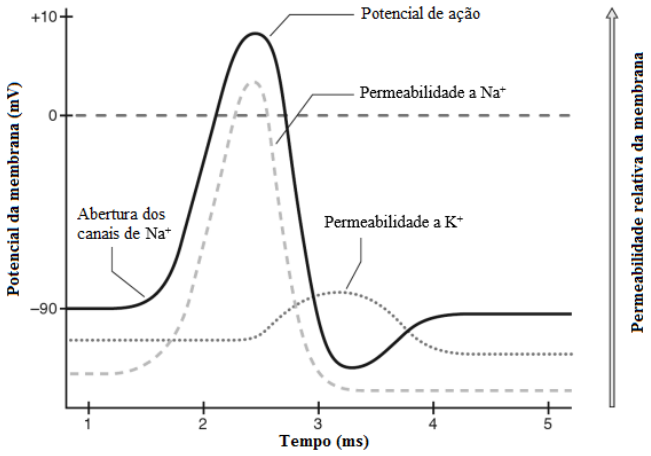


Figura 2.4 – Potencial da membrana nos estágios da condução nervosa

Fonte: Adaptado de Kamen e Gabriel (2010)

2.2.2 Excitação do músculo esquelético

As fibras nervosas originárias dos neurônios motores inervam as fibras musculares esqueléticas. Cada terminação nervosa forma uma junção, denominada junção neuromuscular, com a fibra muscular próxima de sua porção média. O potencial de ação iniciado na fibra muscular pelo sinal nervoso viaja em ambas as direções até as extremidades da fibra muscular. A

fibra nervosa forma um complexo de terminais nervosos ramificados que se invaginam na superfície extracelular da fibra muscular. Essa estrutura é chamada de placa motora. A Figura 2.5 mostra essa junção neuromuscular.

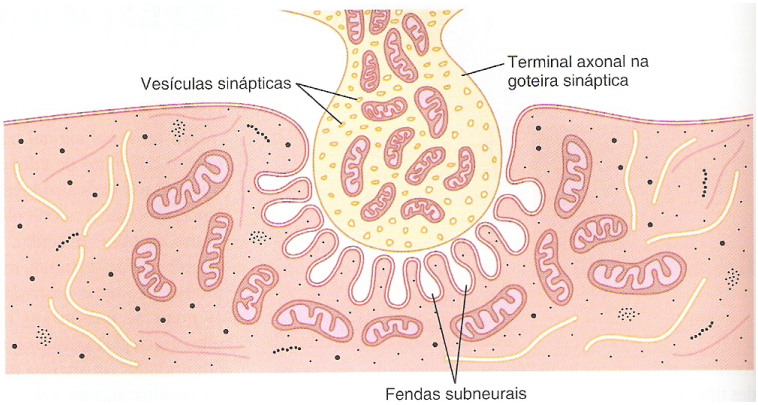


Figura 2.5 – Junção neuromuscular entre a fibra nervosa e a fibra muscular

Fonte: Hall (2011)

O espaço entre o terminal axonal e a membrana da fibra é chamado de fenda sináptica e tem de 20 a 30 nanômetros de largura. No fundo da goteira sináptica econtram-se pequenas dobras da membrana muscular, denominadas fendas subneurais, que aumentam a área de superfície na qual o transmissor sináptico pode atuar. No terminal axonal há muitas mitocôndrias que fornecem trifosfato de adenosina (ATP), que é fonte de energia para a síntese de um neurotransmissor excitatório, a acetilcolina.

Quando um impulso nervoso atinge a junção neuromuscular, a acetilcolina é liberada dos terminais axonais para o espaço sináptico. Isso ocorre devido a existência de canais de cálcio controlados por voltagem na membrana muscular, que se abrem na presença de um potencial de ação propagado no terminal e permitem a difusão de íons de cálcio do espaço sináptico para o interior do terminal nervoso. Esses íons exercem atração sobre as vesículas de acetilcolina, que então se fundem com a membrana neural e lançam a acetilcolina no espaço sináptico.

O efeito da liberação de acetilcolina é a abertura de um canal controlado por acetilcolina, de cerca de 0,65 nanômetro, pelo qual se movimentam íons positivos como sódio (Na^+), potássio (K^+) e cálcio (Ca^{++}). Já os íons negativos são repelidos pelas fortes cargas negativas presentes nesse canal. Isso permite que uma grande quantidade de íons sódio entre na fibra muscular, levando consigo carga positiva para o lado interno da membrana da fibra muscular, o que é chamado de potencial da placa motora. Esse potencial da placa motora inicia então um potencial de ação que se propaga ao longo da membrana muscular, o que gera a contração muscular. (HALL, 2011)

O registro dessa atividade elétrica gerada durante a contração muscular de uma unidade motora forma o SEMG. As unidades motoras são compostas por uma célula do corno anterior, um axônio, suas junções neuromusculares e as fibras musculares inervadas por ele. Um im-

pulso nervoso é conduzido pelo axônio para todas suas fibras musculares, resultando em sua despolarização praticamente simultânea. Essa desporalização produz uma atividade elétrica denominada potencial de ação da unidade motora (MUAP), que é capturado pelos eletrodos e exibida de forma gráfica como um SEMG. (O'SULLIVAN; SCHMITZ, 2010)

O MUAP consiste então no somatório das atividades elétricas das fibras musculares componentes da unidade motora. Sua amplitude é determinada pelos potenciais de ação das fibras musculares (MFAPs) individuais, somados temporalmente e espacialmente. Algumas fibras musculares de uma unidade motora podem ter longas terminações axiais, contribuindo com a componente mais defasada do MUAP, o que pode gerar MUAPs complexos com vários *spikes* (surtos de tensão). Por outro lado, se as terminações axiais são de mesmo comprimento e todas as fibras de uma unidade motora são ativadas simultaneamente, então o MUAP provavelmente terá uma curta duração e alta amplitude. A Figura 2.6 ilustra o MUAP como o somatório dos potenciais de ação das fibras individuais. (KAMEN; GABRIEL, 2010)

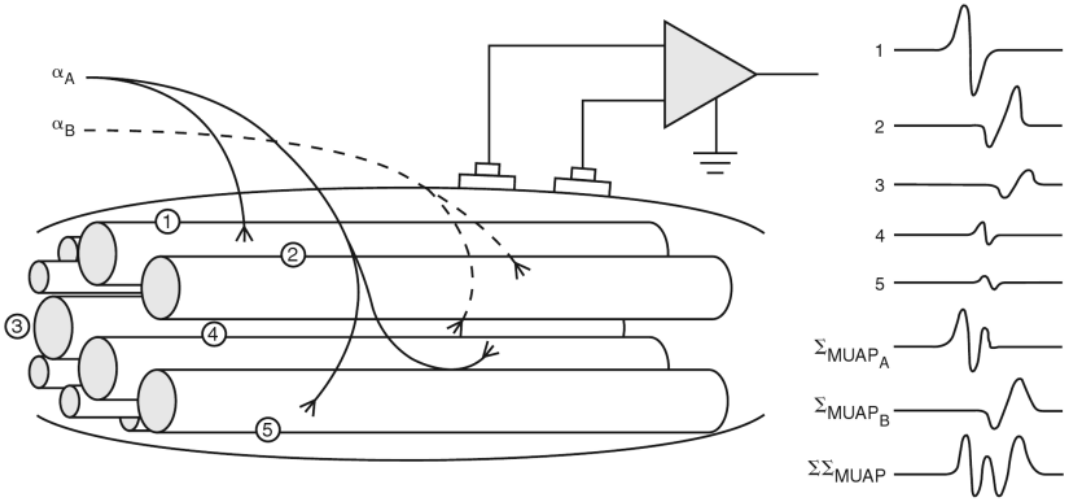


Figura 2.6 – O SEMG medido com eletrodos de superfície é o somatório dos potenciais de ação da unidade motora

Fonte: Kamen e Gabriel (2010)

2.3 Captura dos SEMGs

O sinal eletromiográfico possui uma amplitude muito baixa, na faixa de 0 a 10 mV (pico a pico) ou de 0 a 1,5 mV (rms). Sua frequência está entre 0 e 500 Hz, apesar de sua energia dominante estar na faixa de 50 a 150 Hz. (LUCA, 2002) A Figura 2.7 mostra um SEMG nos domínios do tempo e da frequência, demonstrando suas faixas de amplitude e de frequência típicas.

Para a captura do SEMG, duas classes de eletrodos podem ser utilizadas: invasivos e de superfície. Os eletrodos invasivos são constituídos de fios finos que são inseridos por uma

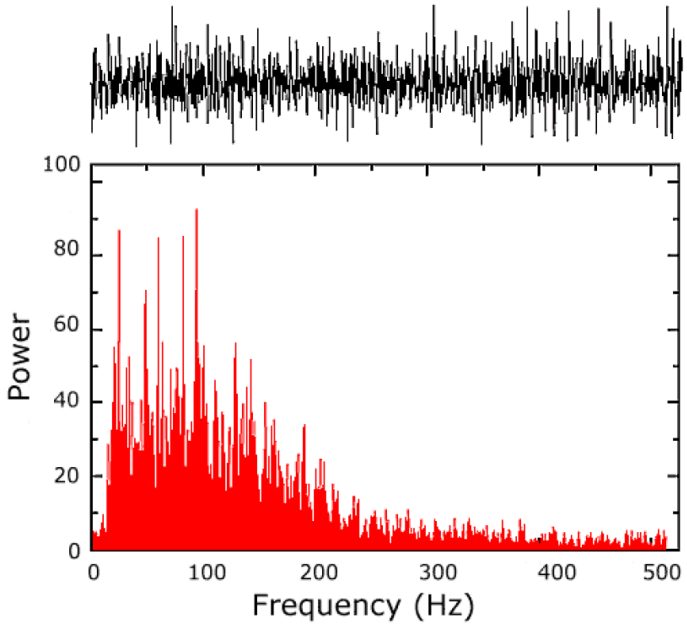


Figura 2.7 – Espectro de frequência de um SEMG típico

Fonte: Luca (2002)

agulha no ventre do músculo. Dessa forma pode-se coletar sinais de músculos bem específicos e reduz-se a possibilidade de *cross-talk*. Entretanto, tal eletrodo tem a desvantagem de causar desconforto e dor ao paciente e uma dificuldade na repetição dos experimentos.

Os eletrodos de superfície são dispostos na pele do paciente, não causando o desconforto das agulhas dos eletrodos invasivos. Eles são compostos por material de Ag/AgCl e são utilizados com o auxílio de um gel ou pasta condutora contendo íons de cloro, com o objetivo de diminuir a impedância entre o eletrodo e a pele. Entretanto, ainda é necessário realizar a tricotomia (raspagem de pelo) e a remoção da camada superficial da pele para uma captura mais precisa. A desvantagem da utilização do eletrodo de superfície é a grande possibilidade de *cross-talk*, já que também são detectados sinais provenientes de músculos vizinhos. (BARROS, 2005) A Figura 2.8 mostra a estrutura típica de um eletrodo de superfície.

2.4 Os músculos responsáveis pelos movimentos da mão

A mão humana é um órgão complexo capaz de realizar movimentos finos ou de força. Eles podem ser divididos em dois grupos: extrínsecos e intrínsecos. Os músculos extrínsicos são os flexores e extensores longos, que se originam no antebraço, externamente à mão e ao punho. Eles são responsáveis por movimentos de força da mão, que não necessitam de muita precisão. Já os músculos intrínsecos tem como função principal o controle dos movimentos dos dedos e de movimentos mais precisos. (KIDPORT, 2013)

Os músculos que controlam a mão conseguem mover os dedos em quatro direções: flexão,

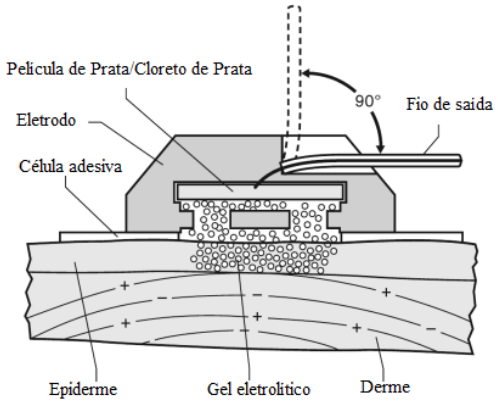


Figura 2.8 – Estrutura de um eletrodo de superfície

Fonte: Adaptado de Luca (2002)

extensão, abdução e adução. Estes músculos capazes de mover os dedos estão localizados na palma da mão e no antebraço. A Figura 2.9 mostra os músculos da mão.

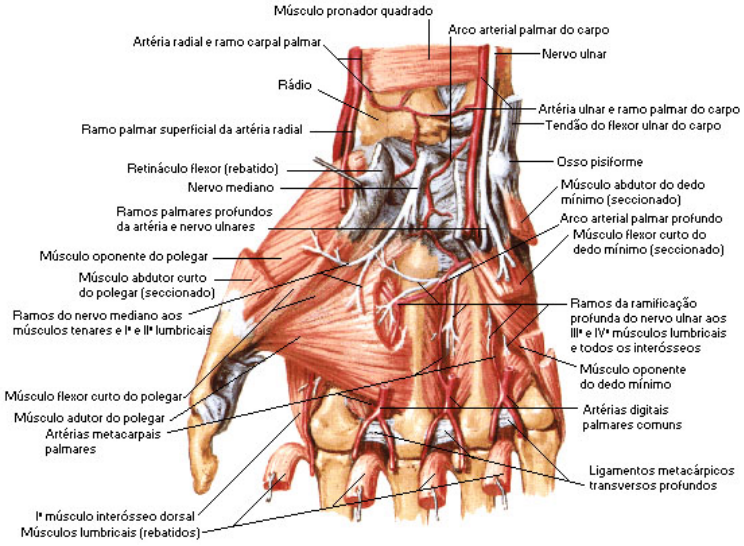


Figura 2.9 –

Fonte: Wecker (2013)

2.4.1 O punho e seus movimentos

O punho é constituído pelas articulações radiocárpica e intercápica, sendo que a maior parte de sua movimentação ocorre na articulação radiocárpica, uma articulação onde o rádio se articula com o escafóide. Essa articulação permite a movimentação no plano sagital: flexão, extensão e hiperextensão. Também permite movimentos no plano frontal: desvio radial e desvio ulnar.

A flexão do punho é o movimento da superfície palmar da mão na direção do antebraço anterior. A extensão é o retorno da mão à posição normal, ao passo que na hiperextensão a superfície da mão aproxima-se do antebraço posterior. A Figura 2.10 mostra os movimentos de flexão e hiperextensão do punho. (HALL, 2000)

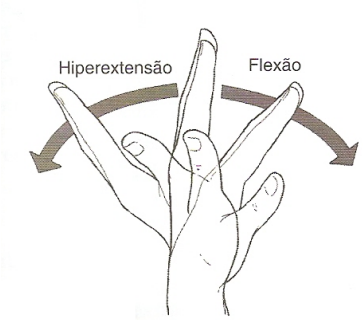


Figura 2.10 – Movimentos de flexão e hiperextensão do punho

Fonte: Hall (2000)

Os músculos responsáveis pela flexão do punho são o flexor radial do carpo e o flexor ulnar do carpo. Também contribui para esse movimento o músculo palmar longo. Quando os dedos estão completamente estendidos, os músculos flexor superficial dos dedos e o flexor profundo dos dedos também podem ajudar no movimento de flexão. A Figura 2.11 apresenta a localização no antebraço dos músculos responsáveis pelo movimento de flexão do punho.

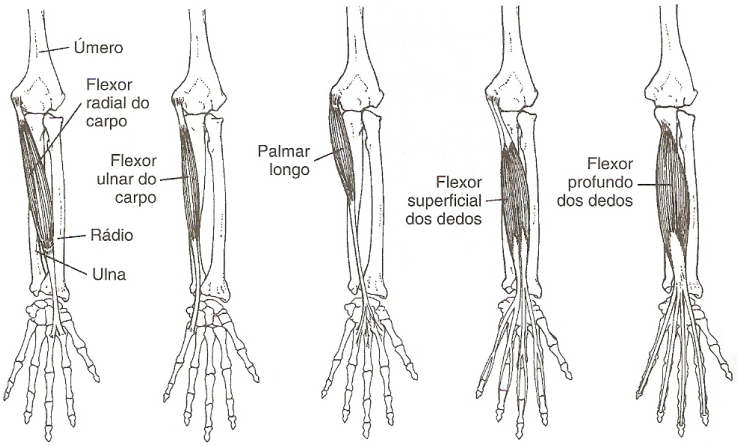


Figura 2.11 – Músculos responsáveis pela flexão do punho

Fonte: Hall (2000)

Os movimentos de extensão e hiperextensão do punho são resultado da contração do músculo extensor longo radial do carpo, do extensor curto radial do carpo e do extensor ulnar do carpo. Outros músculos também podem contribuir para esse movimento, especialmente quando os dedos estão flexionados. Dentre eles pode-se citar o extensor longo do polegar, o extensor

do indicador, o extensor do dedo mínimo e o extensor dos dedos. A Figura 2.12 apresenta a localização dos principais músculos responsáveis pelos movimentos de extensão e hiperextensão do punho.(HALL, 2000)

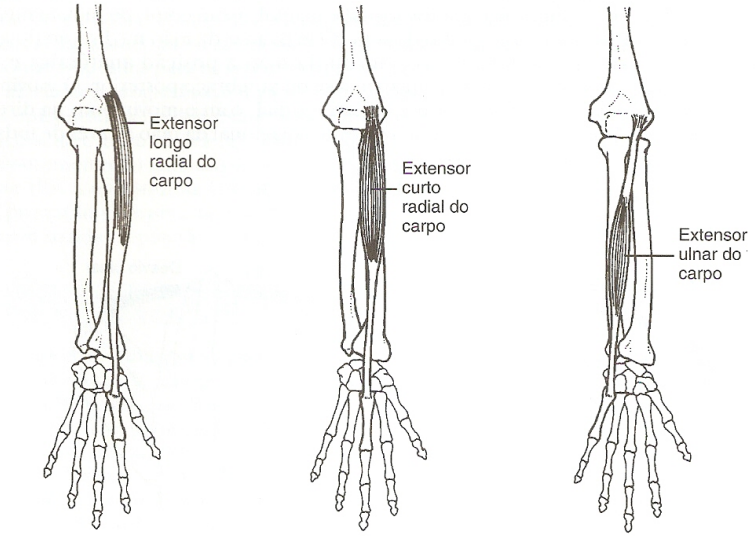


Figura 2.12 – Músculos responsáveis pela extensão e hiperextensão do punho

Fonte: Hall (2000)

Na plano sagital, dois movimentos do punho são possíveis, o desvio radial e o desvio ulnar. Eles são produzidos pela ação conjunta dos músculos flexores e extensores. Para produzir o desvio radial, o flexor radial do carpo e os extensores longo e curto radiais do carpo se contraem. Já para a produção do movimento de desvio ulnar, contraem-se o flexor ulnar do carpo e o extensor ulnar do carpo. A Figura 2.13 ilustra os movimentos de desvio radial e ulnar do punho.

Figura 2.13 – Movimentos de desvio radial e ulnar do punho

Fonte: Hall (2000)

3 Bases metodológicas para resolução do problema

3.1 Amplificadores operacionais

Um amplificador operacional (amp-op) é um amplificador diferencial de ganho muito alto com impedância de entrada muito alta e impedância de saída baixa. Ele é normalmente utilizado para se obter variações na tensão, para a construção de osciladores, filtros e circuitos de instrumentação. (BOYLESTAD, 1998)

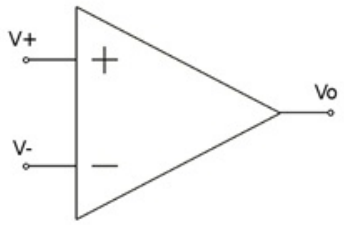


Figura 3.1 – Amplificador operacional básico

Além dessa configuração básica do amp-op, ele pode ser configurado de outras formas levando em consideração os sinais de entrada e o sinal de saída. A seguir descrevem-se algumas dessas configurações.

3.1.1 Entrada com um único terminal

A entrada com um único terminal ocorre quando se conecta o sinal de entrada a uma das entradas, enquanto a outra entrada se conecta à terra.

A Figura 3.2 ilustra a operação de entrada com um único terminal, em que o sinal de entrada é amplificado e a polaridade é mantida devido à ligação da entrada ao terminal positivo. Caso a entrada tivesse sido ligada ao terminal negativo, a poralidade teria sido invertida.

3.1.2 Entrada diferencial com dois terminais

Utilizando-se apenas um sinal de entrada aplicado a ambas as entradas do amp-op configura-se a operação diferencial com dois terminais. Supondo que uma entrada V_d seja aplicada nos

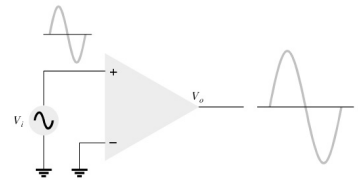


Figura 3.2 – Amp-op na operação com um único terminal

Fonte: Boylestad (1998, p. 428)

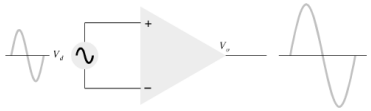


Figura 3.3 – Amp-op na operação com dois terminais

Fonte: Boylestad (1998, p. 429)

dois terminais de entrada, o sinal de saída resultante estará em fase com o sinal aplicado nas duas entradas. A Figura 3.3 mostra esta configuração e os sinais de entrada e saída.

3.1.3 Operação modo comum

Quando os mesmos sinais de entrada são aplicados aos dois terminais de entrada, a operação é chamada de operação modo comum. Nessa situação, as duas entradas são igualmente amplificadas e como produzem sinais com polaridades opostas, estes se cancelarão, resultando em uma saída de $0V$. A Figura 3.4 demonstra essa configuração do amp-op.

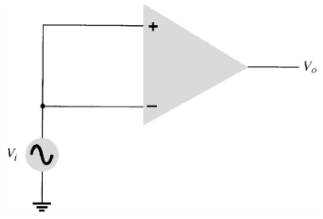


Figura 3.4 – Amp-op na operação modo comum

Fonte: Boylestad (1998, p. 430)

Uma das características mais importantes da conexão diferencial de um amp-op é a capacidade de o circuito amplificar consideravelmente sinais opostos nas duas entradas, enquanto amplifica suavemente sinais comuns a ambas as entradas. Dessa forma, um amp-op fornece uma componente da saída devida à amplificação da diferença dos sinais aplicados às entradas e uma outra componente devida aos sinais comuns às entradas. Pelo fato de a amplificação

dos sinais de entrada opostos ser muito maior que a dos sinais de entrada comuns, o circuito fornece uma rejeição de modo-comum que pode ser quantificada por um parâmetro denominado razão de rejeição de modo-comum (RRMC). Como o ruído geralmente é comum às duas entradas, essa conexão diferencial, juntamente com uma alta RRMC, tende a atenuar o ruído. (BOYLESTAD, 1998)

3.1.4 Tensão de saída

Quando duas entradas separadas V_{i_1} e V_{i_2} são aplicadas ao amp-op, o sinal diferença resultante pode ser dado pela diferença das duas entradas:

$$V_d = V_{i_1} - V_{i_2} \tag{3.1}$$

Já quando os sinais são iguais, o sinal comum às duas entradas é definido como a média aritmética entre eles:

$$V_c = \frac{1}{2}(V_{i_1} + V_{i_2}) \tag{3.2}$$

Em geral, a saída de um amp-op com dois sinais de entrada V_{i_1} e V_{i_2} pode ser dado pela Equação 3.3

$$V_o = A_d V_d + A_c + V_c \tag{3.3}$$

em que

$V_d =$ tensão diferença dada pela Equação 3.1

$V_c =$ tensão comum dada pela Equação 3.2

$A_d =$ ganho diferencial do amplificador

$A_c =$ ganho de modo-comum do amplificador

3.1.5 Razão de rejeição de modo-comum

Utilizando as definições de ganho diferencial e ganho de modo-comum, pode-se definir a RRMC pela razão entre esses ganhos, ou seja, pela Equação 3.4. (BOYLESTAD, 1998)

$$\mathrm{RRMC} = \frac{A_d}{A_c} \tag{3.4}$$

O valor da RRMC também pode ser escrito em escala logarítmica. A Equação 3.5 mostra essa definição, e em tal escala, a unidade de medida é o decibel (dB).

$$\mathrm{RRMC(log)} = 20 \log_{10} \frac{A_d}{A_c} \tag{3.5}$$

A situação mais favorável para a operação do circuito ocorre quando A_d é muito grande e A_c é muito pequeno. Isso implicaria que as componentes do sinal com polaridades opostas

apareceriam muito amplificadas na saída, ao passo que as componentes do sinal que estão em fase se cancelariam em grande parte. Em uma situação ideal, o valor da RRMC de um amp-op seria infinito. Entretanto, na prática isso não ocorre, mas quanto maior for a RRMC de um amp-op real, melhor sua operação. (BOYLESTAD, 1998)

3.1.6 Amplificador Inversor

O circuito do Amplificador Inversor utiliza uma configuração do amp-op com realimentação cuja finalidade é inverter a polaridade do sinal de entrada ao mesmo tempo que o amplifica. A Figura 3.5 mostra essa configuração. Nela, o sinal de entrada é conectado à entrada inversora através do resistor R_1, o resistor de realimentação R_f é conectado entre a saída e a entrada inversora, e a entrada não inversora é aterrada. (ALEXANDER; SADIKU, 2000)

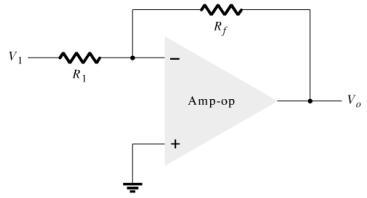


Figura 3.5 – Circuito do Amplificador Inversor

Fonte: Boylestad (2005)

O ganho dessa configuração pode ser obtido pela Equação 3.6 e a tensão de saída pela Equação 3.7. (ALEXANDER; SADIKU, 2000)

$$A = -\frac{R_f}{R_1} \tag{3.6}$$

$$v_0 = -\frac{R_f}{R_1} v_i \tag{3.7}$$

3.1.7 Amplificador Não-Inversor

O circuito do Amplificador Não-Inversor utiliza uma configuração do amp-op parecida com a configuração do Inversor, entretanto, o sinal de entrada é conectado diretamente à entrada não-inversora e o resistor R1 é conectado entre o terra e a entrada inversora. Sua finalidade é amplificar o sinal de entrada, mas sem inverter sua polaridade. A Figura 3.6 mostra essa configuração.

O ganho dessa configuração pode ser obtido pela Equação 3.8 e a tensão de saída pela Equação 3.9. (ALEXANDER; SADIKU, 2000)

$$A = \left(1 + \frac{R_f}{R_1}\right) \tag{3.8}$$

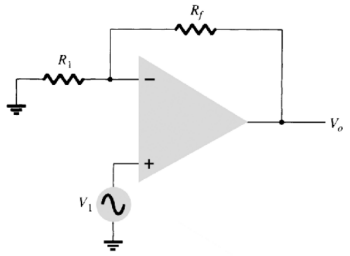


Figura 3.6 – Circuito do Amplificador Inversor

Fonte: Boylestad (2005)

$$v_0 = \left(1 + \frac{R_f}{R_1}\right) v_i \tag{3.9}$$

3.1.8 Amplificador para Instrumentação

O Amplificador para Instrumentação fornece uma saída baseada na diferença entre duas entradas amplificada, cujo fator de amplificação é controlado por um resistor variável. Ele utiliza 3 amp-ops, sendo 2 utilizados como *buffers*, com a finalidade de aumentar a impedância de entrada, e 1 amp-op na configuração diferencial que amplificará a diferença dos dois sinais de entrada. É desejável que a RRMC dos amp-ops utilizados no amplificador para instrumentação seja elevada, de forma que as interferências externas, comuns às duas entradas, sejam eliminadas pelo amplificador diferencial. A Figura 3.7 mostra o amplificador para instrumentação. (BOYLESTAD, 1998) (ALEXANDER; SADIKU, 2000)

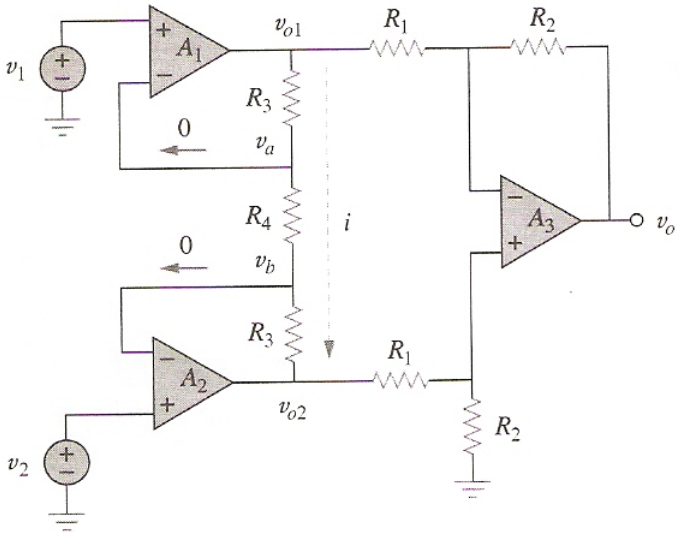


Figura 3.7 – Amplificador para Instrumentação

Fonte: Alexander e Sadiku (2000)

Como pode ser observado na Figura 3.7, para a construção desse circuito, são necessários resistores iguais, o que não é facilmente conseguido na prática, já que os resistores comumente encontrados tem uma tolerância de 5%. Por essa razão, resistores de precisão devem ser utilizados, ou então circuitos integrados que já implementem esse circuito internamente, como o AD620 ou o INA128.

Pode-se mostrar que a tensão de saída do amplificador de instrumentação mostrado na Figura 3.7 é dada pela Equação 3.10. O resistor R_4 é utilizado para regular o ganho do amplificador. (ALEXANDER; SADIKU, 2000)

$$V_0 = \frac{R_2}{R_1}\left(1 + \frac{2R_3}{R_4}\right)(v_2 - v_1) \tag{3.10}$$

3.2 Filtragem e Amplificação dos SEMGs

A baixa amplitude é um fator que torna inviável a transmissão direta desse sinal para um ambiente digital como um microcomputador pessoal. Com isso, dispositivos eletrônicos precisam ser utilizados para amplificar esse sinal, de forma a torná-lo apropriado para a transmissão a um computador, onde pode ser processado e/ou analisado por um especialista da área médica.

Além da baixa amplitude, os SEMGs estão sujeitos a ruídos oriundos do corpo do próprio paciente ou de fatores externos (como a rede de energia elétrica). Assim, eles precisam ser filtrados de forma a minimizar quaisquer ruídos que possam prejudicar sua futura análise e/ou processamento.

3.3 Filtros

Circuitos de filtros deixam passar à saída apenas os sinais de entrada que estão em uma faixa de frequência desejada (chamada de banda passante). A amplitude dos sinais fora dessa faixa de frequência (chamada banda rejeitada) é reduzida idealmente a zero, apesar de que na prática os filtros apenas atenuam essa faixa de frequência. Tipicamente em tais circuitos as entradas e saídas de corrente tem um pequeno valor, e assim a função de transferência da corrente não é um parâmetro importante. O principal parâmetro a se observar é a função de transferência da tensão no domínio da frequência $H(j\omega) = V_0/V_i$. Como $H(j\omega)$ é um número complexo, ele tem magnitude e fase, e os filtros geralmente produzem uma diferença de fase entre o sinal de entrada e o sinal de saída. (NAJMABADI, 2004)

Existem 4 tipos básicos de filtros: filtro passa-baixa, filtro passa-alta, filtro passa-banda e filtro rejeita-banda. Maiores detalhes podem ser obtidos em (NILSSON; RIEDEL, 1999). Neste trabalho serão apenas expostas as características básicas dos filtros passa-baixa, passa-alta e filtro de Notch (que é um tipo especial de filtro rejeita-banda).

3.3.1 Função de Transferência

A função de transferência de um circuito pode ser definida como a razão entre a transformada de Lapalace da saída do circuito e a transformada de Laplace do sinal de entrada. Matematicamente ela pode ser expressa pela Equação 3.11.

$$H(s) = \frac{Y(s)}{X(s)} \tag{3.11}$$

em que $Y(S)$ é a transformada de Laplace do sinal de saída e $X(s)$ é a transformada de Laplace do sinal de entrada. Ela depende da forma com a qual tenham sido escolhidos os sinais de entrada e de saída. (NILSSON; RIEDEL, 1999)

3.3.2 Frequência de corte

Como nos circuitos de filtros reais o módulo da função de transferência não sofre uma variação instantânea para 0, como ocorreria em um filtro ideal, necessita-se de uma definição mais realista para essa frequência. Normalmente a definição mais utilizada pelos engenheiros é a da Equação 3.12.

$$|H(j\omega_c)| = \frac{1}{\sqrt{2}} H_{max} \tag{3.12}$$

em que H_{max} é o valor máximo do módulo da função de transferência e ω_c é a frequência de corte.

3.3.3 Filtro passa-baixa

Um filtro passa-baixa é um circuito que permite a passagem fácil de baixas frequências e torna difícil a passagem de altas frequências. Uma das formas de construí-lo é utilizando-se um circuito RC em série, como indicado na Figura 3.8.

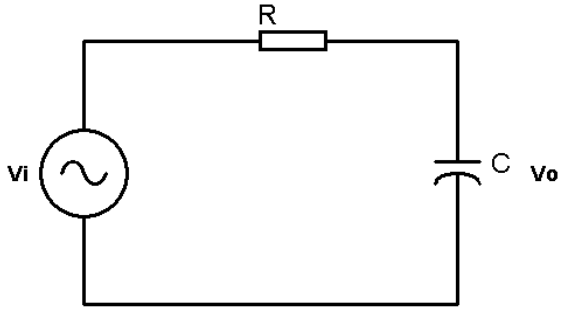


Figura 3.8 – Circuito RC em série para construção de um filtro passa-baixa

No circuito da Figura 3.8, define-se a tensão de entrada como a tensão da fonte V_i e a tensão de saída como a tensão no capacitor V_o. A partir desse circuito, pode-se construir o circuito equivalente no domínio da frequência indicado na Figura 3.9.

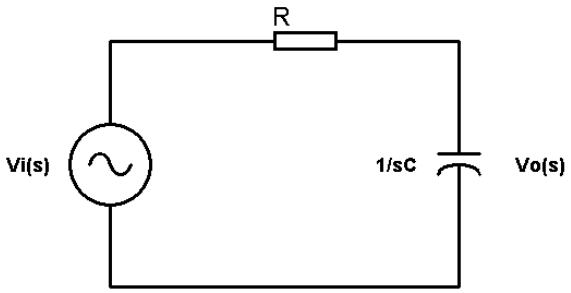


Figura 3.9 – Circuito RC equivalente no domínio da frequência para construção de um filtro passa-baixa

Aplicando-se o princípio da divisão de tensões ao circuito da Figura 3.9, obtém-se a função de transferência da Equação 3.13.

$$H(j\omega) = \frac{\frac{1}{RC}}{j\omega + \frac{1}{RC}} \tag{3.13}$$

Calculando-se o módulo da função de transferência, obtém-se a expressão da Equação 3.14.

$$|H(j\omega)| = \frac{\frac{1}{RC}}{\sqrt{\omega^2 + \left(\frac{1}{RC}\right)^2}} \tag{3.14}$$

Para o circuito RC da Figura 3.9, valor máximo de $|H(j\omega)|$ ocorrerá quando $j\omega = 0$, e seu valor será igual a 1. Dessa forma, usando a Equação 3.12 obtém-se a expressão da Equação 3.15.

$$|H(j\omega_c)| = \frac{1}{\sqrt{2}}(1) = \frac{\frac{1}{RC}}{\sqrt{\omega_c^2 + \left(\frac{1}{RC}\right)^2}} \tag{3.15}$$

Isolando-se ω_c na Equação 3.15, obtém-se a Equação 3.16 para a frequência de corte.

$$\omega_c = \frac{1}{RC} \tag{3.16}$$

em que ω_c é expresso em rad/s.

3.3.4 Filtro passa-alta

Um filtro passa-alta, ao contrário do filtro passa-baixa, permite a passagem com facilidade de altas frequências enquanto torna difícil a passagem de baixas frequências. Uma das formas de construí-lo é também pela utilização de um circuito RC. De fato, o mesmo circuito RC em

série pode funcionar como um filtro passa-baixa ou passa-alta, dependendo da escolha feita para a tensão de saída. No filtro passa-alta, a tensão de saída V_o é medida no resistor, como está indicado na Figura 3.10. (NILSSON; RIEDEL, 1999)

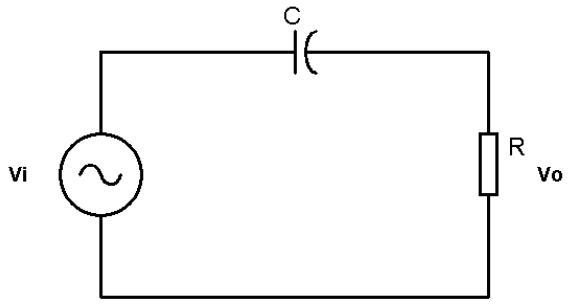


Figura 3.10 – Circuito RC para construção de um filtro passa-alta

A partir do circuito da Figura 3.10, pode-se construir o circuito equivalente no domínio da frequência apresentado na Figura 3.11.

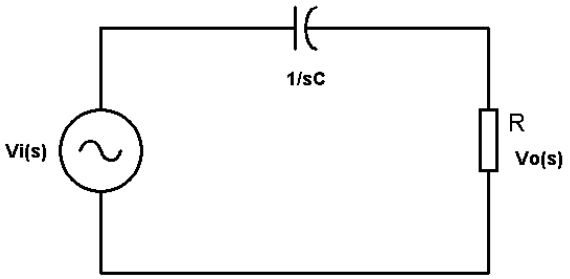


Figura 3.11 – Circuito RC equivalente no domínio da frequência para construção de um filtro passa-alta

Aplicando-se a regra dos divisores de tensão, obtém-se a função de transferência da Equação 3.17

$$H(j\omega) = \frac{j\omega}{j\omega + \frac{1}{RC}} \tag{3.17}$$

Calculando-se o módulo da função de transferência, obtém-se a expressão da Equação 3.18.

$$|H(j\omega)| = \frac{\omega}{\sqrt{\omega^2 + \left(\frac{1}{RC}\right)^2}} \tag{3.18}$$

Para o circuito RC da Figura 3.11, o valor máximo de $|H(j\omega)|$ ocorrerá quando $\omega \to \infty$, e seu valor será igual a 1. Dessa forma, usando a Equação 3.12 obtém-se a expressão da Equação

3.19.

$$|H(j\omega_c)| = \frac{1}{\sqrt{2}}(1) = \frac{\omega_c}{\sqrt{\omega_c^2 + \left(\frac{1}{RC}\right)^2}} \tag{3.19}$$

Isolando-se ω_c na Equação 3.19, obtém-se a Equação 3.20 para a frequência de corte.

$$\omega_c = \frac{1}{RC} \tag{3.20}$$

em que ω_c é expresso em rad/s.

3.3.5 Filtro de Notch

O Filtro de Notch é um filtro rejeita-faixa de banda estreita. Ele pode ser utilizado, por exemplo, para filtrar o ruído de 60 Hz da rede elétrica, sem prejudicar consideravelmente um sinal cuja faixa de frequência inclua a frequência de 60 Hz. Esse é o caso do SEMG, cuja faixa de frequência dominante encontra-se entre 0 e 500 Hz.

Umas das formas de se obter um filtro de Notch é pela utilização do circuito *Twin-T*, mostrado na Figura 4.9. Esse nome é devido à aparência da configuração, na qual há dois Ts simétricos com resistores trocados por capacitores e vice-versa.

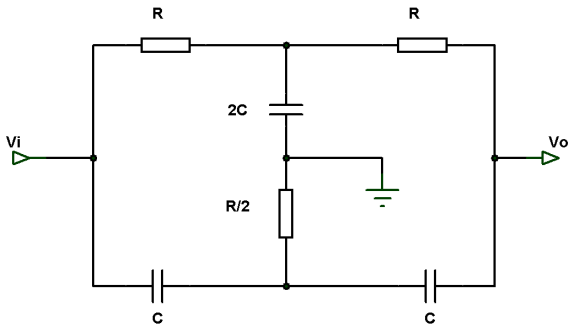


Figura 3.12 – Filtro de Notch utilizando a configuração *Twin-T*

Pode-se mostrar que a frequência de corte do filtro de Notch da Figura 4.9 é dada pela Equação 3.21. (NILSSON; RIEDEL, 1999)

$$\omega_c = \frac{1}{RC} \tag{3.21}$$

em que ω_c é expresso em rad/s.

Além disso, pode-se retirar as restrições dos relacionamentos entre resistores e capacitores, de forma que a frequência de corte do filtro de *Notch* da Figura 3.13 esteja entre as frequências ω_0 e ω_1, que são dadas pelas Equações 3.22 e 3.23. (OKAWA, 2013)

$$\omega_0 = \sqrt{\frac{\frac{1}{C_3} + \frac{1}{C2}}{C_1 R_1 R_2}} \tag{3.22}$$

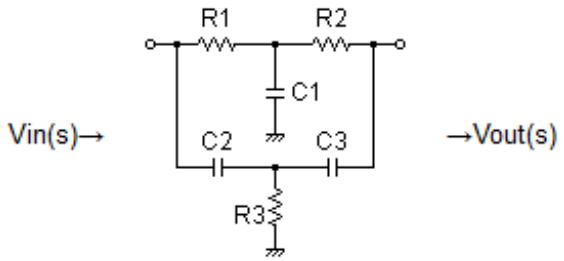


Figura 3.13 – Filtro de Notch genérico utilizando a configuração *Twin-T*

Fonte: Okawa (2013)

$$\omega_1 = \sqrt{\frac{1}{C_2 C_3 R_3 (R_1 + R_2)}} \tag{3.23}$$

3.4 Digitalização dos SEMGs

O processo de digitalização de um sinal analógico consiste em convertê-lo para um sinal digital, ou seja, mudar seu domínio de um conjunto de valores contínuos para um conjunto de valores discretos. Novamente, dispositivos eletrônicos são necessários para realizar esse processo e ele deve ser realizado de forma a minimizar a perda de informação.

A digitalização também é conhecida como amostragem, pois consiste na captura de amostras de um sinal em instantes discretos de tempo. A taxa de amostragem é a medida de quantas amostras são capturadas por uma fração de tempo, e geralmente é medida em Hertz. De forma geral, quanto maior a taxa de amostragem, melhor é a resolução horizontal de um sinal amostrado, e frequências mais altas poderão ser capturadas. (KAMEN; GABRIEL, 2010) A Figura 3.14 mostra 3 sinais analógicos amostrados com a mesma taxa de amostragem: um de baixa frequência, um de média frequência e outro de alta frequência.

Na Figura 3.14 nota-se que os sinais de baixa e média frequência tiveram sua forma de onda mais ou menos preservadas, enquanto o sinal de alta frequência teve sua forma distorcida, inclusive com a redução de sua amplitude. No sinal de alta frequência, ocorre também um fenômeno conhecido como *aliasing*, em que a frequência do sinal, que era de 15 Hz foi reduzida para 5 Hz. O *aliasing* ocorre devido à utilização de uma taxa de amostragem baixa demais para digitalizar um sinal de alta frequência. (KAMEN; GABRIEL, 2010)

3.4.1 Teorema da amostragem de Nyquist

O teorema da amostragem de Nyquist quantifica a relação entre a taxa de amostragem de um sinal e a frequência máxima que pode estar presente nele para que não ocorra o *aliasing*. Segundo ele, a taxa de amostragem deve ser o dobro da maior frequência presente no sinal,

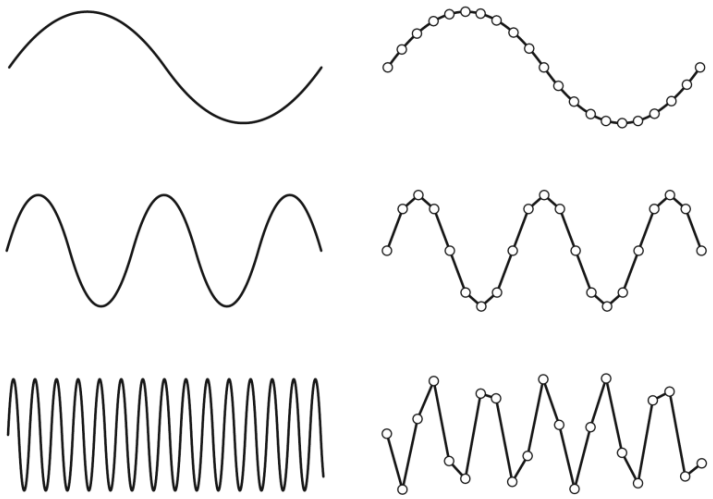


Figura 3.14 – Amostragem de 3 sinais: um de baixa frequência, um de média frequência e um de alta frequência

Fonte: Kamen e Gabriel (2010)

para que o sinal possa ser reconstruído sem *aliasing*. Essa taxa de amostragem é denominada frequência de Nyquist e é definida na Equação 3.24. (KAMEN; GABRIEL, 2010)

$$f_s = 2f_{max} \tag{3.24}$$

em que f_s é a frequência de Nyquist e f_{max} é a frequência máxima presente no sinal.

Dessa forma, é necessário remover frequências altas demais do SEMG (acima de 500 Hz), que possivelmente são oriundas de interferências, para não prejudicar a digitalização do SEMG. A remoção dessas frequências altas pode ser obtida pela utilização de filtros passa-baixa, com uma frequência de corte localizada em 500 Hz.

3.5 Análise de Fourier

A análise de Fourier é o estudo dos sinais e sistemas utilizando representações senoidais. Ela teve como seu contribuidor majoritário Joseph Fourier, que estudou e trouxe contribuições à teoria da representação das funções como superposições ponderadas de senóides. Outros matemáticos que colaboraram com o desenvolvimento desse campo foram Joseph Louis Lagrange, Jean le Rond d'Alembert e Carl Friedrich Gauss.

A análise de Fourier demonstra que qualquer função complexa pode ser representada por uma superposição ponderada de senóides complexas. Por meio dessa representação alternativa de uma função, extrai-se uma caracterização muito criteriosa dos sinais. (HAYKIN, 2001a)

Pode-se classificar as representações de Fourier para sinais em quatro classes, cada uma sendo adequada para cada tipo de sinal. A Tabela 3.1 mostra essa classificação.

Tabela 3.1 – Representações de Fourier

Tipo do Sinal	**Periódico**	**Não Periódico**
Contínuo	Série de Fourier (FS)	Transformada de Fourier (FT)
Discreto	Série de Fourier de Tempo Discreto (DTFS)	Transformada de Fourier de Tempo Discreto (DTFT)

3.5.1 Transformada de Fourier

A Transformada de Fourier é utilizada para representar um sinal não-periódico de tempo contínuo como uma superposição de senóides complexas. O fato de o sinal ser contínuo e não-periódico implica que os coeficientes da FT percorrem um intervalo infinito de frequências. A FT de um sinal contínuo não-periódico $x(t)$ é dada pela Equação 3.25. (HAYKIN, 2001a)

$$X(j\omega) = \int_{-\infty}^{\infty} x(t)e^{-j\omega t}\,\mathrm{d}t \tag{3.25}$$

Os coeficientes $X(j\omega)$ podem ser números complexos mesmo que o sinal original $x(t)$ seja apenas real. Dessa forma, para representar $X(j\omega)$ graficamente, existem 2 tipos de gráficos, o espectro de magnitude e o espectro de fase. O espectro de magnitude é obtido calculando-se o módulo de $X(j\omega)$ ($|X(j\omega)|$) e o espectro de fase é obtido calculando-se o argumento de $X(j\omega)$ ($argX(j\omega)$). A Figura 3.15 mostra o gráfico do sinal $x(t) = e^{-at}$ e sua FT por meio dos espectros de magnitude e de fase.

Uma vez representado no domínio da frequência pela Transformada de Fourier, um sinal pode ser levado ao domínio do tempo por meio da Transformada de Fourier Inversa (IFT). Pode-se mostrar que a IFT é dada pela Equação 3.26. (HAYKIN, 2001a)

$$x(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} X(j\omega)e^{j\omega t}\,\mathrm{d}\omega \tag{3.26}$$

3.5.2 Transformada de Fourier de Tempo Discreto

A representação por DTFT de um sinal é dada pela expressão a seguir (HAYKIN, 2001a):

$$x[n] = \frac{1}{2\pi}\int_{-\pi}^{\pi} X(e^{j\Omega})e^{j\Omega n}\,\mathrm{d}\Omega \tag{3.27}$$

em que

$$X(e^{j\Omega}) = \sum_{n=-\infty}^{\infty} x[n]e^{-j\Omega n} \tag{3.28}$$

A Equação 3.27 mostra que o sinal discreto não contínuo $x[n]$ pode ser representado como uma integral de $X(e^{j\Omega})$. A função complexa $X(e^{j\Omega})$ é denominada transformada de Fourier de Tempo Discreto de $x[n]$. A Equação 3.28 mostra como calcular essa transformada para um sinal que percorre o intervalo de tempo de $-\infty$ a ∞.

Além das representações de Fourier apresentadas na Tabela 3.1, existem variações dessas representações apropriadas para situações específicas. No caso em que se precisa mapear uma

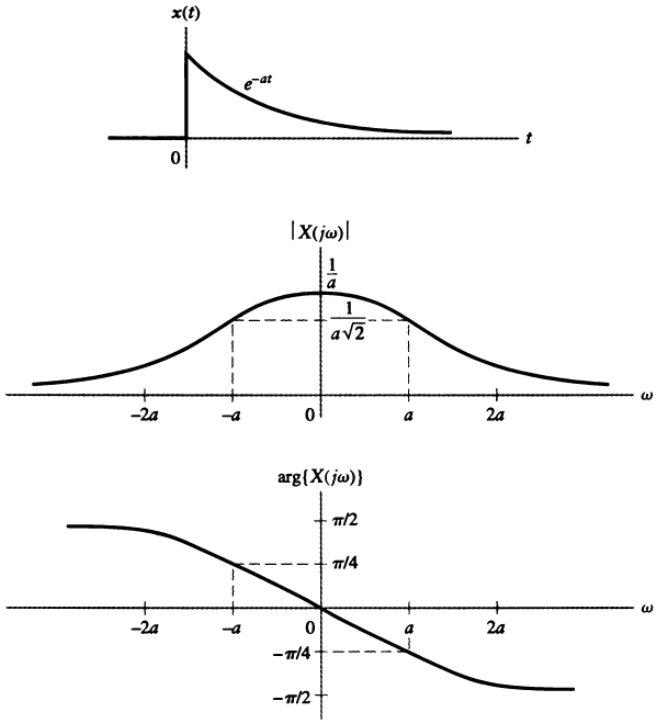


Figura 3.15 – Sinal exponencial e os espectros de magnitude e fase de sua FT

Fonte: Haykin (2001a)

sequência finita no domínio do tempo para uma sequência finita no domínio da frequência, utiliza-se uma categoria de transformações denominadas transformações de tempo-finito. Dentre estas transformações, destacam-se a Transformada Discreta de Fourier (DFT), a Transformada Discreta de Cosseno (DCT) e a Transformada de Haar. (MITRA, 2006) No escopo desse trabalho, foi utilizada a Transformada Discreta de Fourier.

A DFT, apesar da nomenclatura parecida, não é a mesma transformada que a DTFT apresentada na Tabela 3.1. Enquanto na DFT as sequências de entrada e saída são ambas finitas, na DTFT, essas sequências discretas variam em um intervalo infinito.

3.5.3 Transformada Discreta de Fourier

A Transformada Discreta de Fourier de um sinal no domínio do tempo $x[n]$ de comprimento N é definida pela Equação 3.29 (MITRA, 2006):

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j2\pi kn/N}, 0 \leq k \leq N-1 \tag{3.29}$$

Os coeficientes $X[k]$ da DFT geralmente são números complexos, mesmo quando o sinal $x[n]$ é real. Uma vez tendo transformado um sinal discreto do domínio do tempo para o domínio

da frequência, ou seja, os coeficientes $X[k]$ tenham sido encontrados, o sinal original pode ser obtido novamente utilizando-se a Transformada Discreta de Fourier Inversa (IDFT). Pode-se mostrar que a IDFT é dada pela Equação 3.30 (MITRA, 2006):

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] e^{-j2\pi kn/N}, 0 \leq k \leq N-1 \tag{3.30}$$

Similarmente, o sinal $x[n]$ obtido a partir da Equação 3.30 pode ser uma sequência de números complexos, mesmo que a sequência de coeficientes $X[k]$ seja real. Pela análise das Equações 3.29 e 3.30, constata-se que o cálculo da DFT requer aproximadamente N^2 multiplicações complexas e da IDFT requer $N(N-1)$ adições complexas. Entretanto, para reduzir a complexidade computacional do cálculo dessas transformadas, algoritmos especiais foram desenvolvidos. Eles conseguem diminuir essa complexidade computacional para aproximadamente $N(log_2 N)$. Essa classe de algoritmos recebe o nome de Transformada Rápida de Fourier (FFT) e se tornou amplamente utilizada em várias aplicações de ciência, engenharia e matemática. (MITRA, 2006)

3.5.4 Definições e operações sobre sinais discretos

Para simplificar a notação ao trabalhar com a DFT, normalmente utiliza-se a definição da Equação 3.31.

$$W_N = e^{\frac{-j2\pi}{N}} \tag{3.31}$$

Além disso, uma operação importante para o entendimento da FFT é a operação módulo, definida pela Equação 3.32. (MITRA, 2006)

$$k = \langle m \rangle_N \iff r = m + \ell N, \tag{3.32}$$

em que ℓ é um número inteiro tal que $m + \ell N$ esteja entre 0 e $N-1$.

A partir da definição do módulo, pode-se formar novas sequências a partir de uma dada sequência. Por exemplo, seja $x[k]$ uma sequência definida no intervalo de 0 a 3, cujos valores são definidos no gráfico da Figura 3.16. Agora suponha uma nova sequência $x_c[k]$ definida pela Equação 3.33 no intervalo de 0 a 7.

$$x_c[k] = x[\langle k \rangle_4], 0 \leq k \leq 7 \tag{3.33}$$

Pela aplicação da definição do módulo, chega-se à sequência indicada no gráfico da Figura 3.17. Nele pode-se perceber que as amostras do gráfico da Figura 3.16 foram repetidas à direita da sequência original. Essa operação é importante para a reconstrução de uma etapa de decimação no tempo da FFT.

Outras operações básicas sobre sinais discretos são: multiplicador, somador e avanço unitário. Supondo que $x[n]$ e $y[n]$ sejam dois sinais discretos e A um escalar, o operador multiplicador é

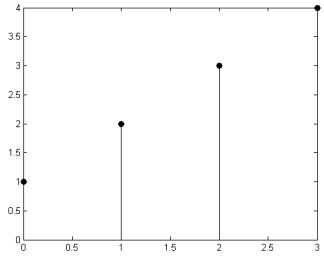


Figura 3.16 – Sequência $x[k]$ definida no intervalo de 0 a 3

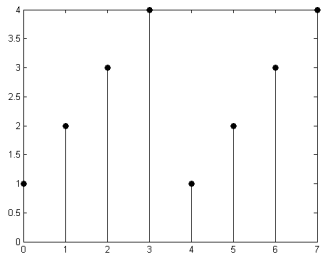


Figura 3.17 – Sequência $x_c[k]$ definida no intervalo de 0 a 7

definido pela Equação 3.34, o somador pela Equação 3.35 e o avanço unitário pela Equação 3.36. (MITRA, 2006)

$$w[n] = x[n] \cdot y[n] \tag{3.34}$$

$$w[n] = Ax[n] \tag{3.35}$$

$$w[n] = x[n+1] \tag{3.36}$$

Essas operações podem ser também representadas pela notação esquemática da Figura 3.18.

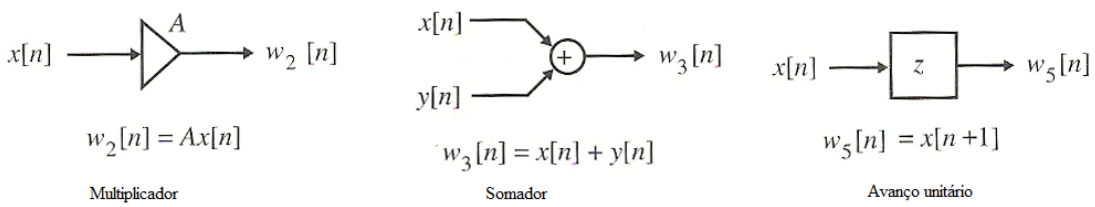


Figura 3.18 – Símbolos da representação esquemática das operações básicas do multiplicador, somador e avanço unitário

Fonte: Mitra (2006)

Outra definição importante é a da operação de *down-sampling*, também conhecida como decimação, por meio da qual é possível reduzir a quantidade de amostras de um sinal discreto. Seja $x[n]$ um sinal discreto qualquer, a operação de *down-sampling* por um fator inteiro $M > 1$ sobre este sinal é definida pela Equação 3.37. A Figura 3.19 mostra a representação esquemática da operação *down-sampling*.

$$y[n] = x[nM] \tag{3.37}$$

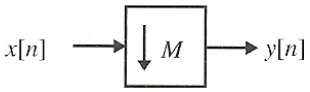


Figura 3.19 – Símbolo da representação esquemática da operação *down-sampling*

Fonte: Mitra (2006)

A Figura 3.20 mostra um sinal discreto original e o resultado da operação de *down-sampling* sobre ele por um fator M igual a 3. Percebe-se que há uma redução na quantidade de amostras por intervalo de tempo do sinal original.

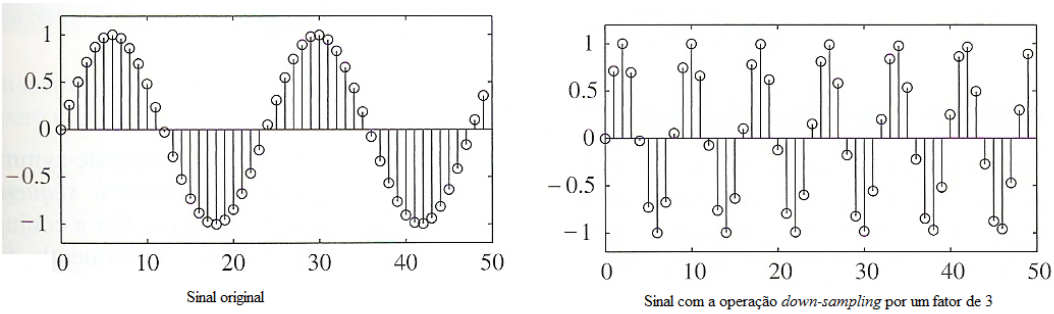


Figura 3.20 – Operação de *down-sampling* sobre um sinal discreto

Fonte: Mitra (2006)

3.5.5 Transformada Rápida de Fourier

A ideia por trás de todos os algoritmos rápidos para o cálculo da DFT, conhecidos como algoritmos de Transformada Rápida de Fourier, consiste em decomposições sucessivas do cálculo da DFT de N pontos em cálculos de DFTs de tamanhos menores que N, tirando proveito das propriedades de periodicidade e simetria do número complexo $e^{-j2\pi kn/N}$ (ou W_N^{kn}, utilizando a notação alternativa). Essas decomposições resultam na diminuição da complexidade computacional em comparação à aplicação direta da definição da DFT. Vários algoritmos de FFT

foram desenvolvidos, dentre os quais destacam-se o algoritmo de Cooley-Tukey e o algoritmo dos Fatores Primos. (MITRA, 2006)

3.5.5.1 Algoritmo de Cooley-Tukey

O algoritmo de Cooley-Tukey utiliza uma técnica chamada decimação no tempo, por meio da qual uma DFT de comprimento N é dividida em 2 DFTs de comprimento $\frac{N}{2}$, e cada uma dessas subdivisões continua o processo de forma recursiva se dividindo cada uma novamente em 2 DFTs, até que o cálculo da DFT se torne um simples cálculo de uma DFT de um sinal de duas amostras. De forma geral, para simplificar o algoritmo, considera-se que o sinal do qual se calculará a DFT tenha um comprimento de potência de dois. (MITRA, 2006)

Pode-se mostrar que DFT de um sinal discreto $x[n]$ pode ser decomposta de acordo com a Equação 3.38. (MITRA, 2006)

$$X[k] = X_0[\langle k \rangle_{\frac{N}{2}}] + W_N^k X_1[\langle k \rangle_{\frac{N}{2}}], 0 \leq k \leq N-1 \tag{3.38}$$

em que

$$X_0[k] = \sum_{r=0}^{\frac{N}{2}-1} x[2r] W_{\frac{N}{2}}^{rk}, 0 \leq k \leq \frac{N}{2} - 1 \tag{3.39}$$

$$X_1[k] = \sum_{r=0}^{\frac{N}{2}-1} x[2r+1] W_{\frac{N}{2}}^{rk}, 0 \leq k \leq \frac{N}{2} - 1 \tag{3.40}$$

As Equações 3.39 e 3.40 representam 2 DFTs de comprimento $\frac{N}{2}$, $X_0[k]$ formada a partir das amostras pares do sinal $x[n]$ e $X_1[k]$ das amostras ímpares de $x[n]$. Como as duas DFTs formadas tem um comprimento $\frac{N}{2}$, para a reconstrução da DFT do sinal original $X[k]$ pela Equação 3.38, faz-se necessário aplicar a operação módulo $\frac{N}{2}$ em cada DFT para que a soma dessas duas DFTs resulte em uma sequência de comprimento N.

A decomposição da Equação 3.38 também pode ser representada em notação de diagrama de blocos, como é mostrado na Figura 3.21.

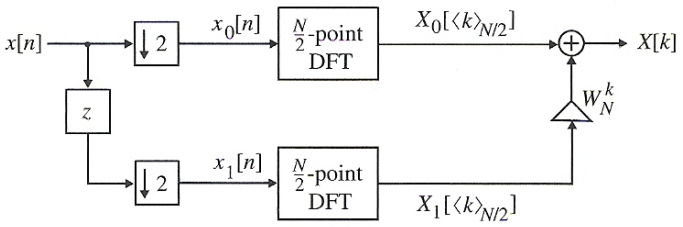


Figura 3.21 – Representação de diagrama de blocos da decomposição da DFT pela decimação no tempo

Fonte: Mitra (2006)

Também é possível representar a decomposição por decimação no tempo por um gráfico de fluxo. Para o caso de um sinal discreto com $N = 8$, o gráfico de fluxo é representado na Figura 3.22.

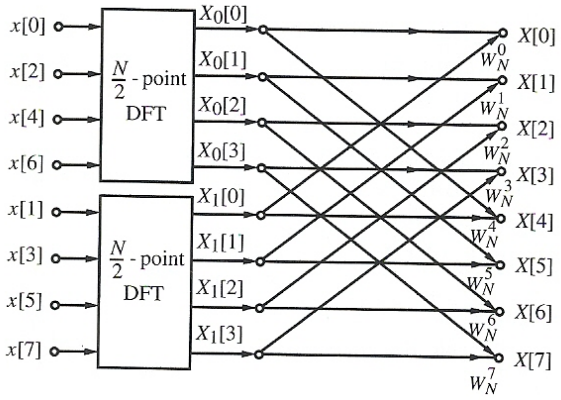


Figura 3.22 – Representação por gráfico de fluxo da decomposição da DFT de um sinal de tamanho 8

Fonte: Mitra (2006)

Depois da primeira etapa de decimação no tempo, o algoritmo prossegue de forma recursiva e cada uma das duas DFTs de tamanho $\frac{N}{2}$ da primeira etapa é dividida em 2 DFTs de tamanho $\frac{N}{4}$, utilizando o mesmo princípio da Equação 3.38. Nessa equação, $X_0[k]$ e $X_1[k]$ representam essas duas DFTs de tamanho $\frac{N}{2}$ e elas poderão ser decompostas de acordo com as Equações 3.41 e 3.42.

$$X_0[k] = X_{00}[\langle k \rangle_{\frac{N}{4}}] + W_{\frac{N}{2}}^{k} X_{01}[\langle k \rangle_{\frac{N}{4}}], 0 \leq k \leq \frac{N}{2} - 1 \tag{3.41}$$

$$X_1[k] = X_{10}[\langle k \rangle_{\frac{N}{4}}] + W_{\frac{N}{2}}^{k} X_{11}[\langle k \rangle_{\frac{N}{4}}], 0 \leq k \leq \frac{N}{2} - 1 \tag{3.42}$$

em que $X_{00}[k]$ é a DFT de comprimento $\frac{N}{4}$ da sequência $x_{00}[n]$, formada a partir das amostras pares da sequência x_0, $X_{01}[k]$ é a DFT de comprimento $\frac{N}{4}$ da sequência $x_{01}[n]$, formada a partir das amostras ímpares da sequência x_0, $X_{10}[k]$ é a DFT de comprimento $\frac{N}{4}$ da sequência $x_{10}[n]$, formada a partir das amostras pares da sequência x_1, e $X_{11}[k]$ é a DFT de comprimento $\frac{N}{4}$ da sequência $x_{11}[n]$, formada a partir das amostras ímpares da sequência x_1.

As decomposições das Equações 3.41 e 3.42 podem ser representadas pelo diagrama de blocos da Figura 3.23. Supondo que o tamanho do sinal discreto seja $N = 8$, tem-se que os 4 DFTs de comprimento $\frac{N}{4}$ serão 4 DFTs de comprimento 2. Uma DFT de comprimento 2 é facilmente calculada. Por exemplo, no caso da DFT $X_{00}[k]$, seu cálculo é dado pela Equação 3.43.

$$X_{00}[k] = \sum_{n=0}^{1} x_{00}[n] W_2^{nk} = x[0] + W_2^k x[4] \tag{3.43}$$

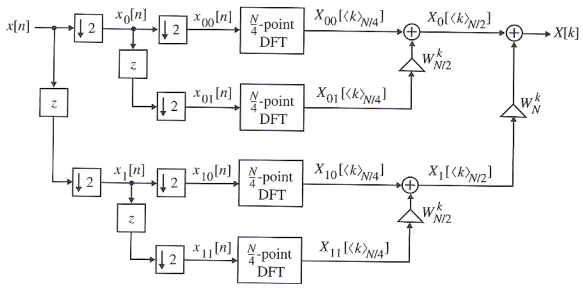


Figura 3.23 – Representação por diagrama de blocos das duas primeiras decomposições da FFT

Fonte: Mitra (2006)

A Figura 3.24 mostra a representação por gráfico de fluxo da etapa final de decomposição da DFT de um sinal de tamanho 8. Esse gráfico ilustra um componente computacional comum em várias etapas do algoritmo da FFT: o módulo computacional borboleta.

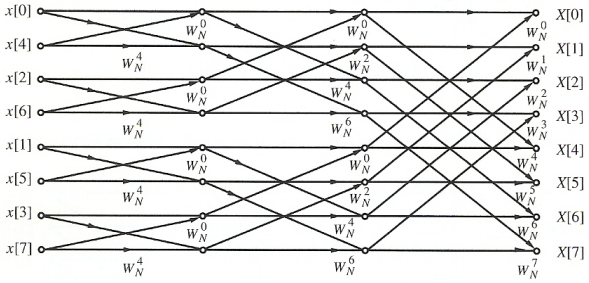


Figura 3.24 – Representação por gráfico de fluxo da 2ª etapa de decomposição da DFT de um sinal de tamanho 8

Fonte: Mitra (2006)

Esse componente de computação recebe esse nome devido à sua forma no gráfico de fluxo, que se assemelha à de uma borboleta. A Figura 3.25 mostra a representação em gráfico de fluxo do módulo borboleta.

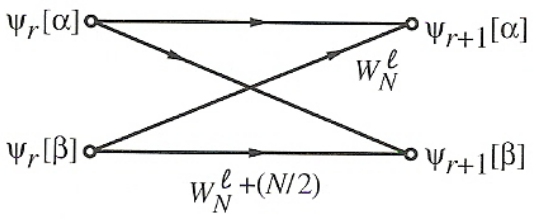


Figura 3.25 – Módulo computacional borboleta

Fonte: Mitra (2006)

As Equações 3.44 e 3.45 definem o componente de computação borboleta.

$$\Psi_{r+1}[\alpha] = \Psi_r[\alpha] + W_N^{\ell}\Psi_r[\beta] \tag{3.44}$$

$$\Psi_{r+1}[\beta] = \Psi_r[\alpha] + W_N^{\ell+\frac{N}{2}}\Psi_r[\beta] \tag{3.45}$$

em que $\Psi_r[\alpha]$ e $\Psi_r[\beta]$ representam duas saídas da r-ésima etapa da DFT e ℓ é um número que varia dependendo de qual etapa se encontra esse cálculo, como pode ser observado na Figura 3.24.

3.6 Extração de características

A extração de características (*feature extraction*) é uma técnica de redução de dimensionalidade normalmente aplicada em processamento de imagens e reconhecimento de padrão. Ela é utilizada quando a quantidade de entradas para um algoritmo é grande demais para ser processada, e então precisa ser reduzida para um conjunto menor de características. (GUYON; ELISSEEFF, 2006)

Essa técnica é útil para o processamento digital do SEMG, tendo em vista que o SEMG de um movimento em particular é formado por uma quantidade grande de amostras. Supondo uma taxa de amostragem de 8000 Hz e uma duração do SEMG de 1 s, por exemplo, teria-se 8000 amostras para processamento, uma quantidade grande demais, que tornaria o processamento pela rede neural lento.

Segundo o estudo de (PHINYOMARK; LIMSAKUL; PHUKPATTARANONT, 2009), uma série de características são úteis para o processamento do SEMG. Há algumas características que são calculadas no domínio do tempo e outras no domínio da frequência. Para o cálculo das características no domínio da frequência, tem-se a necessidade de aplicar a DFT ao SEMG.

3.6.1 Características no domínio do tempo

As características extraídas da representação do SEMG no domínio do tempo são chamadas de lineares e tem um cálculo mais simples do que as características no domínio da frequência. A seguir define-se algumas dessas características.

3.6.1.1 EMG Integrado (IEMG)

O EMG Integrado ($IEMG$) é calculado como o somatório dos módulos das amplitudes do SEMG no domínio do tempo. Geralmente é utilizado para detectar o início de um SEMG. Sua definição é dada pela Equação 3.46.

$$IEMG = \sum_{n=1}^{N} |x_n| \tag{3.46}$$

em que N é o tamanho do sinal e x_n representa uma amostra do SEMG em um instante de tempo n.

3.6.1.2 Valor Absoluto Médio (MAV)

O Valor Absoluto Médio (MAV) é calculado como a média do módulo das amostras do SEMG e é bastante utilizado para a detecção dos níveis de contração muscular. Ele é definido pela Equação 3.47.

$$MAV = \frac{1}{N}\sum_{n=1}^{N}|x_n| \tag{3.47}$$

em que N é o tamanho do sinal e x_n representa uma amostra do SEMG em um instante de tempo n.

3.6.2 Características no domínio da frequência

Estar características são calculadas a partir da representação do sinal no domínio da frequência, ou seja, a partir do espectro de frequência do sinal.

3.6.2.1 Frequência Média (MNF)

A Frequência Média (MNF) é calculada como a média ponderada das frequências utilizando-se as potências em cada frequência como peso. Ela é definida pela Equação 3.48.

$$MNF = \frac{\sum_{j=1}^{M} f_j P_j}{\sum_{j=1}^{M} P_j} \tag{3.48}$$

em que M é a quantidade de amostras no domínio da frequência, f_j é a j-ésima frequência do espectro de frequência e P_j é a j-ésima potência, referente à frequência f_j.

3.6.2.2 Frequência Mediana (MDF)

A Frequência Mediana (MDF) é a medida de frequência na qual o espectro de frequência é dividido em duas regiões com potências iguais. Sua definição é dada pela Equação 3.49.

$$\sum_{j=1}^{MDF} P_j = \sum_{j=MDF}^{M} P_j = \frac{1}{2}\sum_{j=1}^{M} P_j \tag{3.49}$$

em que M é a quantidade de amostras no domínio da frequência, P_j é a potência da amostra j e MDF indica o valor de Frequêcia Mediana.

3.7 Modelagem 3D de uma prótese virtual de mão

A modelagem 3D permite a criação de protótipos com custo reduzido, e por isso está sendo utilizada em diversas áreas. Ela também permite simular o comportamento de protótipos de

forma a verificar se tal comportamento corresponde ao esperado ou como meio de validação de alguma técnica aplicada a uma situação em particular.

Levando em conta esses aspectos, a modelagem 3D de uma prótese virtual de mão é um dos aspectos-chave deste trabalho, pois permite a validação das técnicas de processamento sem a necessidade da construção de uma prótese física, reduzindo consideravelmente o custo do projeto.

3.8 Computação gráfica 3D

A computação gráfica 3D é o ramo da computação que estuda a geração de imagens que utilizam a representação tridimensional de dados geométricos. Ela possui diversas aplicações na ciência e engenharia, das quais podemos destacar: desenho auxiliado por computador (DAC), visualização científica, modelagem computacional, etc.

A computação gráfica 3D se baseia em diversos algoritmos como gráficos vetoriais, raster e modelos de wireframe.

Uma figura pode ser descrita de diversas formas, sendo a forma mais básica um conjunto de intensidades de posições de pixels em um *display*. Por outro lado, também pode-se descrever uma figura como um conjunto de objetos complexos, como árvores, terrenos, móveis e paredes dispostos em localidades especificadas por coordenadas no espaço. As formas e cores dos objetos podem ser descritos internamente como *arrays* de pixels ou como conjuntos de estruturas geométricas mais elementares, como segmentos de linha ou polígonos. (HEARN; BAKER, 1996)

Na representação tridimensional de objetos, a forma mais utilizada é um cojunto de polígonos de superfície que englobam o interior do objeto. Isso simplifica e acelera a renderização dos objetos, já que todas as superfícies são descritas por equações lineares.

3.8.1 Tabelas de polígonos

Uma superfície poligonal pode ser especificada por um conjunto de coordenadas de vértices e parâmetros adicionais associados. As tabelas de polígonos são classificadas em duas categorias: tabelas geométricas e tabelas de atributos. As tabelas geométricas contêm informações sobre as coordenadas dos vértices e outros parâmetros para identificar a orientação espacial da superfície poligonal. Já as tabelas de atributos detalham informações como o grau de transparência do objeto, a refletividade de sua superfície e características de textura.

Uma forma conveniente de organizar o armazenamento dos dados geométricos é a criação de três tabelas: uma tabela de vértices, uma tabela de arestas, e uma tabela de polígonos. As coordenadas de cada vértice do objeto são armazenadas na tabela de vértices. A tabela de arestas contém ponteiros para os elementos da tabela de vértices e identificam cada aresta do polígono. Por fim, a tabela de polígonos contém ponteiros para a tabela de arestas e identifica as arestas de cada polígono.(HEARN; BAKER, 1996)

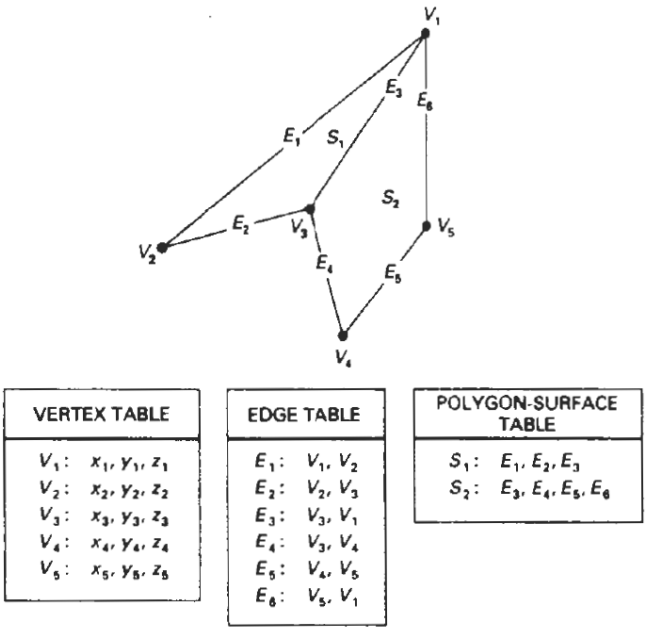


Figura 3.26 – Representação tabular de dados geométricos de duas superfícies poligonais adjacentes

Fonte: Hearn e Baker (1996, p. 307)

3.8.2 Blender

Blender é uma aplicação de gráficos 3D *open source* que pode ser utilizada para a criação de visualizações 3D, imagens e animações 3D. Ele foi concebido inicialmente em dezembro de 1993 e tornou-se um produto usável em agosto de 1994. Originalmente foi desenvolvido pela empresa Not a Number (NaN) e era um produto comercial, tendo sido transferido em março de 2002 para a organização sem fins lucrativos *Blender Foundation*, que o transformou em um produto *open source*. (BLENDER FOUNDATION, 2013)

O Blender provê uma interface de visualização 3D com uma quantidade enorme de ferramentas à disposição do usuário, desde ferramentas para a modelagem 3D, até ferramentas para a construção de animações completas. A Figura 3.27 mostra a interface de visualização 3D padrão do Blender.

3.8.2.1 Animação Esqueletal e *Keyframes*

A animação esqueletal consiste no processo de utilizar um conjunto de ossos interconectados para construir uma animação. Esses ossos são associados aos vértices de um modelo de *mesh*, que então são também deformados pela animação esqueletal. Esse tipo de animação é apropriada para a animação de personagens complexos, como seres humanos, insetos e até invertebrados. Para a animação de objetos como água ou estruturas rígidas simples, outras técnicas como animações de nó ou de cena são mais apropriadas. (SORIANO, 2009)

A animação esqueletal também é útil para a animação de membros do corpo humano, já

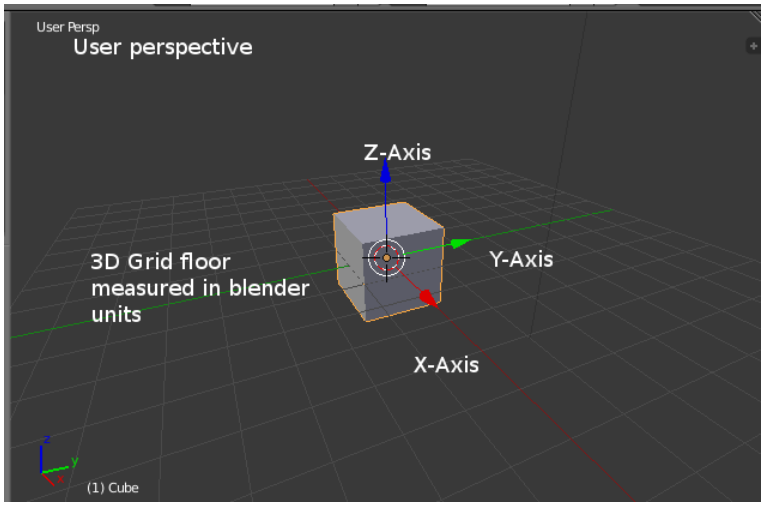


Figura 3.27 – Interface de visualização 3D padrão do Blender

Fonte: Blender Foundation (2013)

que ela mimetiza sua estrutura esquelética. Com seu uso, pode-se reproduzir os movimentos de um membro, como uma mão, com razoável fidelidade. O Blender introduz um objeto especial *armature*, que consiste em um conjunto de ossos, por meio do qual pode-se construir animações.

Um dos conceitos básicos das animações no Blender é o de *keyframe*. Um *keyframe* representa o estado de uma animação em um ponto do tempo. Como seria impraticável construir todos os estados de um objeto durante uma animação em todos os *frames*, os *keyframes* permitem a construção de estados essenciais da animação e o restante dos *frames* são construídos pelo próprio Blender utilizando algoritmos de interpolação.

A interpolação pode ser controlada por um gráfico chamado *F-Curve*. Nele pode ser escolhido o tipo de interpolação: constante, linear e de Bézier. Na interpolação constante, os *frames* são preenchidos com o valor do último *keyframe* anterior. Na interpolação linear, o Blender cria um segmento de reta entre dois *keyframes* para completar os *frames*. O terceiro tipo de interpolação, o que produz animações mais suaves e utilizado por padrão pelo Blender, é a interpolação de Bézier. A Figura 3.28 mostra o gráfico de *F-Curve* para cada tipo de interpolação disponível no Blender. (BLENDER FOUNDATION, 2013)

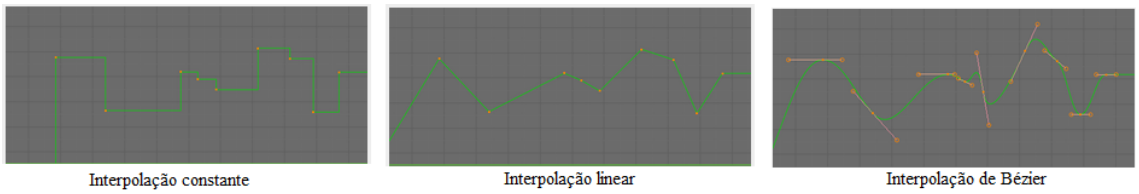


Figura 3.28 – Gráficos *F-Curve* para cada tipo de interpolação do Blender

Fonte: Blender Foundation (2013)

3.9 Redes Neurais Artificiais

Uma Rede Neural Artificial (RNA) é uma máquina projetada para modelar a maneira com que o cérebro realiza uma tarefa ou função de interesse. Ela pode ser implementada com o uso de componentes eletrônicos ou por meio de simulação em *software*. Para obterem um bom desempenho, essas redes utilizam interligações entre células computacionais denominadas neurônios.

As RNA são capazes de obter conhecimento a partir do seu ambiente por meio de um processo de aprendizagem, de forma similiar ao que ocorre com os cérebros das pessoas. Além disso, forças de conexão entre neurônios, chamadas de pesos sinápticos, são utilizadas para armazenar esse conhecimento adquirido. (HAYKIN, 2001b)

3.9.1 Modelos de um neurônio

O neurônio é a unidade básica de processamento de informação em uma RNA e pode-se identificar três elementos que compõem o modelo neuronal: sinapses caracterizadas por pesos próprios, um somador para realizar a combinação linear dos sinais de entrada ponderados pelos pesos sinápticos, e uma função de ativação para restringir a amplitude da saída de um neurônio. A Figura 3.29 apresenta o modelo de um neurônio.

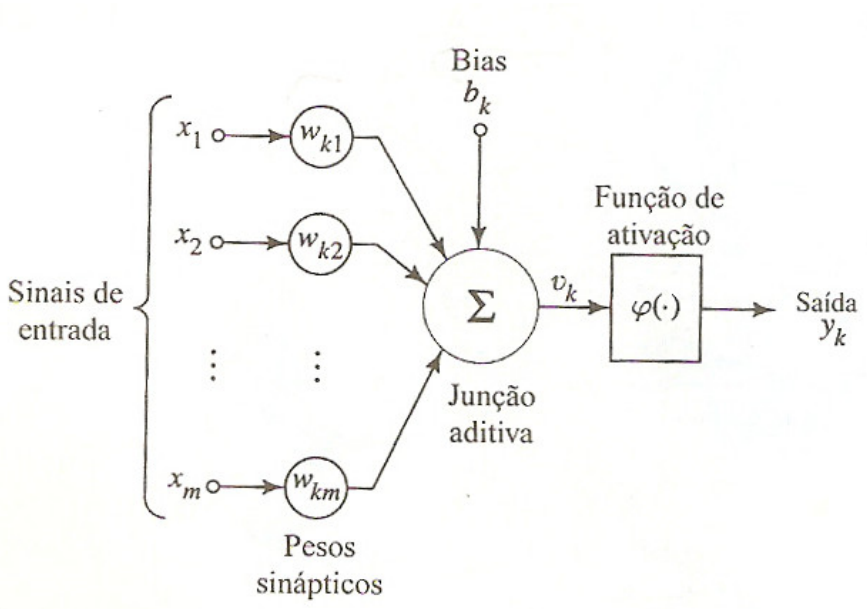


Figura 3.29 – Modelo de neurônio

Fonte: Haykin (2001b, p. 36)

3.9.2 Treinamento de uma RNA

Os neurônios individuais que compõem uma RNA estão interconectados através de suas sinapses, que permitem que eles sinalizem uns aos outros à medida que a informação é processada. Essas conexões não são iguais. De fato, cada conexão entre dois neurônios tem um peso sináptico. Caso dois neurônios não tenham uma conexão entre si, seu peso sináptico é 0. Esses pesos sinápticos são o que determina a saída de uma rede neural e dessa forma pode-se considerar que eles formam a memória de uma RNA.

O treinamento de uma RNA consiste no processo pelo qual esses pesos sinápticos são atribuídos. De uma forma geral, os algoritmos de treinamento iniciam atribuindo números aleatórios aos pesos sinápticos. Em seguida, a validade da rede neural é checada, e os pesos sinápticos são ajustados com base na validade dos resultados da RNA. Esse processo se repete até que o erro de validação da rede esteja dentro de limites aceitáveis. Há diversas formas de se treinar uma RNA. De modo geral, os algoritmos de treinamento se enquadram nas categorias de treinamento supervisionado e não supervisionado. (HEATON, 2008)

O treinamento supervisionado funciona através do fornecimento à rede neural de um conjunto de dados de amostra juntamente com as saídas esperadas de cada amostra. O treinamento ocorre por meio de uma sequência de iterações (conhecidas como *epochs*), até que a saída da rede neural corresponda à saída esperada, com uma taxa de erro relativamente pequena. Cada *epoch* consiste em uma passagem por todas as amostras de treinamento.

O treinamento não supervisionado também utiliza a ideia de *epochs*, entretanto, ao contrário do treinamento supervisionado, nenhuma informação sobre as saídas esperadas é fornecida. Geralmente ele é utilizado quando a rede neural está sendo usada para classificar as entradas em vários grupos. À medida que o treinamento progride pelas *epochs*, os grupos de classificação são descobertos pela rede neural.(HEATON, 2008)

Além desses métodos, ainda há métodos híbridos que combinam aspectos dos treinamentos supervisionado e não supervisionado. Pode-se citar, por exemplo, o treinamento por reforço, em que a rede neural recebe um conjunto de dados de amostra sem as saídas esperadas, de modo similar ao treinamento não supervisionado. Entretanto, para cada saída, a rede neural recebe a informação se a saída foi correta ou não.

3.9.3 Problemas que podem ser resolvidos por RNAs

Há vários tipos de problemas que podem ser resolvidos por uma RNA. Dentre eles, destacam-se 4 categorias: classificação, predição, reconhecimento de padrão e otimização. A classificação consiste em classificar amostras de entrada em grupos. Por exemplo, uma aplicação de *e-mail* pode ter a funcionalidade classificar um *e-mail* de entrada como *spam* ou não. A predição tenta resolver o seguinte problema: dada uma série temporal de dados de entrada, como prever valores futuros dessa série. Ela é comumente aplicada em problemas envolvendo a predição de movimentos no mercado financeiro.

O reconhecimento de padrão é um dos usos mais comuns das redes neurais. Ele é uma forma de classificação na qual a RNA é treinada para reconhecer padrões, mesmo quando estes padrões estejam distorcidos. Por exemplo, uma pessoa consegue, ao visualizar uma imagem de um cachorro, reconhecê-lo e ter certeza de que não é um gato que está na imagem. Agora considere que a mesma imagem seja apresentada a um programa de computador, o problema de classificá-la como uma imagem de um cachorro e não de um outro animal, mesmo que seja de um pastor alemão ou de um poodle, seria um problema de classificação que poderia ser resolvido por uma RNA.

Uma quarta categoria de problemas normalmente resolvidos pelas RNAs é o da otimização. Ela é aplicada sempre que uma solução ótima está sendo procurada. Entretanto, a rede neural nem sempre encontrará a solução ótima, mas procurará uma solução aceitável. Como exemplo, pode-se citar o roteamento de placas de circuito impresso ou o clássico problema do caixeiro viajante. (HEATON, 2008)

3.9.4 Redes Neurais *Feedforward Backpropagation*

Uma Rede Neural *Feedforward Backpropagation* (também conhecidas como percéptrons de múltiplas camadas) é um dos tipos de arquiteturas de Redes Neurais mais comuns, já que ela pode ser aplicada a vários tipos de problemas. O termo *feedforward* reflete a forma pela qual ela processa e reconhece padrões: seus neurônios são conectados apenas em uma direção. Ela é composta por camadas, uma de entrada, uma de saída e possivalmente camadas ocultas. Cada camada é conectada à próxima e não há conexões de volta de uma camada para outra.

Já o termo *backpropagation* (retropropagação) descreve a forma pela qual a rede neural é treinada. O algoritmo de retropropagação é um algoritmo de treinamento supervisionado, dessa forma a rede neural recebe um conjunto de amostras juntamente com as saídas esperadas, o algoritmo procede então processando essas amostras pela rede neural e calculando o erro em relação às saídas esperadas. Os pesos e *thresholds* são então modificados e as amostras são passadas novamente pela rede neural e o erro é calculado. Esse processo continua até que o erro tenha sido minimizado para um valor aceitável. (HEATON, 2008)

A Figura 3.30 mostra uma arquitetura típica de rede neural *Feedforward*. A camada mais acima é a camada de entrada, cuja quantidade de neurônios varia de acordo com o problema específico à qual ela esteja sendo aplicada. A camada central é a camada oculta e a camada mais abaixo é a camada de saída e suas quantidades de neurônios também dependem do problema específico.

3.9.5 Funções de Ativação

As funções de ativação tem o objetivo de mudar a escala de uma entrada específica de uma rede neural para intervalos determinados. Para o funcionamento do algoritmo de treinamento de *backpropagation*, a função de ativação escolhida deve ser diferenciável. As funções de ativação

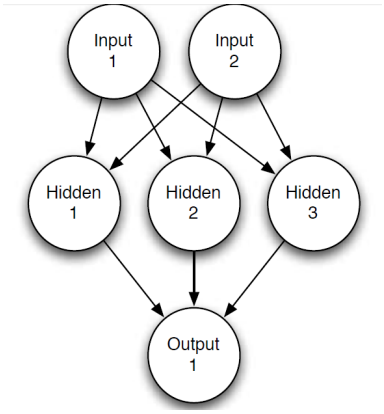


Figura 3.30 – Arquitetura de uma rede neural do tipo *feedforward*

Fonte: Heaton (2008)

mais comuns em redes neurais são: sigmoid, tangente hiperbólica e linear. A função sigmoid caracteriza-se por ter valores apenas positivos e sua definição pode ser dada pela Equação 3.50. A Figura 3.31 mostra o gráfico da função de ativação sigmoid.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{3.50}$$

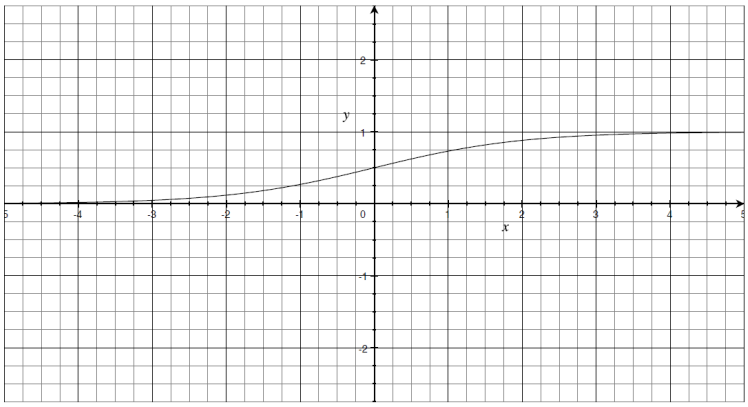


Figura 3.31 – Gráfico da função sigmoid

Fonte: Heaton (2008)

A função de ativação tangente hiperbólica, ao contrário da função sigmoid, pode apresentar valores negativos e por isso é usada em situações em que se deseja obter também saídas negativas. A tangente hiperbólica é definida pela Equação 3.51. A Figura 3.32 mostra o gráfico da função tangente hiperbólica.

$$f(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{3.51}$$

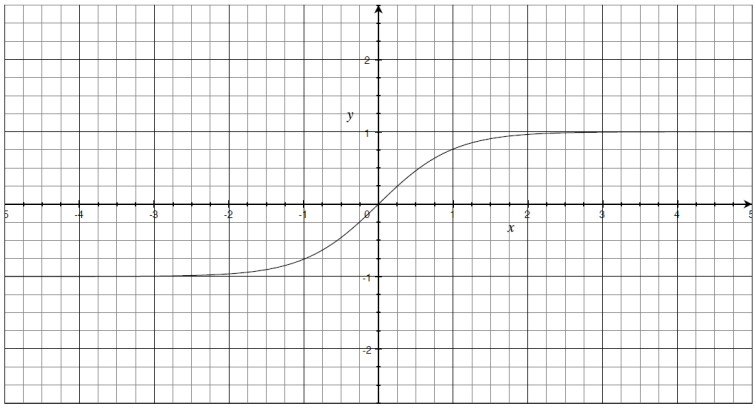


Figura 3.32 – Gráfico da função tangente hiperbólica

Fonte: Heaton (2008)

A função de ativação linear é a mais simples de todas e praticamente não é util, pois ela simplesmente não modifica um padrão de entrada. Sua definição é dada pela Equação 3.52, e um contexto no qual pode-se conceber seu uso é quando se deseja obter uma saída em um intervalo sem limites inferiores e superiores.

$$f(x) = x \tag{3.52}$$

Na rede neural desenvolvida neste trabalho, utilizou-se a função de ativação sigmoid em suas camadas. Essa rede neural utilizou a arquitetura *feedforward* e o algoritmo de aprendizado *backpropagation*. Essa escolha foi motivada pelos diversos trabalhos sobre a aplicação de redes neurais aos SEMGs, que relataram um bom desempenho dessa configuração da rede neural. (CAMARGO, 2008) (VASCONCELLOS, 2008)

4 Protótipo de prótese mioelétrica virtual

4.1 Estrutura Geral do Protótipo

O protótipo de prótese mioelétrica virtual construído consiste de componentes de *hardware* e *software*. Os componentes de *hardware* foram construídos para realizar a captura, filtragem e amplificação do SEMG. Já os componentes de *software* são responsáveis por realizar o processamento do SEMG, treinamento da rede neural e movimentação do modelo 3D de mão. A Figura 4.1 mostra a estrutura geral do protótipo construído. Os retângulos representam as etapas do projeto e as setas entre elas indicam sua ordem de precedência.

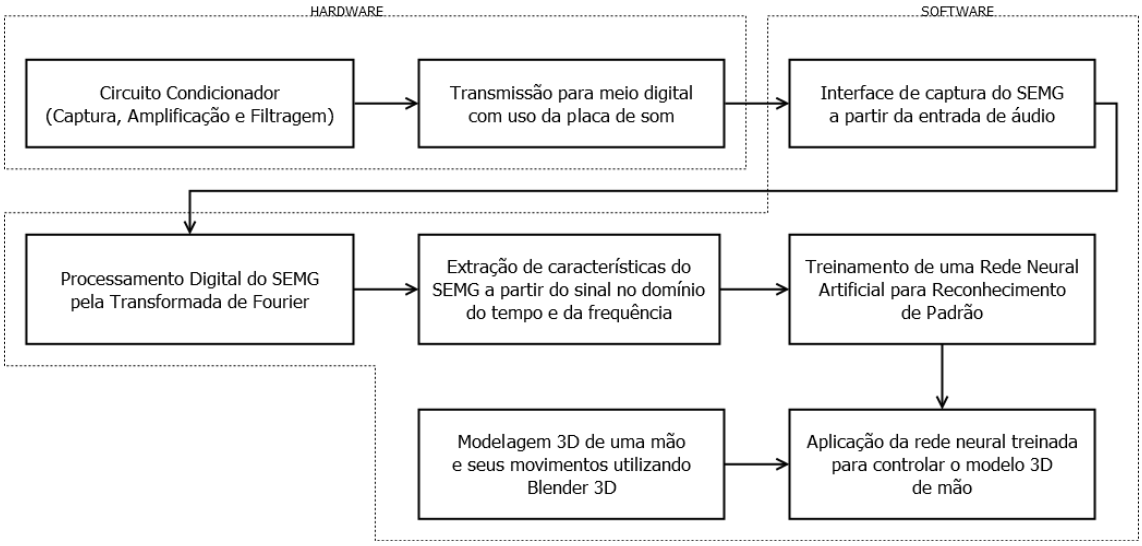


Figura 4.1 – Estrutura geral do protótipo desenvolvido

4.2 Componentes do Protótipo

Nas seções a seguir descreve-se em detalhes os componentes da estrutura do protótipo mostrados na Figura 4.1. Primeiramente serão descritos os componentes relacionados ao *hardware*, por meio de esquemáticos de circuitos e justificativas das abordagens utilizadas. Na sequência, serão tratados os componentes de *software*, utilizando diagramas descritivos da estrutura do *software* e listagens de código-fonte com trechos importantes do *software*.

4.2.1 Eletrodos de superfície

O SEMG é capturado por meio de eletrodos de superfície, dispositivos transdutores dos sinais biomédicos, detectados na pele devido à diferença de potencial causada pelos potenciais de ação da unidade motora. Os eletrodos utilizados nos testes foram da série Kendall Medi-Trace 200 Foam. A Figura 4.2 mostra uma foto do eletrodo utilizado. Os SEMGs captados pelos eletrodos de superfície tem uma amplitude muito baixa, da ordem de 10 mV. Além disso, eles também capturam ruídos de outros fenômenos biomédicos, como o batimento cardíaco, e também estão suscetíveis às interferências de aparelhos elétricos e da rede elétrica. Desse modo, torna-se necessário o desenvolvimento de circuitos de amplificação e filtragem do SEMG.

A configuração utilizada dos eletrodos foi a bipolar, na qual dois eletrodos, E_1 e E_2, são colocados sobre o músculo que está sendo analisado, próximo um do outro, e um terceiro eletrodo E_g é colocado sobre uma região neutra, normalmente uma proeminência óssea, desempenhando a função de eletrodo terra. Essa configuração é propícia para a diminuição da interferência não desejada com o uso de um amplificador diferencial, já que as componentes do ruído provavelmente serão comuns nos dois eletrodos E_1 e E_2, por estarem próximos, e serão canceladas devido a operação de rejeição de modo comum do amp-op diferencial.

Figura 4.2 – Eletrodos da série Kendall Medi-Trace 200 Foam

Fonte: Kendall (2013)

4.2.2 Circuito de captura, filtragem e amplificação do SEMG

4.2.2.1 Circuito utilizando o TLC274 e amplificador de áudio

O primeiro circuito de captura foi desenvolvido pelo autor em colaboração com os integrantes do PIC do UniCEUB e do Grupo de Engenharia de Reabilitação (GER-UniCEUB), durante pesquisa realizada em 2012 sobre a utilização dos SEMGs para o controle de um *game*. (TEIXERA et al., 2012) A Figura 4.3 mostra as partes componentes do *hardware* desenvolvido.

O circuito condicionador de captura e amplificação utiliza o amp-op TLC274CN, que é um CI caracterizado pela alta impedância de entrada, baixo ruído e alta RRMC. Sua RRMC varia de 65 a 85 dB, na temperatura de 25° C e operando com uma alimentação de 10 V. O TLC274CN foi montado na configuração de amplificador de instrumentação, com 3 amp-

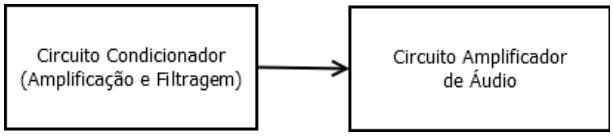


Figura 4.3 – Componentes do *hardware* desenvolvidos

ops interligados, dois 2 amp-ops de *buffer* e um terceiro amp-op na configuração diferencial. (TEXAS INSTRUMENTS, 1998) A Figura 4.4 mostra o esquemático do circuito desenvolvido.

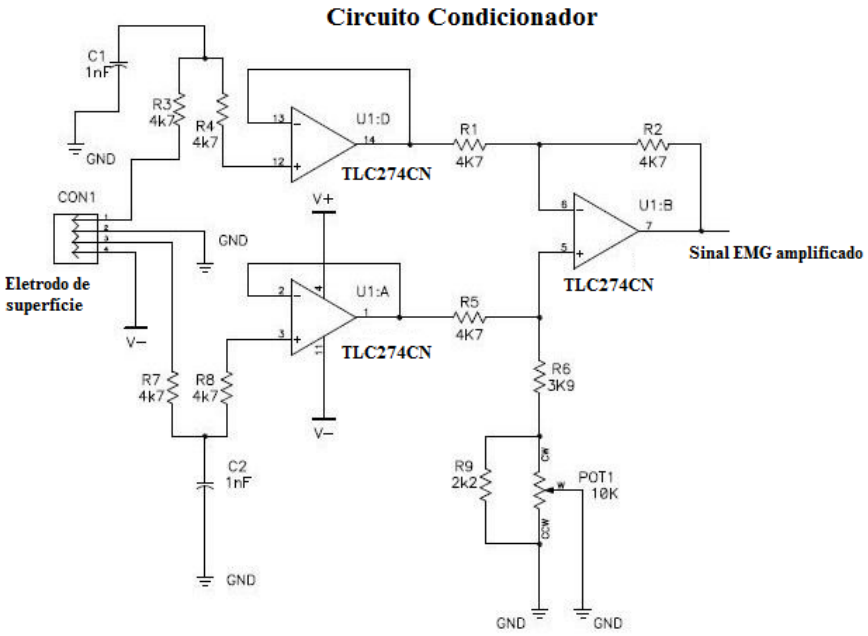


Figura 4.4 – Esquemático do circuito de captura e amplificação do SEMG

Fonte: Teixera et al. (2012)

O circuito tem capacitores e resistores na entrada dos amp-ops para a construção de um filtro ativo RC, que tem o papel de eliminar ruídos oriundos de outras atividades biomédicas, como os sinais cardíacos e outras interferências eletromagnéticas, deixando passar apenas a faixa do SEMG de 10 Hz a 500 Hz. A frequência de corte pode ser obtida a partir da Equação 4.1.

$$f = \frac{1}{2\pi RC} \tag{4.1}$$

4.2.2.2 Circuito utilizando o AD620

O circuito usando o TLC274CN para configurar um amplificador para instrumentação utiliza resistores externos convencionais com $\pm 5\%$ de tolerância, e por essa razão pode ter imprecisões devido à dificuldade de se encontrar resistores com valores reais iguais, que são necessários para construir a configuração amplificador para instrumentação. Por essa razão, no

intuito de melhorar a precisão da captura do SEMG, foi desenvolvido um circuito utilizando o AD620, que é um amplificador para instrumentação de alta precisão, com RRMC de 100 dB, tensão de alimentação de $\pm 2,3V$ a $\pm 18V$, e requer apenas um resistor externo para ajustar um ganho de 1 até 10.000. (ANALOG DEVICES, 2011) A Figura 4.5 mostra o encapsulamento DIP (Dual In-Line Package) de 8 pinos do AD620 e seu esquemático simplificado.

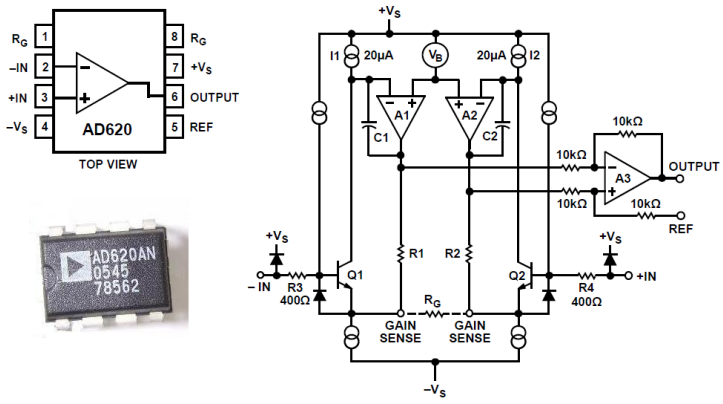


Figura 4.5 – Encapsulamento DIP do AD620 e seu esquemático simplificado

Fonte: ANALOG DEVICES (2011)

O ganho G do AD620 é regulado por um resistor R_G ligado aos pinos 1 e 8, cujo valor pode ser calculado pela Equação 4.2 (ANALOG DEVICES, 2011).

$$G = \frac{49,4k\Omega}{R_G} + 1 \tag{4.2}$$

A Figura 4.6 mostra o trecho do circuito construído que utiliza o AD620 para realizar a amplificação diferencial dos eletrodos bipolares. Nos terminais 2 e 3 do AD620 são conectados os cabos ligados aos eletrodos, o terminal 5 é conectado ao terra do circuito e ao eletrodo terra. Entre os terminais 1 e 8 foi ligado um *trimpot* de $10K\Omega$ em série com um resistor de 47Ω, para regular o ganho do AD620 de um valor 1052,06 ($R_G = 47\Omega$) até um valor 5,92 ($R_G = 10.047\Omega$).

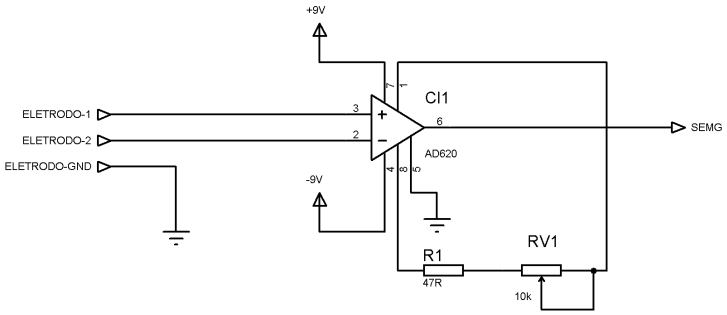


Figura 4.6 – Trecho do circuito que realiza a amplificação diferencial do SEMG

O terminal 6 é a saída do AD620, pela qual sairá o sinal diferencial das entradas, que terá grande parte das interferências externas canceladas devido a sua alta RRMC. Esse circuito extrai o SEMG sem nenhuma filtragem. Como o SEMG está sujeito a diversas interferências, torna-se necessário o desenvolvimento de filtros.

O primeiro filtro construído foi o filtro passa-alta da Figura 4.7. Ele é composto pelo capacitor de 100 nF e pelo resistor de 82 K Ω. Após o circuito de filtragem, há um amp-op na configuração de amplificador não-inversor com ganho 2, para fornecer um ganho à saída do filtro e criar um *buffer* com as outras etapas de filtragem.

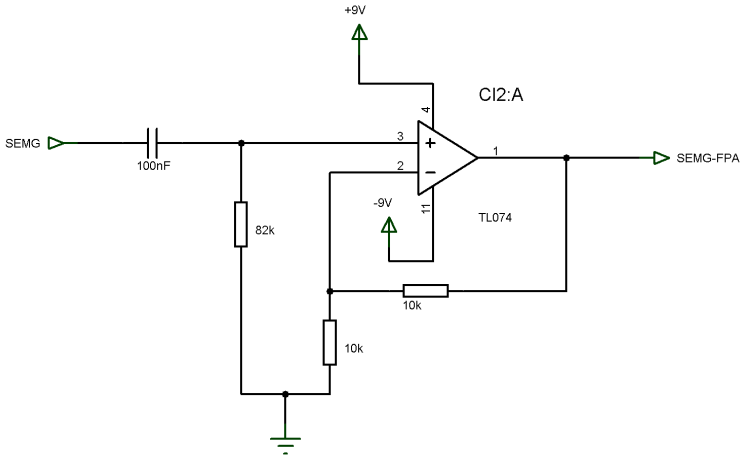


Figura 4.7 – Filtro passa-alta

Fonte: Adaptado de Teixera et al. (2012)

A frequência de corte do filtro passa-alta pode ser calculado utilizando-se a Equação 3.20. Esse cálculo é feito com os valores específicos de capacitor e resistor do filtro da Figura 4.7 na Equação 4.3.

$$\omega_c = \frac{1}{RC} = \frac{1}{82K \times 100n} = 121,95\,\mathrm{rad/s} = 19,4\,\mathrm{Hz} \tag{4.3}$$

Dessa forma, apenas frequências superiores a 19,5 Hz passarão sem atenuação neste filtro, sendo que as frequências inferiores a 19,5 Hz serão atenuadas. A saída do circuito da Figura 4.7 é aplicada a um filtro passa-baixa, a fim de limitar a frequência máxima do sinal e evitar o *aliasing* durante a digitalização. O filtro passa-baixa construído é mostrado na Figura 4.8.

O valor do resistor é de 3,3 K Ω e o do capacitor é de 100 nF. Com isso, a frequência de corte desse filtro pode ser calculado por meio da Equação 3.16. Esse cálculo é mostrado na Equação 4.4.

$$\omega_c = \frac{1}{RC} = \frac{1}{3,3K \times 100n} = 3030,30\,\mathrm{rad/s} = 482,29\,\mathrm{Hz} \tag{4.4}$$

Dessa forma, apenas frequências inferiores a 482,29 Hz passarão sem atenuação por este filtro, sendo que as frequências superiores a 482,29 Hz serão atenuadas. A saída desse filtro

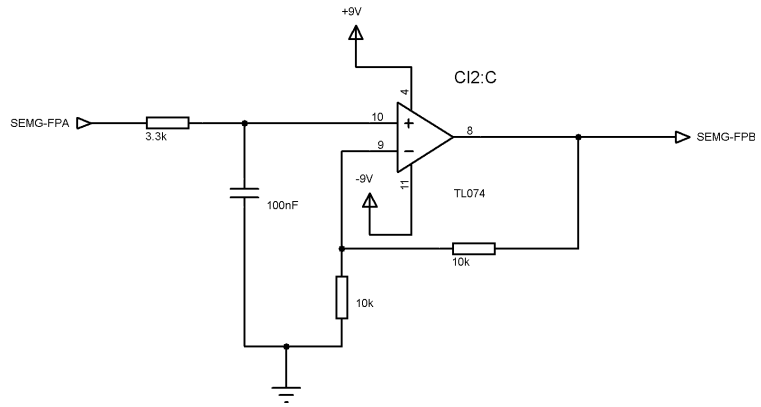


Figura 4.8 – Filtro passa-baixa

Fonte: Adaptado de Teixera et al. (2012)

ainda é aplicada a um amp-op na configuração não-inversor e com um ganho 2 para amplificar o sinal filtrado e servir de *buffer* para a próxima etapa de filtragem.

A saída do filtro passa-baixa da Figura 4.8 foi aplicado à entrada de um circuito de filtro de *Notch*, com o objetivo de filtrar a interferência da rede elétrica de frequência 60 Hz. Tal circuito é mostrado na Figura 4.9.

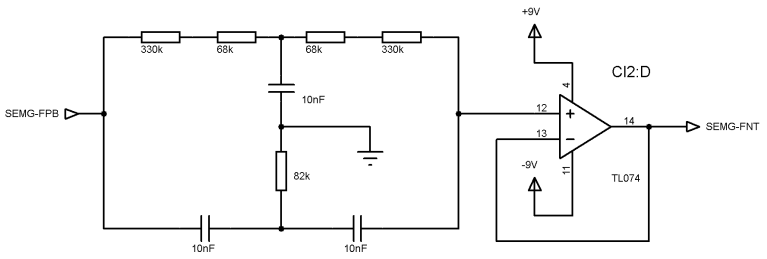


Figura 4.9 – Filtro de *Notch*

Fonte: Adaptado de Teixera et al. (2012)

A frequência central desse filtro está entre as frequências de 56,55 Hz e 62,30 Hz. Esses valores podem ser obtidos utilizando-se as Equações 3.22 e 3.23. Ou seja, a frequência de 60 Hz será atenuada por este filtro. Sua saída ainda é aplicada a um amp-op na configuração de seguidor de tensão, que serve como *buffer* para a próxima etapa do circuito.

4.2.3 Prótese virtual computadorizada

4.2.3.1 Modelo em Blender e Animações Esqueletais

A prótese virtual computadorizada foi construída a partir de um modelo de mão para o Blender, contido na biblioteca *open source* LibHand. (SARIC, 2011) A licença desse modelo

de mão é a *Creative Commons Attribution 3.0* e por isso permite a adaptação do modelo para outros trabalhos. A LibHand é uma biblioteca para reconhecimento de articulações da mão para C++ e MatLab, entretanto neste trabalho apenas o modelo de mão disponível na biblioteca em Blender foi adaptado para a utilização no Java. A Figura 4.10 mostra o modelo de mão para o Blender, cuja armadura de ossos também já estava presente no modelo original.

Figura 4.10 – Modelo de mão para Blender disponível na LibHand

O modelo original de mão não tinha animações implementadas. Para transformar o modelo estático de mão em uma prótese de mão com movimentos, foi necessário utilizar as animações esqueletais do Blender. Para a criação das animações, utilizou-se a estratégia de definir 3 *keyframes* do tipo LocRotScale em um intervalo de 30 *frames*, o primeiro (no *frame* 1) para a posição inicial aberta da mão, o segundo (no *frame* 15) para a posição máxima do movimento e o terceiro (no *frame* 30) novamente para a posição aberta da mão, de forma que o movimento é realizado a partir de uma posição de repouso e termina novamente em uma posição de repouso. O restante dos *frames* foram interpolados pelo Blender usando a interpolação de Bézier. A Figura 4.11 mostra a tela do Blender no momento da criação da animação esqueletal de flexão do dedo médio. As curvas à esquerda são as curvas-F de cada osso interpoladas por meio de curvas de Bézier. A parte da direita da figura mostra o modelo de mão na posição do *keyframe* intermediário, que indica o ponto máximo do movimento.

Seguindo essa metodologia, foram criados os seguintes movimentos para a prótese virtual: flexão dos dedos, flexão do punho, extensão do punho, hiperextensão do punho, desvio radial e ulnar do punho.

A linguagem de programação escolhida para o desenvolvimento do *software* deste projeto foi *Java*, por essa razão o modelo de mão teve que ser exportado do formato do Blender para

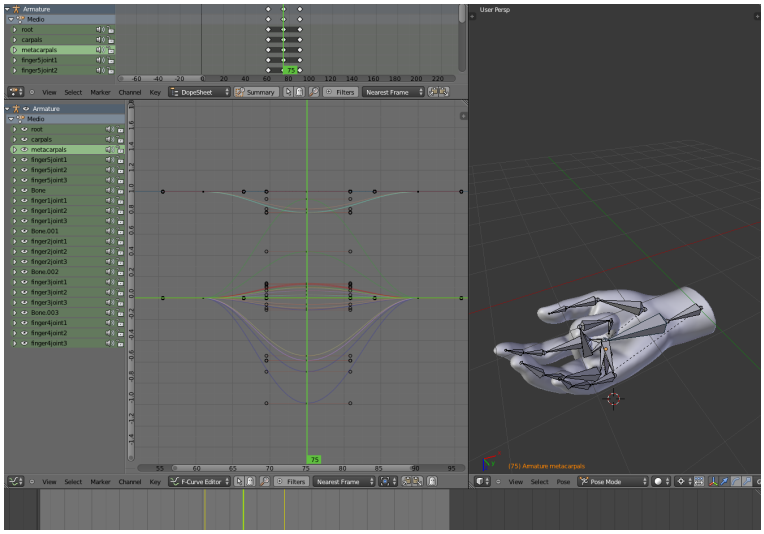


Figura 4.11 – Tela do Blender de construção da animação esqueletal da flexão do dedo médio

um formato reconhecido pelo *Java*. Inicialmente tentou-se exportar o modelo para a biblioteca *Java 3D*, por meio do formato *wavefront*, entretanto essa biblioteca não suportava as animações esqueletais, de forma que era possível exportar apenas o modelo estático, o que era insuficiente para o propósito deste trabalho. Outras bibliotecas de gráficos 3D para *Java* foram pesquisadas e a que demonstrou um bom suporte às animações esqueletais e facilidade de importação de modelos oriundos do Blender foi a *jMonkeyEngine*, e por essa razão foi a escolhida para a implementação da prótese mioelétrica virtual.

Cada movimento feito no Blender foi registrado como uma *action* com um nome representativo do movimento, e adicionado em uma *NlaTrack*. Tomou-se o cuidado para que os diferentes movimentos não ocupassem os mesmos *frames*, para que um movimento não influenciasse no outro. Dessa forma, os movimentos ocuparam os *frames* de 1 a 30, 31 a 60, 61 a 90 e assim sucessivamente. Para a exportação do modelo com as animações esqueletais, utilizou-se o plugin para o Blender *OGRE Exporter*, que transforma um arquivo .blend em dois arquivos XML (eXtensible Markup Language): *.mesh.xml e *.skeleton.xml. O arquivo *.mesh.xml contém as tabelas de polígonos do modelo, com informações geométricas e de atributos. O arquivo *.mesh.xml gerado a partir do modelo de mão contém 27047 vértices e 69885 superfícies poligonais e ocupa aproximadamente 24 MB. O arquivo *.skeleton.xml contém as informações das animações esqueletais e ocupa quase 1MB. Por essa razão, tais arquivos foram disponibilizados no apêndice em CD, juntamente com o arquivo *.blend. O plugin OGRE Exporter pode ser instalado pelo próprio Blender clicando-se em *User Preferences*, *Addons* e buscando por *OGRE Exporter* na categoria *Import-Export*. A versão do Blender utilizada foi a 2.66a e a versão da biblioteca *jMonkeyEngine* foi a 3.

4.2.3.2 Controle dos movimentos da prótese virtual

Depois de realizados alguns testes e verificada a estabilidade do processo de exportação do *Blender* para o *jMonkeyEngine*, foram levantados requisitos para o desenvolvimento de um componente em *Java Swing* que renderizasse a prótese mioelétrica virtual e permitisse seu controle pelos outros componentes do sistema. Os requisitos identificados foram:

a) O componente deve ser um java.awt.Component, ou seja, deve herdar de java.awt.Component direta ou indiretamente, para que possa ser adicionado em outros componentes gráficos do *Swing*.

b) Ele deve isolar o código que utiliza o *jMonkeyEngine* para renderização do restante do sistema, para diminuir o acoplamento e permitir que outras bibliotecas de gráficos 3D sejam utilizadas futuramente.

c) Ele deve prover uma interface por meio da qual outros objetos do sistema são capazes de enviar comandos para a movimentação da prótese.

d) Um comando para a realização de movimento não deverá surtir efeito se um movimento já estiver sendo realizado.

A fim de atender ao requisito de enviar comandos para a prótese mioelétrica virtual, criou-se a enumeração (*enum*) Comando, que representa os comandos de movimentos possíveis da prótese mioelétrica. A Listagem 4.1 apresenta a definição dessa enumeração. Nela, apenas 4 movimentos estão enumerados, entretanto, mais movimentos podem ser adicionados em tempo de compilação, para contemplar novas animações acrescentadas ao modelo 3D.

```
1 public enum Comando {
2   DESVIO_RADIAL("DesvioRadial"), DESVIO_ULNAR("DesvioUlnar"),
       FLEXAO_PUNHO(
3       "FlexaoPunho"), HIPEREXTENSAO_PUNHO("HiperextensaoPunho");
4
5   private final String nome;
6
7   private Comando(String nome) {
8     this.nome = nome;
9   }
10
11  public String getNome() {
12    return nome;
13  }
14 }
```

Listagem 4.1 – Definição da enumeração dos comandos da prótese mioelétrica

As linhas de 2 a 3 definem os movimentos possíveis com seu identificador de nome, que deve corresponder ao nome da *action* do *Blender* na qual foi construída a animação. A variável na linha 5 é declarada final porque uma enumeração não muda seu valor depois de criada. O método público definido nas linhas de 11 a 13 fornece um meio de clientes dessa enumeração recuperarem o nome do movimento.

Criou-se ainda uma interface MaoControlavel para representar objetos que serão controlados por comandos. A Listagem 4.2 apresenta sua definição. Ela contém apenas um método, enviarComando(), que recebe uma enumeração Comando, que indica qual comando está sendo ativado.

```
1 public interface MaoControlavel {
2   void enviarComando(Comando com);
3 }
```

Listagem 4.2 – Interface MaoControlavel

O modelo em *Blender* exportado é carregado por meio de uma extensão da classe SimpleApplication do *jMonkeyEngine*. A Figura 4.12 apresenta o diagrama de classe da classe MaoVirtual, que herda da classe SimpleApplication, uma classe do *jMonkeyEngine* que permite o acesso a recursos de gráficos 3D, e implementa as interfaces AnimEventListener e MaoControlavel. A interface MaoControlavel indica que a classe MaoVirtual poderá receber comandos aos quais reagirá com a execução da animação correspondente. Os comandos são recebidos pelo método enviarComando(), que deve ser implementado pela classe MaoVirtual. A interface AnimEventListener é utilizada para detectar eventos durante uma animação e no escopo da classe MaoVirtual é utilizada para detectar o fim de uma animação, de forma a auxiliar no bloqueio de novas animações enquanto uma animação não tiver sido finalizada. Esse controle de bloqueio de animações é feito pela variável de *flag* estahEmMovimento.

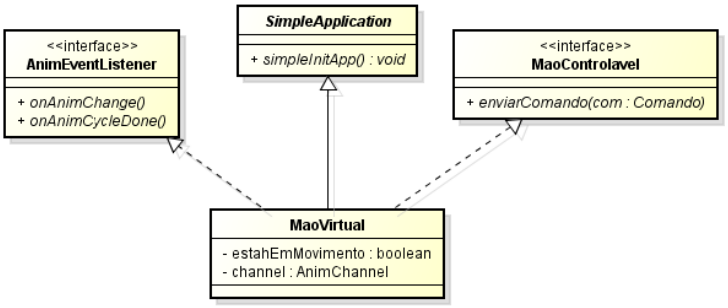


Figura 4.12 – Diagrama de classe da classe MaoVirtual

Por herdar da classe abstrata SimpleApplication, que também possui um método abstrato simpleInitApp(), a classe MaoVirtual é obrigada a implementar esse método. Ele é o método

no qual os recursos de gráficos 3D são carregados, como o modelo exportado do *Blender* e a textura desse modelo. A Listagem 4.3 apresenta a definição desse método.

```
1  public class MaoVirtual extends SimpleApplication implements
       AnimEventListener, MaoControlavel {
2    ...
3   @Override
4   public void simpleInitApp() {
5     getFlyByCamera().setDragToRotate(true);
6
7     flyCam.setMoveSpeed(20);
8     DirectionalLight dl = new DirectionalLight();
9     dl.setDirection(new Vector3f(-0.1f, -1f, -1).normalizeLocal
          ());
10    rootNode.addLight(dl);
11
12    assetManager
13        .registerLocator(
14            Configuracao.getCaminhoModeloMao(),
15            FileLocator.class);
16
17    Spatial mao = assetManager.loadModel("hand.mesh.xml");
18    Material materialTexturaMao = new Material(assetManager,
19        "Common/MatDefs/Misc/Unshaded.j3md");
20    materialTexturaMao.setTexture("ColorMap",
21        assetManager.loadTexture("Textures/hand_texture.png"));
22    mao.setMaterial(materialTexturaMao);
23
24    AnimControl control = mao.getControl(AnimControl.class);
25    control.addListener(this);
26    channel = control.createChannel();
27
28    rootNode.attachChild(mao);
29  }
30    ...
31 }
```

Listagem 4.3 – Método simpleInitApp() da classe MaoVirtual

As linhas de 5 a 7 configuram a câmera para girar e sua velocidade. As linhas de 8 a 10 adicionam a iluminação do cenário 3D, caso contrário nada seria visto. As linhas de 12 a 17 carregam o modelo exportado do *Blender*. As linhas de 17 a 22 adicionam a textura do modelo

3D de mão. As linhas de 24 a 26 configuram as animações e registram a classe MaoVirtual como uma *listener* de eventos de animação.

A classe MaoVirtual ainda não é um componente que pode ser adicionado em uma interface gráfica *Swing*. Dessa forma, torna-se necessário criar uma outra classe que encapsule a renderização da MaoVirtual em um componente reconhecido pelo *Swing*. A classe desenvolvida que desempenha esse papel é a MaoPanel, que herda diretamente de JPanel e utiliza a classe do *jMonkeyEngine* JmeCanvasContext para fornecer a renderização do modelo em um Canvas. A Figura 4.13 apresenta o diagrama de classe da classe MaoPanel. Nele observa-se que MaoPanel herda de JPanel e tem uma associação com uma MaoVirtual. Além disso, ela também implementa a interface MaoControlavel, mas o controle é simplesmente delegado à MaoVirtual com a qual se associa.

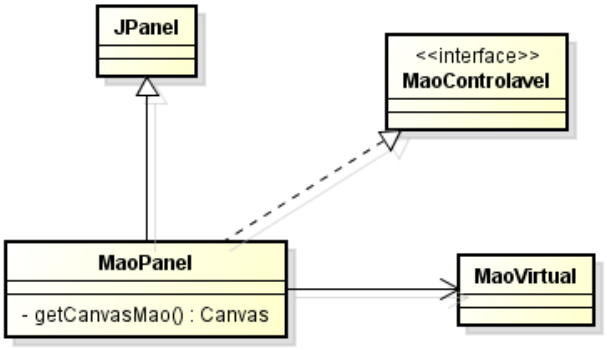


Figura 4.13 – Diagrama de classe da classe MaoPanel

O método chave da MaoPanel é o getCanvasMao(), no qual é instanciado um objeto MaoVirtual e é extraído um Canvas que poderá ser adicionado à MaoPanel. A Listagem 4.4 apresenta a definição do método getCanvasMao().

```
1 public class MaoPanel extends JPanel implements MaoControlavel
    {
2 ...
3   private MaoVirtual maoVirtual;
4   private static Application app;
5   private static JmeCanvasContext context;
6   private static Canvas canvas;
7
8   private Canvas getCanvasMao() {
9     AppSettings settings = new AppSettings(true);
10    settings.setWidth(640);
```

```
11      settings.setHeight(480);
12      settings.setRenderer(AppSettings.LWJGL_OPENGL1);
13
14      maoVirtual = new MaoVirtual();
15      app = maoVirtual;
16
17      app.setPauseOnLostFocus(false);
18      app.setSettings(settings);
19      app.createCanvas();
20      app.startCanvas(true);
21
22      context = (JmeCanvasContext) app.getContext();
23      canvas = context.getCanvas();
24      canvas.setSize(settings.getWidth(), settings.getHeight());
25
26      return canvas;
27    }
28  ...
29  }
```

Listagem 4.4 – Método getCanvasMao() da classe MaoPanel

As linhas de 9 a 12 criam um objeto de configuração com a dimensão da tela de renderização do modelo de mão e a versão do *OpenGL* utilizada. As linhas de 14 e 15 instanciam e guardam referências para a MaoVirtual. As linhas de 17 a 20 aplicam a configuração preparada anteriormente à MaoVirtual e criam um Canvas para renderizar a MaoVirtual. As linhas de 22 a 26 recuperam uma referência ao Canvas criado para retorná-lo do método.

4.2.4 Módulos de entrada, processamento e controle dos EMGs

O módulo de processamento dos SEMGs foi desenvolvido utilizando a linguagem de programação *Java*. Ela foi escolhida por ter um bom suporte de programação concorrente, necessário para o funcionamento dos componentes de *software* do protótipo. Além disso, a existência de bibliotecas para a captura da entrada de áudio do computador, construção de redes neurais e renderização de gráficos 3D também são pontos que levaram a sua escolha. Adicionalmente, a utilização de *Java* como linguagem de implementação do protótipo torna-o portável entre sistemas operacionais e plataforma diferentes, desde que as bibliotecas utilizadas também sejam compatíveis.

4.2.4.1 Entrada dos SEMGs

A entrada para o computador do SEMG produzido no circuito de captura foi feita utilizando-se a entrada de áudio. Dessa forma, a placa de som atuou como um conversor analógico-digital (ADC) de baixo custo, por estar presente na maioria dos computadores pessoais. A taxa de amostragem dessas placas vai de 5.000 Hz até 128.000 Hz e tem uma resolução de 8 bits até 24 bits, sendo mais do que suficiente para a captura do SEMG. Seu intervalo de frequência corresponde ao intervalo do som humanamente audível: de 20 Hz a 20.000 Hz. Como o intervalo de frequência dominante do SEMG é de 50 Hz a 150 Hz, esse sinal pode ser capturado satisfatoriamente por uma placa de som.

A API de som do *Java* foi utilizada para realizar a comunicação baixo nível com a placa de som. Essa API é dividida em dois pacotes principais: javax.sound.sampled e javax.sound.midi. O pacote javax.sound.sampled contém um conjunto de classes e interfaces para captura, mixagem e reprodução de áudio digital amostrado. Já o pacote javax.sound.midi provê uma interface para sintetização, sequenciamento e transporte de eventos MIDI. Neste trabalho, foram utilizadas as classes do pacote javax.sound.sampled para a captura do sinal de áudio.

Apesar de a entrada de áudio estar sendo utilizada como mecanismo de entrada do sinal, o *software* de processamento foi desenvolvido de forma a permitir a extensibilidade para outros mecanismos de entrada do sinal, como pela entrada serial ou USB. Isso foi obtido pela definição de uma interface genérica LeitorDeSinal que é utilizada pelo módulo de processamento, e representa um mecanismo de captura do sinal cujos aspectos de implementação não estão visíveis ao módulo de processamento. Deste modo, para que outros mecanismos de captura de sinal possam ser utilizados, apenas uma implementação da interface deve ser escrita, dispensando qualquer outra alteração no *software*. A Listagem 4.5 mostra a definição dessa interface.

```
1 public interface LeitorDeSinal {
2   InputStream comecarLeitura() throws LeituraDeSinalException;
3   void pararLeitura();
4 }
```

Listagem 4.5 – Definição da interface LeitorDeSinal

O método comecarLeitura() retorna um InputStream, que é uma classe abstrata que representa um *stream* de *bytes*, servindo como mecanismo genérico de transferência de dados entre dois objetos. Além disso, ele pode lançar exceções do tipo LeituraDeSinalException, que indica que algum erro ocorreu durante a leitura do sinal. Para uma implementação adequada de um LeitorDeSinal, o método comecarLeitura() não deve bloquear a execução do programa. Por essa razão, uma implementação deverá criar e iniciar uma *thread* para executar a captura, que será transferida ao módulo de processamento por meio do InputStream. O método pararLeitura() interrompe a leitura do SEMG.

No escopo deste trabalho, apenas a implementação do LeitorDeSinal que utiliza a entrada de áudio para a captura do SEMG foi utilizada efetivamente nos testes. Apesar de que uma

implementação do LeitorDeSinal que captura o SEMG a partir da entrada analógica do Arduíno ter sido desenvolvida, o fato de o conversor AD do Arduíno não conseguir capturar valores negativos do sinal mostrou-se um obstáculo para a utilização desse método de entrada.

A implementação do LeitorDeSinal que realiza a captura pela entrada de áudio foi feita por meio da classe LeitorDeSinalDeAudio, cujo diagrama de classe é apresentado na Figura 4.14. Essa classe implementa a interface LeitorDeSinal e a interface Runnable. A interface Runnable já faz parte da biblioteca padrão do Java e provê um mecanismo de criar código que será executado em uma *thread* separada. Isso é necessário pois a classe LeitorDeSinalDeAudio, em seu método comecarLeitura(), cria uma *thread* para executar continuamente a leitura do SEMG até que seja interrompido pelo método pararLeitura().

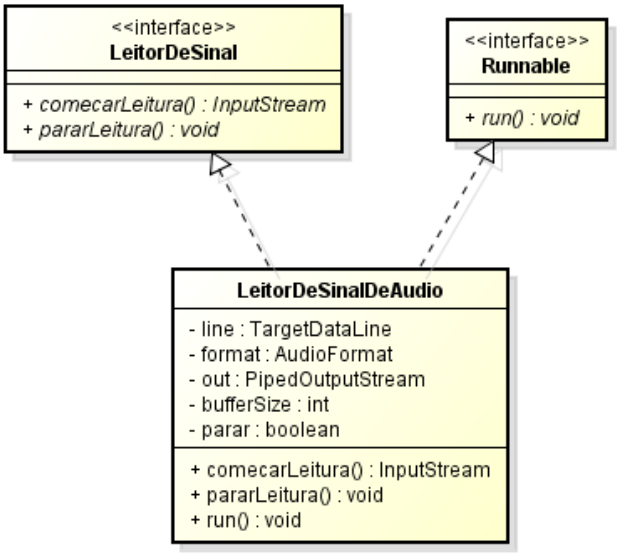


Figura 4.14 – Diagrama de classe do LeitorDeSinalDeAudio

Os atributos *line* e *format* são do tipo TargetDataLine e AudioFormat, respectivamente, e pertencem à API de som do Java. O TargetDataLine representa uma linha de entrada de áudio do computador, construída a partir de um AudioFormat, que por sua vez representa uma configuração de formato de áudio a ser seguida pela linha de entrada. O AudioFormat define alguns parâmetros importantes como a taxa de amostragem da captura, a resolução de uma amostra em bits e a quantidade de canais.

O atributo *out*, do tipo PipedOutputStream é o *stream* no qual a *thread* leitora do SEMG irá gravar as amostras digitalizadas desse sinal. Ele fornece um mecanismo de transferência do SEMG para outras classes que vão utilizar o LeitorDeSinalDeAudio, já que ele é conectado, no método comecarLeitura(), a um PipedInputStream, que é retornado como InputStream (por polimorfismo), ao cliente da classe LeitorDeSinalDeAudio.

O atributo *bufferSize* indica o tamanho do *buffer* do PipedOutputStream e pode ser configurado pelo construtor do LeitorDeSinalDeAudio. O atributo *parar*, que é do tipo *boolean*, é uma variável interna que é utilizada como *flag* de parada da *thread* leitora do SEMG. Por padrão, ela inicia com o valor *false* e quando seu valor é mudado para *true* pelo método pararLeitura(), o código da *thread* leitora (localizado dentro do método run()) termina sua execução e a *thread* é finalizada. O código completo da classe LeitorDeSinalDeAudio pode ser encontrado no Apêndice C.

4.2.4.2 Entrada de amostras do SEMG

As implementações da interface LeitorDeSinal fornecem um mecanismo de transferir *bytes* capturados utilizando algum método específico (entrada de áudio, porta serial, porta USB) para seus clientes, entretanto esses *bytes* transferidos são dependentes do método de entrada do SEMG, o que prejudicaria a transparência percebida por seus clientes em relação ao método de entrada do SEMG. Dessa forma, criou-se uma classe abstrata para encapsular essas diferenças, de forma que apenas a amostra final seja retornada para o cliente, ao invés de componentes de uma amostra (como ocorreria pela leitura do InputStream retornado pelo LeitorDeSinal).

Por exemplo, na leitura da entrada de áudio com uma configuração de resolução de amostra de 16 bits, cada 2 *bytes* do InputStream retornado pelo LeitorDeSinal corresponderia a uma única amostra e realizar esse tratamento de remontagem da amostra no cliente do LeitorDeSinal seria inviável, já que muitas responsabilidades ficariam concentradas nele. A classe abstrata criada foi denominada AmostraInputStream e herda de FilterInputStream, uma classe da biblioteca padrão do Java que representa um filtro sobre um InputStream. O diagrama de classe da AmostraInputStream é apresentado na Figura 4.15.

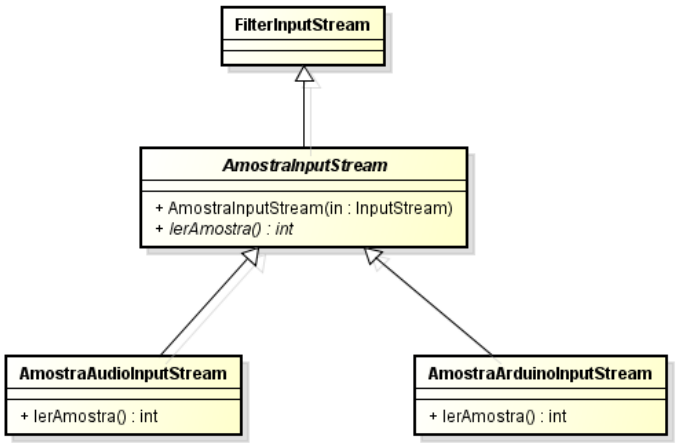


Figura 4.15 – Diagrama de classe da AmostraInputStream

Como pode ser observado na Figura 4.15, uma implementação concreta deve ser fornecida para cada forma diferente de entrada do SEMG. A classe concreta AmostraAudioInputStream,

por exemplo, realiza uma filtragem no InputStream retornado pelo LeitorDeSinalDeAudio de forma a retornar a próxima amostra do sinal disponível. Já a classe AmostraArduinoInputStream realiza uma filtragem sobre o InputStream quando a leitura estiver sendo feita por meio do conversor AD do Arduíno.

Para coordenar a leitura das amostras e notificar possíveis interessados na chegada de amostras de um sinal, desenvolveu-se a classe abstrata LeitorDeAmostras, que mantém referências para objetos da classe AmostraListener. Esses objetos serão notificados quando a estratégia específica do LeitorDeAmostras determinar que existem novas amostras disponíveis. A Figura 4.16 mostra o diagrama de classe da classe LeitorDeAmostras e sua hierarquia de herança.

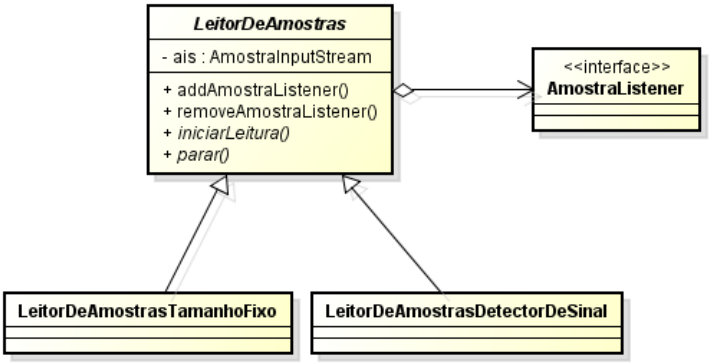


Figura 4.16 – Diagrama de classe do LeitorDeAmostras

Observa-se na Figura 4.16 que a classe LeitorDeAmostras tem um relacionamento de agregação com a interface AmostraListener. Isso ocorre para permitir que objetos que implementem a interface AmostraListener sejam registrados, por meio do método addAmostraListener(), para serem notificados quando novas amostras estiverem disponíveis. O LeitorDeAmostras possui internamente uma instância de AmostraInputStream, que irá encapsular a leitura de amostras, independente da tecnologia de entrada utilizada. O método removeAmostraListener() permite remover uma AmostraListener da lista de objetos que serão notificados.

O método abstrato iniciarLeitura() da classe LeitorDeAmostras será implementado nas classes concretas, e normalmente iniciará uma nova *Thread* para realizar a tarefa de leitura das amostras e notificação dos *listeners* registrados. A classe concreta LeitorDeAmostrasTamanhoFixo utiliza a estratégia de notificar os *listeners* quando uma janela de amostras de tamanho determinado estiver disponível. Já a classe concreta LeitorDeAmostrasDetectorDeSinal utiliza uma estratégia para tentar detectar o início e o término do SEMG, notificando os *listeners* apenas quando encontrar um SEMG utilizando essa estratégia, independente do tamanho detectado.

A Listagem 4.6 mostra a definição da interface AmostraListener. O único método dessa interface amostrasChegaram() será chamado pelo LeitorDeAmostras, assim que um conjunto

de amostras forem detectados, e esse conjunto de amostras será passado como parâmetro desse método. As implementações das outras classes do diagrama da Figura 4.16 podem ser encontradas no Apêndice C.

```
1 public interface AmostraListener {
2   void amostrasChegaram(Integer[] amostras);
3 }
```

Listagem 4.6 – Definição da interface AmostraListener

4.2.4.3 Visualização dos SEMGs

Para realizar a visualização do SEMG foram desenvolvidas várias classes especializadas, todas herdando da classe abstrata RenderizadorSEMG. A Figura 4.17 mostra o diagrama de classes com a hierarquia das classes desenvolvidas. Nela pode-se observar que a raiz da hierarquia, RenderizadorSEMG, herda de JComponent, tornando assim cada classe da hierarquia um componente que pode ser adicionado em interfaces gráficas *Swing*.

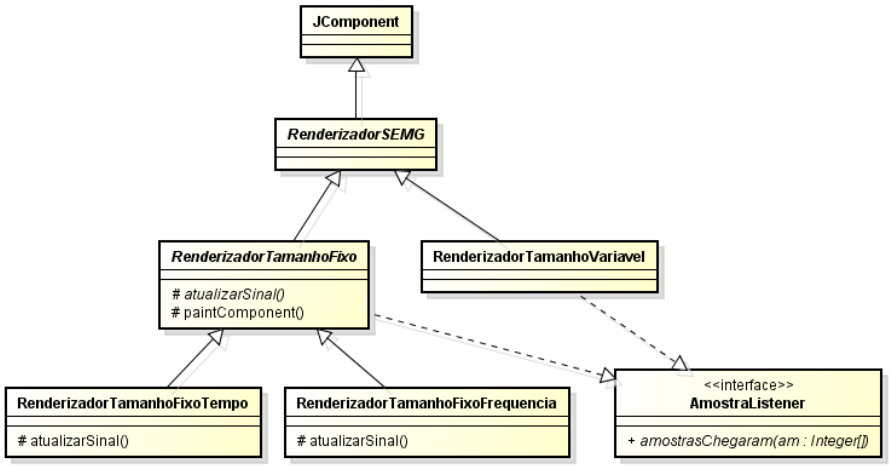


Figura 4.17 – Diagrama de classe com a hierarquia da classe RenderizadorSEMG

A classe abstrata RenderizadorTamanhoFixo provê a funcionalidade de renderizar o SEMG em uma janela com uma quantidade constante de amostras por vez, sendo atualizada sempre que uma nova janela de amostras estiver disponível. Ela implementa o método paintComponent(), que prepara o gráfico de visualização, desenhando os eixos X e Y, e chama o método abstrato atualizarSinal() para desenhar o sinal. Entretanto, a implementação do método atualizarSinal() só estará disponível nas implementações concretas do RenderizadorTamanhoFixo. A Listagem 4.7 mostra a implementação do método paintComponent() da classe RenderizadorTamanhoFixo.

```
1 public abstract class RenderizadorTamanhoFixo extends
    RenderizadorSEMG implements AmostraListener {
```

```
2 ...
3   @Override
4   protected void paintComponent(Graphics g) {
5     super.paintComponent(g);
6
7     Graphics2D g2 = (Graphics2D)g.create();
8
9     int w = getWidth() - getInsets().left - getInsets().right;
10    int h = getHeight() - getInsets().top - getInsets().bottom;
11
12    if (isOpaque()) {
13      g2.setPaint(getBackground());
14      g2.fillRect(0, 0, getWidth(), getHeight());
15    }
16
17    g2.setPaint(Color.BLACK);
18
19    g2.drawLine(0, h / 2, w, h / 2);
20
21    int dottedLineWidth = 3;
22    int iterations = h / (2 * dottedLineWidth);
23
24    for (int i = 0; i < iterations; i++) {
25      int startY = i * 2 * dottedLineWidth;
26      int endY = startY + dottedLineWidth;
27      g2.drawLine(w / 2, startY, w / 2, endY);
28    }
29
30    atualizarSinal(g2, w, h);
31  }
32
33  protected abstract void atualizarSinal(Graphics2D g2, int
        width, int heigth);
34 ...
35 }
```

Listagem 4.7 – Implementação do método paintComponent() da classe RenderizadorTamanhoFixo

As linhas 9 e 10 calculam o tamanho da tela, desconsiderando neste cálculo o tamanho das bordas. As linhas de 12 a 15 desenham o fundo do gráfico (com uma cor branca que foi definida no construtor). A linha 17 muda a cor dos próximos desenhos para preto. A linha 19 desenha

o eixo X na metade da altura do componente. As linhas de 21 a 28 desenham o eixo Y como uma linha pontilhada. Na linha 30, é chamado o método atualizarSinal(), cuja implementação estará disponível nas classes concretas que herdarem de RenderizadorTamanhoFixo. Ele tem o objetivo de atualizar a representação do sinal na tela e dependerá de cada estratégia utilizada pelas subclasses.

A classe RenderizadorTamanhoFixoTempo é apropriada para representar sinais no domínio do tempo. Para renderizar o sinal, ela utiliza a estratégia de unir dois pontos de duas amostras consecutivas, formando um gráfico contínuo. A implementação do método atualizarSinal() utilizando essa estratégia é mostrado na Listagem 4.8. O fatorEscalaVertical é um parâmetro que ajusta a escala vertical do sinal.

```
1 public class RenderizadorTamanhoFixoTempo extends
    RenderizadorTamanhoFixo {
2 ...
3   @Override
4   protected void atualizarSinal(Graphics2D g2, int width, int
      heigth) {
5     int amplitude;
6     int yMeio = heigth / 2;
7     int amplitudeAnterior = 0;
8     for (int i = 0; i < amostras.length; i++) {
9       amplitude = amostras[i];
10
11      g2.drawLine(i, yMeio - amplitudeAnterior /
          fatorEscalaVertical, i, yMeio
12          - amplitude / fatorEscalaVertical);
13
14      amplitudeAnterior = amplitude;
15    }
16  }
17 ...
18 }
```

Listagem 4.8 – Implementação do método atualizarSinal() da classe RenderizadorTamanhoFixoTempo

A classe RenderizadorTamanhoFixoFrequencia é apropriada para representar sinais no domínio da frequência. Para renderizar o sinal, ela utiliza a estratégia de desenhar impulsos verticais em cada ponto de amostra. A implementação de seu método atualizarSinal() pode ser encontrada no Apêndice C.

A classe RenderizadorTamanhoVariavel renderiza o SEMG sem utilizar uma janela constante, de modo que o comprimento desse componente varia juntamente com o tamanho do

SEMG recebido. Ele pode ser utilizado quando se deseja capturar o SEMG e mostrá-lo na íntegra. Sua implementação completa também está disponível no Apêndice C.

Ambas as classes RenderizadorTamanhoFixo e RenderizadorTamanhoVariavel implementam a interface AmostraListener. Essa interface fornece um meio de registrar um objeto para que seja notificado quando novas amostras do SEMG estiverem disponíveis. Assim, esses componentes poderão ser registrados a um LeitorDeAmostras, para que sejam notificados de novas amostras e possam atualizar a interface gráfica.

4.2.4.4 Transformada Rápida de Fourier

Para realizar a transformada rápida de Fourier sobre o SEMG, utilizou-se a classe FFT da universidade de Princeton, juntamente com a classe Complex, que é utilizada para representar números complexos. (SEDGEWICK; WAYNE, 2012) A Listagem 4.9 apresenta essa implementação da FFT.

```
1  public class FFT {
2    public static Complex[] fft(Complex[] x) {
3      int N = x.length;
4
5      if (N == 1)
6        return new Complex[] { x[0] };
7
8      if (N % 2 != 0) {
9        throw new RuntimeException("N␣is␣not␣a␣power␣of␣2");
10     }
11
12     Complex[] even = new Complex[N / 2];
13     for (int k = 0; k < N / 2; k++) {
14       even[k] = x[2 * k];
15     }
16     Complex[] q = fft(even);
17
18     Complex[] odd = even;
19     for (int k = 0; k < N / 2; k++) {
20       odd[k] = x[2 * k + 1];
21     }
22     Complex[] r = fft(odd);
23
24     Complex[] y = new Complex[N];
25     for (int k = 0; k < N / 2; k++) {
26       double kth = -2 * k * Math.PI / N;
```

```
27      Complex wk = new Complex(Math.cos(kth), Math.sin(kth));
28      y[k] = q[k].plus(wk.times(r[k]));
29      y[k + N / 2] = q[k].minus(wk.times(r[k]));
30    }
31    return y;
32  }
33  ...
34 }
```

Listagem 4.9 – Cálculo da FFT de um sinal discreto

Na Listagem 4.9, o método fft() calcula a DFT de um sinal complexo discreto, que é recebido como parâmetro (no Java, um *array* da classe Complex), utilizando-se o algoritmo de Cooley-Tukey. As linhas 5 e 6 verificam se é o caso base do algoritmo recursivo, ou seja, se o sinal cuja FFT será calculada tem comprimento igual a 1. Nesse caso, a FFT é simplesmente a própria amostra do sinal original. As linhas de 8 a 10 verificam se o sinal tem comprimento potência de 2, e caso não tenha, lança uma exceção informando que a FFT não poderá ser calculada.

As linhas de 12 a 16 começam o processo recursivo, chamando o próprio método fft() para calcular a FFT das amostras pares (que tem um comprimento $\frac{N}{2}$). Esse resultado é armazenado em uma variável temporária q. As linhas de 18 a 22 continuam o processo recursivo, chamando o próprio método fft() para calcular a FFT das amostras ímpares (que tem um comprimento $\frac{N}{2}$). Esse resultado é armazenado em uma variável temporária r.

Finalmente, nas linhas de 24 a 30, as duas DFTs de comprimento $\frac{N}{2}$ armazenadas em q e r são utilizadas para reconstruir a decimação no tempo, utilizando o componente de cálculo borboleta, já apresentado no Capítulo 3. A linha 31 retorna a DFT resultante como uma sequência de números complexos (um *array* de Complex, no Java).

4.2.4.5 Implementação da Rede Neural Artificial

Para melhorar o controle da prótese mioelétrica virtual, construiu-se uma rede neural artificial (RNA) do tipo *feedforward backpropagation* utilizando-se 3 camadas, uma camada de entrada, uma camada oculta e uma camada de saída. A Figura 4.18 mostra a arquitetura da rede neural construída, com 4 neurônios na camada de entrada, 7 neurônios na camada oculta e 4 neurônios na camada de saída.

Utilizou-se 4 características extraídas do SEMG como entrada da rede neural, *IEMG*, *MAV*, *MNF* e *MDF*, já discutidas na Seção 3.6. A quantidade de neurônios da camada oculta foi determinada experimentalmente como uma quantidade que fornecia uma taxa de erro razoável com um menor número de iterações de treinamento. Cada um dos 4 neurônios da camada de saída representa um movimento dos movimentos escolhidos: flexão do punho, hiperextensão do punho, desvio radial e desvio ulnar.

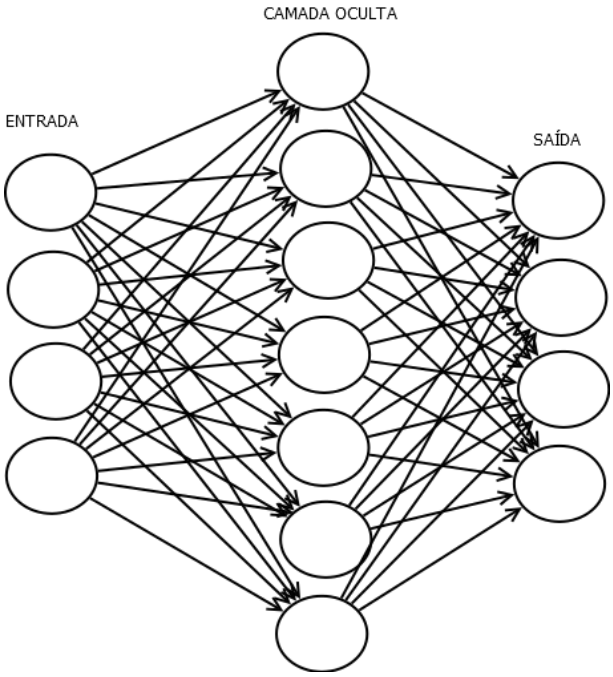


Figura 4.18 – Arquitetura da rede neural construída

Nas camadas oculta e de saída, utilizou-se a função de ativação sigmoid. O algoritmo de treinamento utilizado foi o *backpropagation*, realizando um treinamento com até 5000 iterações ou até que o erro seja menor que 0,1%.

Para a implementação em *Java* da rede neural, utilizou-se a biblioteca de redes neurais *Encog* versão 3.1.0. (HEATON RESEARCH, 2013) Desenvolveu-se a classe RedeNeuralSEMG, que utiliza o *framework Encog* para construir uma rede neural com a arquitetura da Figura 4.18. A Figura 4.19 mostra o diagrama de classe da classe RedeNeuralSEMG.

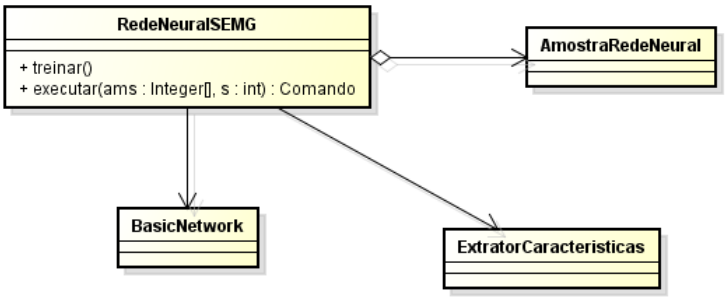


Figura 4.19 – Diagrama de classe da classe RedeNeuralSEMG

A classe RedeNeuralSEMG utiliza a classe SimpleNetwork do *Encog* para a implementação da rede neural. Ela forma uma agregação com a classe AmostraRedeNeural, que formam o conjunto de amostras para treinamento da rede neural. O método treinar() constrói a rede neural e

realiza o seu treinamento utilizando as amostras da lista de objetos da classe AmostraRedeNeural. A Listagem 4.10 mostra a definição desse método.

```
1 public class RedeNeuralSEMG {
2 ...
3   public void treinar() {
4     network = new BasicNetwork();
5     network.addLayer(new BasicLayer(null, true, 4));
6     network.addLayer(new BasicLayer(new ActivationSigmoid(),
        true, 7));
7     network.addLayer(new BasicLayer(new ActivationSigmoid(),
        false, 4));
8     network.getStructure().finalizeStructure();
9     network.reset();
10
11    MLDataSet trainingSet = new BasicMLDataSet(getEntrada(),
        getSaida());
12
13    final Backpropagation train = new Backpropagation(network,
        trainingSet);
14
15    int epoch = 1;
16
17    do {
18      train.iteration();
19      System.out
20         .println("Epoch␣#" + epoch + "␣Error:" + train.getError
              ());
21      epoch++;
22    } while ((epoch < 5000) && (train.getError() > 0.001));
23
24  }
25 ...
26 }
```

Listagem 4.10 – Criação e treinamento da rede neural

As linhas de 4 a 9 montam a rede neural utilizando a classe BasicNetwork e adicionando camadas utilizando a classe BasicLayer, da *Encog*. A classe ActivationSigmoid implementa a função de ativação sigmoid. As linhas de 11 a 22 preparam o algoritmo de treinamento *backpropagation*, que é implementado pela classe do *Encog* Backpropagation. Em seguida é realizado o treinamento da rede neural com um total de 5.000 iterações ou um erro menor do

que 0,1%.

A classe ExtratorCaracteristicas calcula as 4 características das Equações 3.46, 3.47, 3.48 e 3.49. Seu código-fonte completo pode ser encontrado no Apêndice C.

O método executar() da classe RedeNeuralSEMG() recebe um SEMG e retorna o comando detectado pelo padrão por meio da rede neural. Seu código-fonte completo também pode ser encontrado no Apêndice C. Ele utiliza o método compute() da classe BasicNetwork para calcular os valores de cada neurônio de saída. O neurônio com uma saída mais alta corresponde ao movimento detectado para o SEMG e é utilizado para construir um *enum* Comando, que é retornado pelo método.

5 Teste e resultados do protótipo de prótese mioelétrica virtual

5.1 Metodologia de testes

Como o protótipo de prótese mioelétrica virtual é composto de muitos subcomponentes, a metodologia de testes que se julgou mais adequada foi a de testar cada subcomponente isoladamente (testes de unidade), e apenas depois de verificado o correto funcionamento de cada um destes, realizar os testes do sistema com seus componentes já integrados.

5.2 Testes

5.2.1 Componente Renderizador do SEMG

O componente renderizador do SEMG é um componente Java Swing cuja função é capturar o SEMG de alguma entrada de sinal e desenhá-lo na tela da interface gráfica. Para a realização deste teste foi utilizado o Labcenter Proteus ISIS 7.7 como mecanismo gerador de sinais. O ISIS possui um gerador de sinais e um auto-falante, de forma que é possível gerar sinais que serão transmitidos à saída de som. Utilizando-se uma placa de som com suporte à mixagem estéreo, é possível transformar uma saída de som qualquer em uma entrada virtual, permitindo simular um ambiente em que um SEMG real seria aplicado à entrada. A parte esquerda da Figura 5.1 mostra a configuração de um circuito montado no ISIS para envio de um sinal à saída de som e a parte da direita mostra a configuração de mixagem estéreo sendo ativada, em um sistema operacional Windows 8, para permitir que a saída de som seja enviada para uma entrada de som virtual.

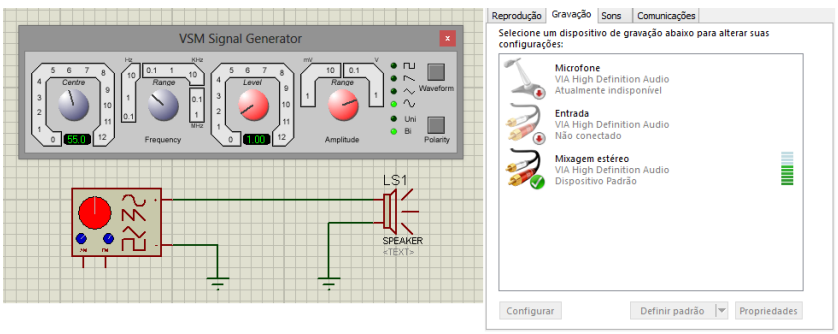


Figura 5.1 – Tela do Proteus ISIS com um circuito gerador de sinais e a configuração de som para ativação da mixagem estéreo

A utilização desse mecanismo de simulação de sinais mostrou-se bastante útil durante o desenvolvimento e testes dos componentes de *software*, tendo em vista que a utilização do cicuito de captura durante o desenvolvimento não é muito prática, já que necessita que toda a configuração seja montada e os eletrodos posicionados no braço de uma pessoa.

Para testar o componente renderizador do SEMG, seguiu-se o roteiro de teste da Tabela 5.1. O resultado das etapas 1 e 2 da Tabela 5.1 são mostrados na Figura 5.2 e o resultado da etapa 3 é mostrado na Figura 5.3.

Tabela 5.1 – Roteiro de teste do Componente Renderizador do SEMG

Ordem	**Ação**	**Checagem**
1	Gerar um sinal senoidal de frequência 200 Hz	Verificar a forma de onda na tela do componente
2	Diminuir a frequência do sinal senoidal gerado no passo 1 para 100 Hz	Verificar a forma de onda na tela do componente e conferir se menos picos e vales aparecem na tela devido a diminuição da frequência
3	Gerar um sinal que seja a soma de 2 sinais senoidais, um de 180 Hz e outro de 90 Hz	Verificar a forma de onda na tela do componente e conferir com uma onda gerada por um *software* de gráficos matemáticos

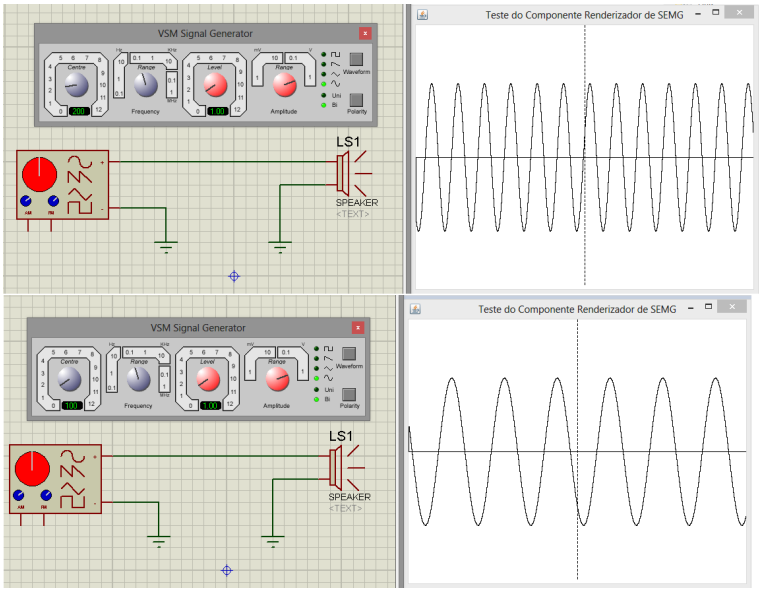


Figura 5.2 – Resultado das etapas 1 e 2 do roteiro de teste da Tabela 5.1

Na parte superior da Figura 5.2 pode-se observar que o renderizador mostrou corretamente um sinal senoidal, como era esperado na etapa 1 da Tabela 5.1. Já na parte inferior da Figura 5.2 observa-se que novamente um sinal senoidal é renderizado, mas dessa vez com a metade da quantidade de picos e vales do sinal na parte superior. Isso indica que esse sinal tem a metade da frequência do sinal da parte superior, validando a etapa 2 da Tabela 5.1.

Na Figura 5.3 observa-se que o sinal renderizado aparentemente tem duas componentes. Entretanto, para validar o sinal, utilizou-se o *Geogebra* para plotar o gráfico da função $f(x) =$

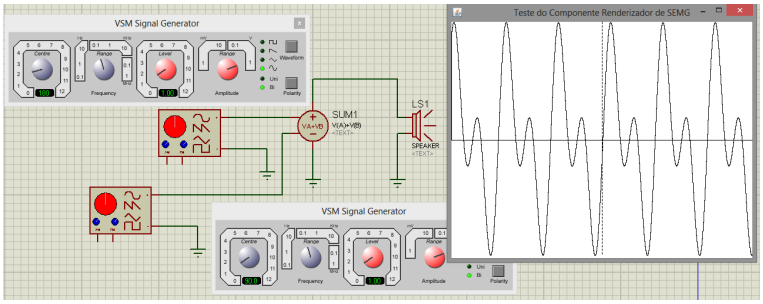


Figura 5.3 – Resultado da etapa 3 do roteiro de teste da Tabela 5.1

$\sin(90 \cdot 2\pi x) + \sin(180 \cdot 2\pi x)$ (uma função senoidal com duas componentes, uma de 90 Hz e outra de 180 Hz). Essa plotagem é mostrada na Figura 5.4 e valida o sinal renderizado na Figura 5.3 como sendo composto por duas componentes, uma de 90 Hz e outra de 180 Hz.

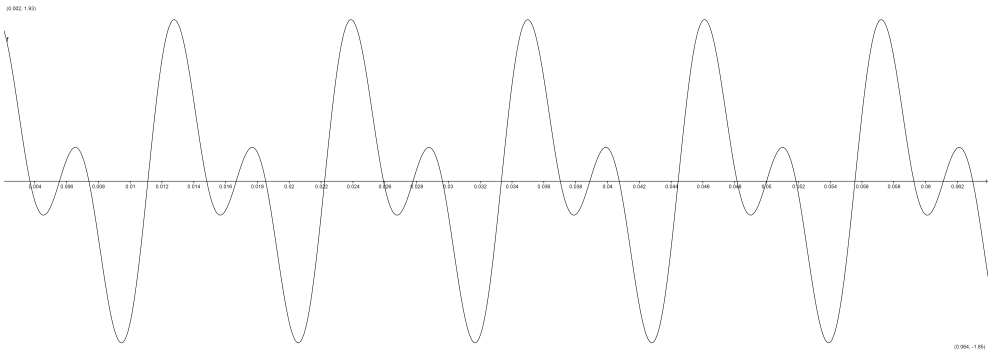


Figura 5.4 – Plotagem da função $f(x) = \sin(90 \cdot 2\pi x) + \sin(180 \cdot 2\pi x)$ feita no *Geogebra*

5.2.2 Circuito de captura, amplificação e filtragem usando o AD620

O circuito de captura, amplificação e filtragem do SEMG construído foi testado utilizando eletrodos da série Kendall Medi-Trace na configuração bipolar, posicionados no músculo flexor ulnar do carpo. No teste antes da digitalização do SEMG, utilizou-se um osciloscópio Tektronix TDS 1001C-30EDU para averiguar o correto funcionamento de cada etapa de filtragem/amplificação e o nível do SEMG em relação ao ruído em cada etapa.

Primeiramente, testou-se o circuito de amplificador para instrumentação usando o CI AD620 com um resistor de $47\,\Omega$ e o potenciômetro ajustado para $0\,\Omega$, a fim de se obter um ganho de aproximadamente 990. O sinal resultante com o músculo em repouso foi capturado com o osciloscópio e é mostrado na Figura 5.5. Nela pode-se perceber que existe uma interferência de 60 Hz, como indicado pela medida do osciloscópio na parte inferior direita. Além disso, o sinal está deslocado abaixo do eixo X, provavelmente devido a alguma componente DC que foi amplificada. Sua amplitude de saída foi de aproximadamente 1 V no repouso.

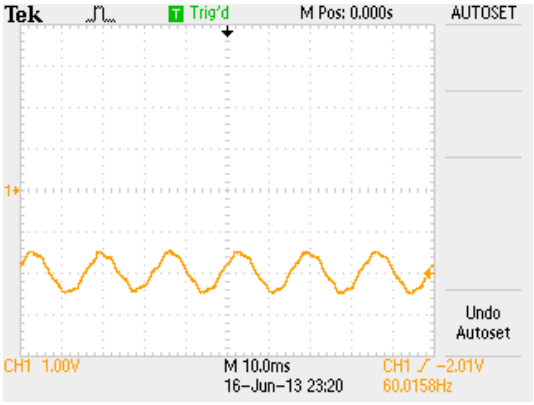


Figura 5.5 – Tela do osciloscópio medindo a saída do AD620 com o músculo em repouso

Na sequência, realizou-se a mesma medição da saída do CI AD620 enquanto se realizava o movimento de flexão do punho. Essa medição é mostrada na Figura 5.6. Nela constata-se que o SEMG está sendo capturado e amplificado corretamente pelo AD620, apesar de componentes DC e ruídos da rede elétrica (60 Hz) ainda estarem presentes nesta etapa.

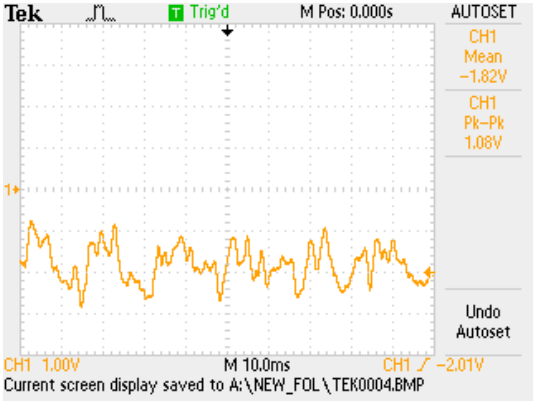


Figura 5.6 – Tela do osciloscópio medindo a saída do AD620 com o músculo em contração

Em seguida, realizaram-se as medições do músculo em repouso e em contração na saída após a passagem do filtro passa-alta (com corte em 10 Hz) e do filtro passa-baixa (com corte em 500 Hz). A Figura 5.7 mostra em sua parte esquerda o SEMG no repouso do músculo e em sua parte direita o SEMG durante a contração muscular do movimento de flexão do punho. Nela observa-se que o ruído de 60 Hz ainda permanece, mas a componente DC presente nas Figura 5.5 e Figura 5.6 foi eliminada com sucesso. Esse resultado era esperado, já que o filtro passa-alta com frequência de corte de 10 Hz rejeita a componente DC de 0 Hz.

Na última etapa de filtragem, em que o sinal de saída dos filtros passa-baixa e passa-alta são aplicados a um filtro de *Notch* com frequência central de 60 Hz para filtragem da interferência da rede elétrica, realizou-se a medição do SEMG com o músculo em repouso e durante a contração do movimento de flexão do punho. Essas medições são mostradas na Figura 5.8: na parte esquerda, a medição do SEMG com o músculo em repouso, e na parte direita, a medição durante

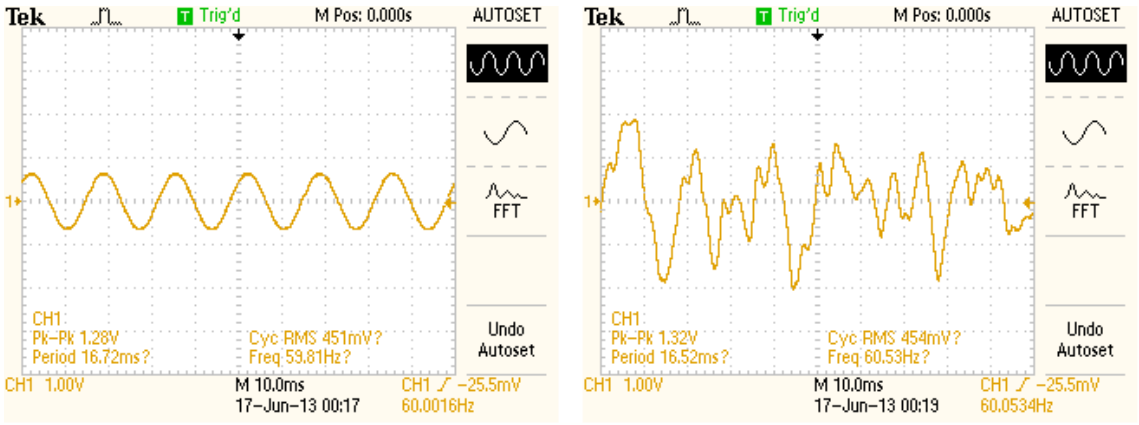


Figura 5.7 – Tela do osciloscópio medindo a saída dos filtros passa-alta e passa-baixa

a contração muscular. Como observa-se na parte esquerda da Figura 5.8, o ruído de 60 Hz oriundo da rede elétrica foi atenuado de uma amplitude pico a pico de 1,28 V, observado na Figura 5.7, para uma amplitude de 160 mV, aproximadamente. Ou seja, o filtro de *Notch* teve um funcionamento satisfatório, apesar das limitações das faixas de tolerância dos capacitores utilizados (20%) e do método artesanal de confecção da placa de circuito impresso.

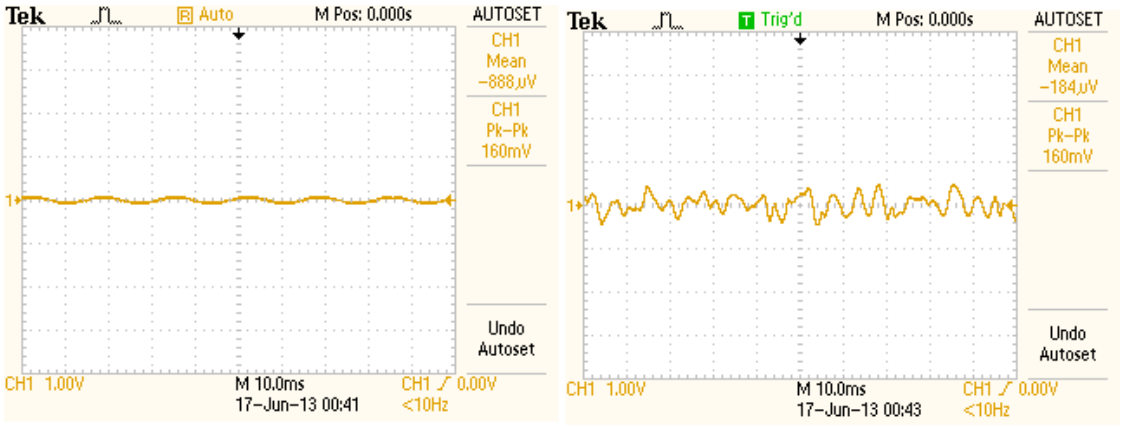


Figura 5.8 – Tela do osciloscópio medindo a saída do filtro de *Notch*

5.2.3 Prótese virtual computadorizada

A prótese virtual de mão foi testada adicionando-se o componente *Swing* desenvolvido a um JFrame e posicionando botões para realizar o controle, validando assim os requisitos definidos. Além disso, tentou-se realizar um movimento enquanto outro movimento já estava sendo realizado. A Figura 5.9 mostra a tela de realização do teste durante os movimentos de desvio radial (esquerda) e desvio ulnar do punho (direita). Como pode-se observar na Figura 5.9, esses movimentos estão de acordo com os movimentos esperados, que estão ilustrados na Figura 2.13.

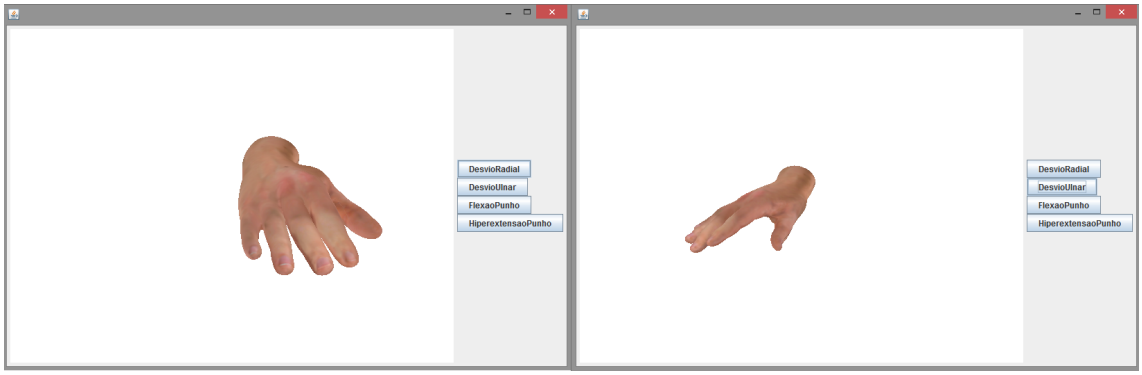


Figura 5.9 – Teste do componente *Swing* renderizador da prótese mioelétrica virtual durante os movimentos de desvio radial e desvio ulnar do punho

Os 4 botões adicionados no teste da Figura 5.9 acionaram corretamente os movimentos correspondentes. Além disso, ao tentar clicar em um botão enquanto algum movimento já estivesse sendo realizado, o movimento do botão clicado era ignorado, atingindo portanto os requisitos definidos na Seção 4.2.3.2. A Figura 5.10 mostra a prótese realizando os movimentos de flexão (esquerda) e hiperextensão do punho (direta), que estão de acordo com os movimentos esperados, ilustrados na Figura 2.10.

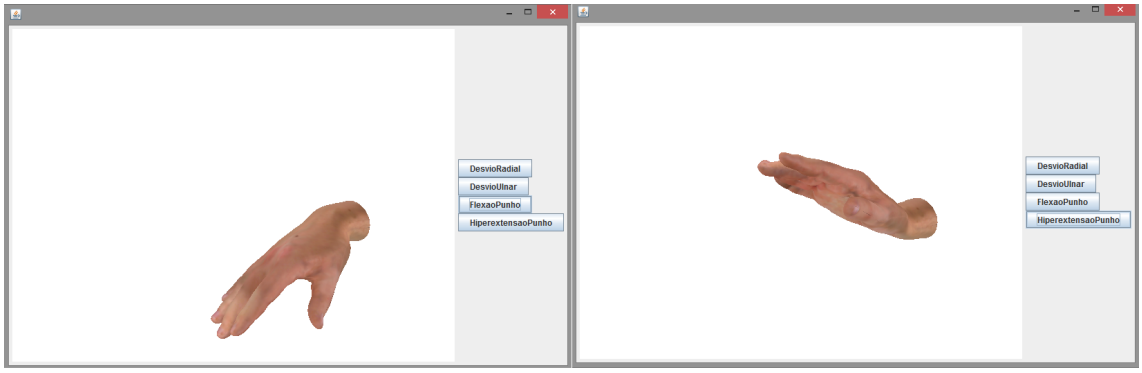


Figura 5.10 – Teste dos movimentos de flexão e hiperextensão do punho

5.2.4 Interface de Entrada do SEMG e Transformada Rápida de Fourier

Os módulos de entrada e processamento por meio da FFT foram testados utilizando-se eletrodos da série Kendall Medi-Trace na configuração bipolar, posicionados no músculo flexor ulnar do carpo. Utilizou-se um cabo RCA/P2, em que o conector RCA foi conectado à saída do circuito e o conector P2 foi conectado na entrada de microfone de um PC. Configurou-se uma taxa de amostragem de 8.000 Hz. A Figura 5.11 mostra em sua parte esquerda o sinal capturado com o músculo em repouso e em sua parte direita o sinal capturado durante a contração muscular.

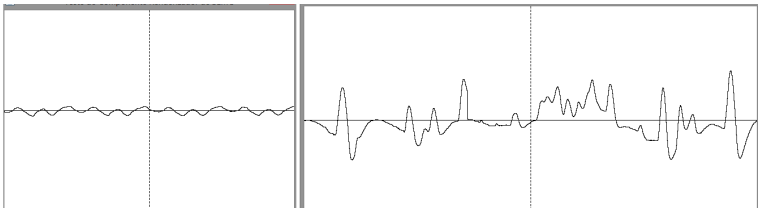


Figura 5.11 – Teste da interface de entrada do SEMG.

Como pode ser observado na Figura 5.11, o SEMG foi capturado com sucesso no computador, por meio da entrada de áudio, pois esta captura está de acordo com a captura do osciloscópio da Figura 5.8. Com o objetivo de testar a implementação da FFT, realizou-se a mesma medida, mas agora ativando-se também o componente renderizador da FFT com uma janela de 512 amostras. O resultado obtido é mostrado na Figura 5.12.

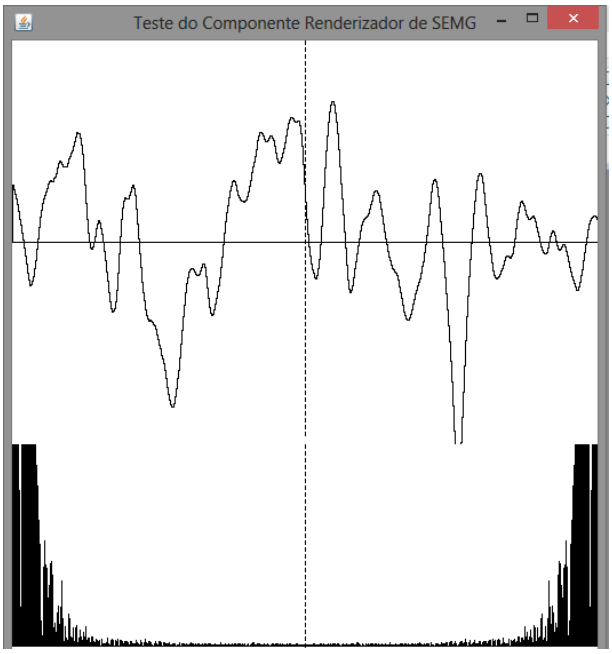


Figura 5.12 – Teste do componente renderizador da FFT aplicada ao SEMG

Como pode-se observar no espectro de frequência da Figura 5.12, há uma concentração das frequências do SEMG nos valores mais próximos da frequência 0 (mais à esquerda) do que da frequência de amostragem de 8000 Hz (mais ao centro). Além disso, nota-se que o espectro de frequência é simétrico em relação ao ponto central do gráfico. Essa é uma das características da DFT e isso indica que a DFT está sendo calculada corretamente.

Um outro teste foi realizado para validar a FFT, agora com sinais gerados no ISIS. Foram gerados dois sinais senoidais, um de 200 Hz e outro de 500 Hz. A Figura 5.13 mostra em sua parte esquerda a FFT do sinal de 200 Hz e em sua parte direita a FFT do sinal de 500 Hz. Como pode-se observar, o sinal de 200 Hz (da esquerda) sofre o fenômeno do vazamento espectral, já que 200 Hz não é múltiplo da frequência fundamental ($\frac{8000}{512} = 15,625$). Já o sinal da direita, de

500 Hz, não sofre vazamento espectral (sua potência é concentrada em apenas uma frequência na FFT). Isso pode ser explicado pelo fato de que 500 Hz é múltiplo da frequência fundamental ($500 = 32 \times 15,625$).

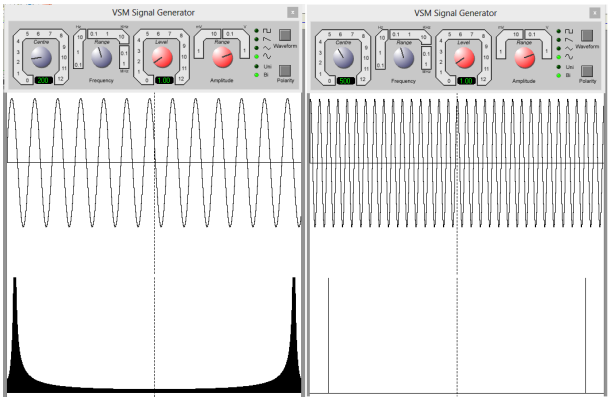


Figura 5.13 – Teste do componente renderizador da FFT aplicada em dois sinais senoidais gerados

5.2.5 Controle da prótese mioelétrica virtual utilizando redes neurais

Para a realização do teste da prótese mioelétrica virtual, escolheu-se 4 movimentos: flexão do punho, hiperextensão do punho, desvio radial e desvio ulnar. Utilizou-se a tela de treinamento da rede neural para efetuar os treinamentos. Essa tela é mostrada na Figura 5.14 e provê uma interface para iniciar ou parar uma medida, armazenar a medida, treinar a rede neural e testar a rede neural.

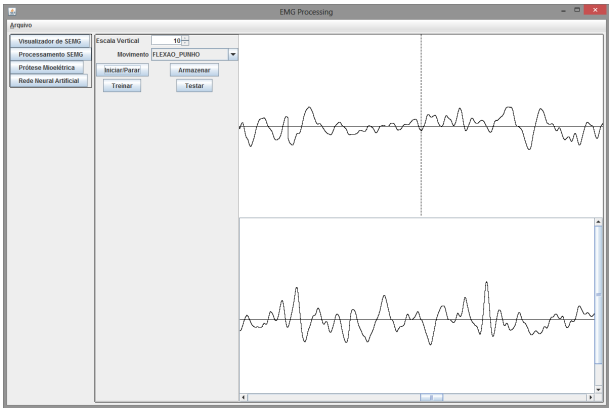


Figura 5.14 – Teste do treinamento da rede neural

Realizou-se o treinamento da rede neural com os 4 movimentos escolhidos, coletando-se 10 medidas de cada movimento. Ao realizar o treinamento, obteve-se uma taxa de erro de 17%. A tela de controle da prótese mioelétrica virtual é mostrada na Figura 5.15. Nela, ao clicar no botão Iniciar/Parar pode-se iniciar ou parar uma medida. Ao término de uma medida, a rede

neural é usada para detectar o movimento, e em seguida a prótese realiza o movimento indicado pela rede neural.

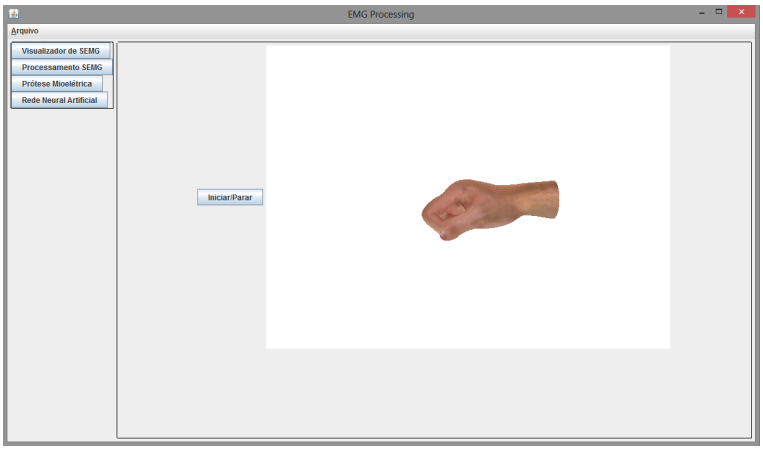


Figura 5.15 – Teste da prótese de mão virtual controlada pela rede neural

Durante o teste, a prótese realizou o movimento detectado pela rede neural apenas após o término da realização da medida, o que causou um pequeno *delay* entre o movimento feito pela pessoa e o movimento da prótese virtual.

5.3 Apresentação da área de aplicação do modelo

O protótipo desenvolvido pode ser aplicado na área médica, pois permite a visualização do SEMG, além de permitir seu processamento digital em um PC convencional. Dessa forma, pode servir como um eletromiógrafo de custo reduzido, especialmente em clínicas fisioterapêuticas em comunidades carentes, tendo em vista seu custo ser bem menor ao de um eletromiógrafo profissional (que pode custar alguns milhares de reais).

Além disso, o protótipo de prótese mioelétrica virtual tem um grande potencial de aplicação na área de reabilitação para amputados. Apesar de a taxa de erro do protótipo ainda ser alto, de 17%, o estudo de outras arquiteturas de redes neurais e a utilização de mais características do sinal pode levar a taxas de erro menores. O protótipo desenvolvido fornece uma base para trabalhos futuros que poderão explorar mais técnicas de processamento digital de sinais, tendo em vista que a parte básica de captura, amplificação e filtragem do SEMG já foi desenvolvida e documentada neste trabalho.

Uma outra aplicação que pode ter como base o protótipo construído é o controle de uma prótese física. Desde que seja utilizado um processadaor digital de sinais, é possível transferir a rede neural para esse chip, permitindo o controle de uma prótese real. Ainda assim a prótese virtualizada pode continuar sendo utilizada na etapa de treinamento da rede neural, que exige um maior processamento do que a etapa da execução da rede neural.

Por ter um tamanho reduzido e funcionar utilizando-se apenas 2 bateriais de 9 V, o protótipo desenvolvido poderia ser utilizado como um eletromiógrafo portátil. Seria possível alterá-lo para mostrar o SEMG no visor de um *smartphone*, por exemplo, e permitir seu uso em ambientes esportivos para detecção rápida de lesões. Uma outra aplicação deste protótipo de tamanho reduzido seria sua utilização em academias, para medir o nível de contração muscular durante uma série de exercícios e verificar o desempenho muscular e a correta execução do exercício.

5.4 Custos do protótipo proposto

Estimaram-se apenas os custos dos componentes físicos do protótipo. Como o *software* foi desenvolvido pelo autor, utilizando-se de bibliotecas e ferramentas de desenvolvimento gratuitas, o custo da parte de *software* foi zero. Entretanto, seria possível estimar quanto custaria o *software* no mercado utilizando-se abordagens como Análise de Pontos de Função (APF). A Tabela 5.2 detalha o custo de cada componente e o custo total.

Tabela 5.2 – Custos do protótipo proposto

Materiais	**Quantidade**	**Custo**	**Total**
Placa de fenolite virgem (10x15 cm)	1	R$ 3,10	R$ 3,10
Bateria de 9 V	2	R$ 3,90	R$ 7,80
CI AD620AN	1	R$ 55,00	R$ 55,00
CI TL074CN	2	R$ 0,90	R$ 1,80
Trimpot Multivoltas Vertical 10K	2	R$ 0,90	R$ 1,80
Resistor de 3,3 K	5	R$ 0,04	R$ 0,20
Resistor de 47 R	1	R$ 0,04	R$ 0,04
Resistor de 82 K	2	R$ 0,04	R$ 0,08
Resistor de 330 K	2	R$ 0,04	R$ 0,08
Resistor de 1 K	2	R$ 0,04	R$ 0,08
Resistor de 10 K	4	R$ 0,04	R$ 0,16
Resistor de 68 K	2	R$ 0,04	R$ 0,08
Resistor de 33 R	1	R$ 0,04	R$ 0,04
Capacitor cerâmico de 10 nF	3	R$ 0,04	R$ 0,12
Capacitor cerâmico de 100 nF	2	R$ 0,05	R$ 0,10
Conector DB-9 Fêmea 180°	1	R$ 0,56	R$ 0,56
Conector DB-9 Macho 90°	1	R$ 1,33	R$ 1,33
Capa curta para conector DB9 180°	1	R$ 0,67	R$ 0,67
Cabo manga blindado 70 cm	1	R$ 2,50	R$ 2,50
Clipe para bateria 9V tipo T	2	R$ 0,44	R$ 0,88
Soquete DIP de 8 pinos	1	R$ 0,12	R$ 0,12
Soquete DIP de 14 pinos	2	R$ 0,15	R$ 0,30
Garra de jacaré	3	R$ 0,40	R$ 1,20
Cabo RCA/P2	1	R$ 5,99	R$ 5,99
Total			R$ 84,03

Verifica-se que o custo total do protótipo (R$ 84,03) é bem inferior ao de um eletromiógrafo profissional, que pode custar alguns milhares de reais. Obviamente a precisão de um eletromiógrafo profissional é bem maior do que ao do protótipo desenvolvido, entretanto o protótipo tem uma precisão suficiente para a realização de uma análise básica do SEMG.

6 Conclusão

6.1 Conclusões

O SEMG, por ter uma amplitude muito baixa e sofrer interferências do corpo humano e do ambiente, é um sinal cuja captura exige uma série de cuidados especiais de forma a não ser comprometida. A utilização de circuitos de filtros passa-alta, passa-baixa e de *Notch* pode auxiliar fortemente no tratamento do SEMG antes de sua transmissão para meio digital. Aliado a esses filtros, a utilização de amplificadores operacionais possibilitam a amplificação do SEMG, além de funcionarem como mecanismo de *buffer* entre etapas de filtragem. Entretanto, tais circuitos de filtros utilizam resistores e capacitores, cujos valores poderão não ser precisos o suficiente para a construção de filtros com frequências de corte corretas. Além disso, o método artesanal de confecção de circuitos pode acrescentar ainda uma imprecisão no circuito final devido às capacitâncias parasitas que podem surgir entre as trilhas.

O SEMG pode ser capturado por um circuito amplificador para instrumentação, que pode ser construído utilizando-se 3 amp-ops e um conjunto de resistores, ou pode-se utilizar um CI que já implemente o amplificador para instrumentação internamente, aumentando assim a precisão. Um exemplo é o AD620, que é um CI de amplificador para instrumentação de alto desempenho.

Utilizando circuitos de filtros, amp-ops e o AD620, este trabalho atingiu seu objetivo específico de construção de um circuito de captura, filtragem e amplificação do SEMG. O ruído de 60 Hz ainda foi observado na saída do circuito, entretanto com uma amplitude bastante atenuada pelo filtro de *Notch*. Uma atenuação maior desse ruído não foi possível devido à alta taxa de tolerância (20%) dos capacitores utilizados, que impediu a construção de um filtro de *Notch* com um fator de qualidade maior. A digitalização do SEMG para o computador foi feita com sucesso por meio da utilização da placa de som como conversor analógico/digital.

A utilização de *Java* como linguagem de programação favoreceu a utilização de bibliotecas especializadas que ajudaram no desenvolvimento rápido da parte de *software* do protótipo construído. Essa escolha aumenta consideravelmente a produtividade da tarefa de programação, pois o *Java* é uma linguagem para a qual há várias bibliotecas prontas (geralmente gratuitas e bem testadas) em diversas áreas diferentes, favorecendo a reutilização por meio de uma componentização de módulos de *software*. A comunidade de desenvolvedores centrados na linguagem *Java* e suas bibliotecas também é extensa e ativa, o que facilita a resolução de problemas encontrados no uso dessas bibliotecas, já que a maioria dos problemas e suas soluções já foram relatados em fóruns de discussões pela *Internet*.

O uso da API de som do *Java* permitiu capturar o SEMG através da entrada de microfone

de um PC, ajudando atingir o objetivo específico da digitalização do SEMG. A escolha da entrada de som diminuiu consideravelmente o preço do protótipo, pois dispensou a utilização de *hardware* adicional para a realização dessa digitalização.

Para atingir o objetivo específico da visualização do SEMG no computador, utilizou-se a biblioteca de interface gráfica *Java Swing*, por meio da qual o SEMG foi renderizado na tela do computador, o que possibilita sua análise por um especialista da área médica. Como o *Java* é uma linguagem independente de plataforma, o mesmo *software* poderia ser executado em vários sistemas operacionais, como *Windows*, *Linux* e *MacOS*.

O *Blender 3D* permitiu a modelagem de uma prótese mioelétrica virtual de mão, além da construção de animações de seus movimentos. Assim, ele auxiliou o atingimento do objetivo específico de modelar uma mão virtual. Ele possui uma integração fácil com o *jMonkeyEngine*, que foi utilizado para a renderização do modelo 3D de mão no *Java*.

A FFT (por meio do algoritmo de Cooley-Tukey) mostrou-se uma técnica eficiente de calcular-se a Transformada Discreta de Fourier, pois diminui a complexidade computacional de $N(N-1)$ para $N(log_2 N)$. Ela transforma um sinal do domínio do tempo para o domínio da frequência, permitindo a extração de características do sinal no domínio da frequência.

A extração de características do SEMG reduziu o tamanho da entrada para a rede neural desenvolvida, permitindo um treinamento mais rápido dela. A rede neural do tipo percéptron de múltipla camada (*feedforward backpropagation*), utilizando a função de ativação sigmoid e 3 camadas, obteve uma taxa de erro de 17% para o reconhecimento de 4 movimentos da mão: flexão do punho, hiperextensão do punho, desvio radial e desvio ulnar. Dessa forma, o objetivo específico de controlar uma prótese virtual foi atingido parcialmente, já que 17% de erro não é uma taxa de erro desprezível.

Assim, o objetivo geral também foi atingido parcialmente, devido à taxa de erro de 17%. Entretanto, todas as etapas propostas foram concluídas, sendo essa taxa de erro e sua possível diminuição um outro problema maior, que poderá ser atacado em trabalhos futuros.

6.2 Sugestões para Trabalhos Futuros

Durante o desenvolvimento deste projeto, algumas possibilidades de trabalhos futuros surgiram. A seguir citam-se 3 sugestões.

a) No campo da captura do sinal eletromiográfico, melhorias podem ser realizadas. O desenvolvimento de um sistema de aquisição com mais canais tornaria possível a captura de mais movimentos diferentes da mão simultaneamente. Em particular, a utilização de uma malha de eletrodos que percorresse toda a circunferência do braço traria uma cobertura bastante completa dos músculos superficiais. Além disso, abordagens alternativas poderiam ser utilizadas nas etapas de transmissão para o meio digital, como por exemplo a utilização de pequenos eletrodos com transmissores *wireless*, que reduziria a

quantidade de artefatos causados pela movimentação dos eletrodos junto com os cabos do sistema de aquisição.

b) Utilizando-se os módulos de *software* de captura do sinal eletromiográfico desenvolvidos neste trabalho, seria possível a criação de um sistema de informações médicas voltado para o armazenamento de dados históricos de pacientes e sua evolução durante um determinado tratamento médico. O desenvolvimento desse sistema envolveria a utilização de banco de dados, além de métodos estatísticos que calculassem o desempenho de um paciente em um tratamento fisioterapêutico, por exemplo.

c) A construção de uma prótese mioelétrica de mão real seria uma outra possibilidade de trabalho futuro. Utilizando-se as técnicas de processamento digital de sinais abordadas neste trabalho, implementadas em um *hardware* apropriado, como um DSP, permitiria a implementação de uma prótese mecânica, que seria controlada pela rede neural, previamente treinada no computador pelo *software* desenvolvido neste trabalho.

Referências

ALEXANDER, Charles K.; SADIKU, Matthew N. O. *Fundamentos de Circuitos Elétricos*. [S.l.]: Bookman, 2000. 34, 35, 36

AMABIS, José Mariano; MARTHO, Gilberto Rodrigues. *Fundamentos da Biologia Moderna*. [S.l.]: Editora Moderna, 2002. 23

ANALOG DEVICES. *Datasheet do AD620*. 2011. 64

ANDRADE, Nei Augusto. Desenvolvimento de um sistema de aquisição e processamento de sinais eletromiográficos de superfície para a utilização no controle de próteses motoras ativas. 2007. Universidade de Brasília, 2007. 23, 24

BARROS, Kety Rosa de. Metodologia para classificação de sinais emg para controle de próteses com baixo esforço computacional. 2005. Universidade Federal de Uberlândia, 2005. 27

BASMAJIAN, JV; DELUCA, CJ. Muscles alive: their functions revealed by electromyography, 5th edn williams and wilkins. *Baltimore, MD*, 1985. p. 561, 1985. 20

BLENDER FOUNDATION. *Blender User Manual 2.6*. [S.l.], 2013. 54, 55

BOYLESTAD, Louis Nashelsky Robert. *Dispositivos Eletrônicos e Teoria de Circuitos*. [S.l.]: LTC, 1998. 31, 32, 33, 34, 35

BOYLESTAD, Robert L. *Dispositivos Eletrônicos e Teoria de Circuitos*. [S.l.]: Pearson, 2005. 34, 35

CAMARGO, Daniel Rodrigues de. Desenvolvimento do protótipo de uma prótese antropomórfica para membros superiores. 2008. USP, 2008. 21, 60

FORTI, Fabiana. Análise do sinal eletromiográfico em diferentes posicionamentos, tipos de eletrodos, ângulos articulares e intensidades de contração. 2005. Universidade Metodista de Piracicaba, 2005. 20

GRAUPE, D; CLINE, WK. Functional separation of emg signals via arma identification methods for prosthesis control purposes. *IEEE Trans Syst Man Cybernet*, 1975. n. 2, p. 252–258, 1975. 21

GUYON, Isabelle; ELISSEEFF, Andre. An introduction to feature extraction. 2006. Pascal2, 2006. 51

HALL, John Edward. *Tratado de Fisiologia Médica*. [S.l.]: Elsevier, 2011. 24, 25

HALL, Susan J. *Biomecânica Básica*. Third. [S.l.]: Guanabara Koogan S.A., 2000. 29, 30

HAYKIN, Barry Van Veen Simon. *Sinais e Sistemas*. [S.l.]: Bookman, 2001. 42, 43, 44

HAYKIN, Simon. *Redes Neurais*. [S.l.]: Bookman, 2001. 56

HEARN, Donald; BAKER, M. Pauline. *Computer Graphics*. [S.l.]: Prentice Hall, 1996. 53, 54

HEATON, Jeff. *Introduction to Neural Networks for Java*. Second. [S.l.]: Heaton Research, Inc., 2008. 57, 58, 59, 60

HEATON RESEARCH. *Encog Machine Learning Framework*. 2013. Disponível em: <http://www.heatonresearch.com/encog>. Acesso em: 30.5.2013. 83

HUDGINS, B; PARKER, P; SCOTT, RN. A new strategy for multifunction myoelectric control. *IEEE Trans Biomed Eng*, 1993. p. 40:82–94, 1993. 21

IBRAHIM, Zunaidi et al. Electromyography signal based for intelligent prosthesis design. In: SPRINGER. *4th Kuala Lumpur International Conference on Biomedical Engineering 2008*. [S.l.], 2008. p. 187–190. 20

KAMEN, Gary; GABRIEL, David A. *Essentials of electromyography*. [S.l.]: Human Kinetics 10%, 2010. 22, 24, 26, 41, 42

KENDALL. *Página de venda do Kendall Medi-Trace 200 Foam Electrode*. 2013. Disponível em: <http://www.cardiologyshop.com/kenmed230foa.html>. Acesso em: 21.5.2013. 62

KIDPORT. *Muscular System - Hand Muscles*. 2013. Disponível em: <http://www.kidport-.com/reflib/science/HumanBody/MuscularSystem/HandMuscles.htm>. Acesso em: 28.5.2013. 27

LUCA, Carlo. Electromyography. *Encyclopedia of Medical Devices and Instrumentation*, 2006. Wiley Online Library, 2006. 20

LUCA, Carlo J De. Surface electromyography: Detection and recording. *DelSys Incorporated*, 2002. v. 10, p. 2011, 2002. 26, 27, 28

MALTA, Juliana; CAMPOLONGO, Gabriel Denser; BARROS, Tarley Eloy Pessoa de; OLIVEIRA, Reginaldo Perilo de. Eletromiografia aplicada aos músculos da mastigação. 2006. 2006. 21

MASSó, Núria et al. Aplicaciones de la electromiografía de superficie en el deporte. *Apunts Medicina de L'Esport*, 2010. 2010. 21

MITRA, Sanjit K. *Digital Signal Processing*. [S.l.]: McGraw-Hill, 2006. 44, 45, 46, 47, 48, 49, 50

NAJMABADI, Farrokh. *ECE60L, Components & Circuits Laboratory*. [S.l.], 2004. Disponível em: <http://www-ferp.ucsd.edu/najmabadi/CLASS/ECE60L/04-S/NOTES/>. Acesso em: 20.5.2013. 36

NILSSON, James W.; RIEDEL, Susan A. *Circuitos Elétricos*. Fifth. [S.l.]: LTC, 1999. 36, 37, 39, 40, 103, 104

OKAWA, Electric Design. *Twin-T Notch Filter Design Tool*. 2013. Disponível em: <http://sim.okawa-denshi.jp/en/TwinTCRkeisan.htm>. Acesso em: 30.5.2013. 40, 41

O'SULLIVAN, Susan B.; SCHMITZ, Thomas J. *Fisioterapia - Avaliação e Tratamento*. [S.l.: s.n.], 2010. 26

PACKTER, Lúcio. *Neurociência: Elementos de Neurofisiologia, Farmacologia e Psiquiatria*. [S.l.], 2011. Disponível em: <http://www.filosofiaadistancia.com.br-/grava%C3%A7%C3%B5es%20agosto/Neuroci%C3%AAncias/Neuroci%C3%AAncia%20-%20apostila%20I.htm>. Acesso em: 21.5.2013. 23

PHINYOMARK, Angkoon; LIMSAKUL, Chusak; PHUKPATTARANONT, Pornchai. A novel feature extraction for robust emg pattern recognition. 2009. Journal of Computing, 2009. 51

PICCOLINO, Marco. Luigi galvani and animal electricity: two centuries after the foundation of electrophysiology. *Trends in neurosciences*, 1997. Elsevier, v. 20, n. 10, p. 443–448, 1997. 20

PRENTICE, William E. *Modalidades Terapêuticas para Fisioterapeutas*. [S.l.]: Artmed Editora - Grupo A, 2008. 17

SARIC, Marin. *LibHand: A Library for Hand Articulation*. 2011. Version 0.9. Disponível em: <http://www.libhand.org/>. Acesso em: 20.3.2013. 66

SEDGEWICK, Robert; WAYNE, Kevin. *Introduction to Programming in Java*. 2012. Disponível em: <http://introcs.cs.princeton.edu/java/97data/FFT.java.html>. Acesso em: 29.5.2013. 81

SORIANO, Marc. *Lab 5 - Skeletal Animation*. [S.l.], 2009. Disponível em: <http://alumni.cs-.ucr.edu/~sorianom/cs134_09win/lab5.htm>. Acesso em: 25.5.2013. 54

TEIXERA, Ingred Carvalho; GODOI, Tomás da Silva Martins de; LIMONGE, Samantha Coimbra; ASSIS, Matheus Sant'Anna de; BISPO, Vinícius Araújo. Proposta de gamificação dos sinais mioelétricos aplicados na reabilitação fisioterapêutica em pacientes com dificuldades em executar movimentos em membros superiores ou inferiores. 2012. Centro Universitário de Brasília - UniCEUB, 2012. 62, 63, 65, 66

TEXAS INSTRUMENTS. *Datasheet do TLC274CN*. 1998. 63

VASCONCELLOS, Henrique Augusto Silva. Interpretação de sinais emg para joelho de prótese robótica. 2008. UnB, 2008. 60

WECKER, Jonas Edison. *Aula de Anatomia - Sistema Muscular*. 2013. Disponível em: <http://www.auladeanatomia.com/sistemamuscular/mao.htm>. Acesso em: 27.5.2013. 28

ZECCA, M; MICERA, Silvestro; CARROZZA, MC; DARIO, P et al. Control of multifunctional prosthetic hands by processing the electromyographic signal. *Critical reviews in biomedical engineering*, 2002. Boca Raton, Fla.: CRC Press, 1981-, v. 30, n. 4-6, p. 459, 2002. 21, 22

Apêndices

APÊNDICE A – A transformada de Laplace e análise de circuitos

A transformada de Laplace de uma função é definida pela Equação A.1.

$$\mathcal{L}\{f(t)\} = \int_0^\infty f(t)e^{-st}\,\mathrm{d}t \tag{A.1}$$

A transformada de Laplace de uma função $f(t)$ também pode ser representada como $F(s)$, de acordo com a Equação A.2.

$$F(s) = \mathcal{L}\{f(t)\} \tag{A.2}$$

A Tabela A.1 mostra os pares de transformadas de Laplace mais comuns.

Tabela A.1 – Pares de transformadas de Laplace

$f(t)$	$F(s)$
Impulso unitário $\delta(t)$	1
Degrau unitário $u(t)$	$\frac{1}{s}$
t	$\frac{1}{s^2}$
$\frac{t^{n-1}}{(n-1)!}$	$\frac{1}{s^n}$
t^n	$\frac{n!}{s^{n+1}}$
e^{-at}	$\frac{1}{s+a}$

A Figura A.1 mostra o circuito equivalente de um resistor no domínio da frequência.

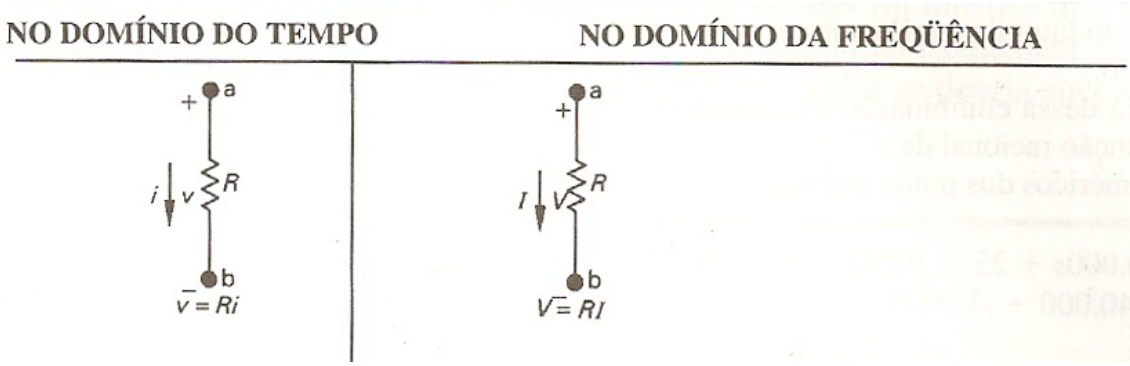


Figura A.1 – Circuito de um resistor equivalente no domínio da frequência

Fonte: Nilsson e Riedel (1999)

A Figura A.2 mostra o circuito equivalente de um capacitor no domínio da frequência.

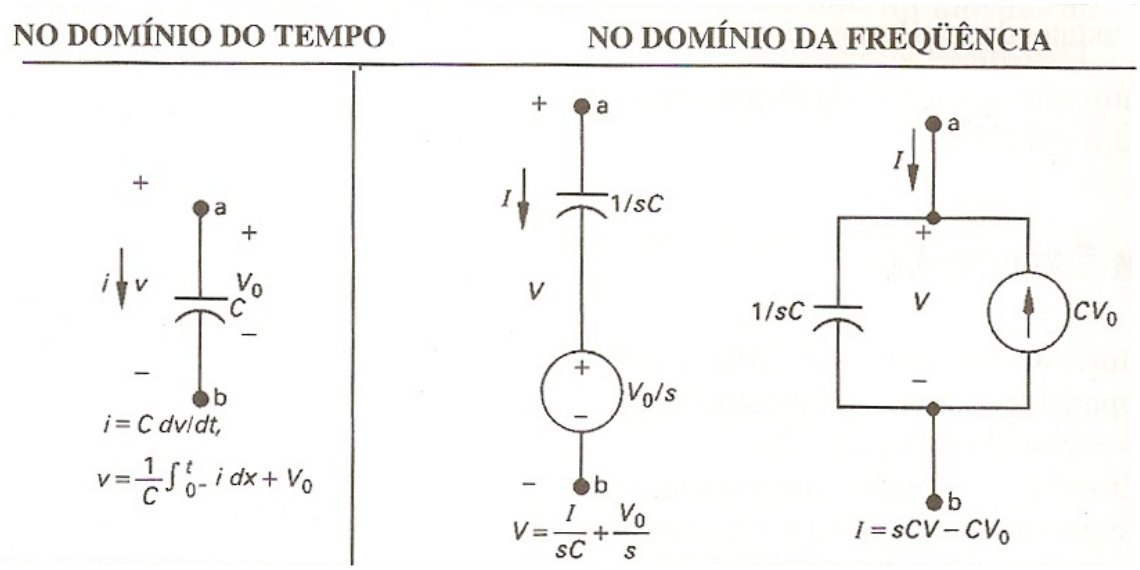


Figura A.2 – Circuito de um capacitor equivalente no domínio da frequência

Fonte: Nilsson e Riedel (1999)

APÊNDICE B – Artigo publicado no 12° Congresso Nacional de Iniciação Científica (CONIC-SEMESP)

Este apêndice apresenta o artigo publicado no 12° CONIC-SEMESPE (2012), de coautoria do autor deste trabalho, resultado da pesquisa realizada em Projeto de Iniciação Científica do UniCEUB.

PROPOSTA DE GAMIFICAÇÃO DOS SINAIS MIOELÉTRICOS APLICADOS NA REABILITAÇÃO FISEOTERAPÊUTICA EM PACIENTES COM DIFICULDADES EM EXECUTAR MOVIMENTOS EM MEMBROS SUPERIORES OU INFERIORES.

Ingred Carvalho Texeira[1], Tomás da Silva Martins de Godoi[1], Samantha Coimbra Limonge[1], Matheus Sant'Anna de Assis[1], Vinícius Araújo Bispo[1]

Orientador: Msc. Prof. Luciano Henrique Duque[1]

[1]Faculdade de Tecnologia e Ciências Sociais Aplicadas (FATECS)
Centro Universitário de Brasília (UniCEUB) – Brasília, DF – Brasil

Resumo

Esse trabalho apresenta um sistema de condicionamento para a coleta e análise de sinais Eletromiográficos (EMG), através de medições dos sinais nervosos, que surgem com a produção de esforços musculares e que são quantizados através de ***games*** em fases, conforme o exercício recomentado pelo fisioterapeuta. A gamificação dos sinais mioelétricos visa melhorar o desempenho na reabilitação de membros superiores ou inferiores, oferecendo um feedback motivacional ao paciente, através de desafios moldados em etapas. O sistema proposto é composto por um hardware, o amplificador de eletromiografia (EMG), um software analisador de sinais e o desenvolvimento de vários ***games*** conforme as necessidades do paciente. As medições dos sinais mioelétricos e sua tratativa são executadas pelo software analisador, peça fundamental para o funcionamento adequado dos ***games*** conforme os exercícios atribuídos pelo profissional de fisioterapia.

Introdução

Eletromiografia é uma técnica que se propõe a estudar os fenômenos bioelétricos que ocorrem no corpo humano dentro das membranas celulares na musculatura humana e em todo o sistema nervoso. O espectro de frequências dos sinais mioelétricos estendem-se de 10Hz a 500Hz para a maioria dos eletrodos de superfície, para fins cinesiológicos que é o alvo desse trabalho. No cérebro humano, há uma grande quantidade de atividade neural que nos define como somos e o que fazemos. O amplificador de Eletromiografia (EMG) é utilizado mediante o uso de instrumentos eletrônicos que disponibilizam

informações relacionadas à resposta captada por um determinado esforço muscular. (Adaptado de: ANDRADE, 2007).

Nesse contexto, torna-se importante obter o sinal elétrico produzido pelo esforço muscular, utilizando um amplificador de eletromiografia (EMG), que amplifica o sinal captado e um software processa os sinais e apresenta em tela sua amplitude e sons gerados pela atividade muscular. Após o sinal captado, amplificado e analisado, um software com o sinal captado e o paciente efetua os exercícios necessários a sua reabilitação, recebendo um feedback através do ***game*** conforme a recomendação do terapeuta, que acompanha e avalia o sinal captado.O EMG é a base para captar os sinais, sendo bastante útil em procedimentos de fisioterapia e outros campos de pesquisa, e consegue obter informações sobre a atividade elétrica do músculo que é induzido, assim podemos efetuar o processamento e a gamificação dos sinais (Adaptado de: AMADIO, 2006).

Antes do surgimento do EMG (amplificador de eletromiografia), clínicos confiavam apenas em seu “tato” ou na inspeção visual para diagnosticar os músculos que estavam sendo analisados durante a excitação. Com o amplificador de Eletromiografia, software de processamento e o ***game***, teremos mais objetividade e exatidão no diagnóstico e auxílio na recuperação de pacientes que apresentem lesões em membros superiores ou inferiores. Nesse cenário, torna-se mais fácil os clínicos e terapeutas obterem um resultado melhor da situação do músculo analisado, e assim auxiliar o paciente em sua recuperação (Adaptado de: LUCA, 2007).

O EMG proposto (hardware) para captura dos sinais será desenvolvido com amplificador operacional diferencial e filtros baseado na família TLC274CN e os softwares para processamento de sinais e gamificação são desenvolvidos em linguagem de programação ***java***. Os ***games*** desenvolvidos são elaborados para exercícios de reabilitação que envolve força e controle dos membros superiores ou inferiores. Vamos aplicar nesse trabalho o conceito de gamificação, que consiste em usar estrutura e dinâmica encontrada em jogos para motivar e aprimorar os pacientes em reabilitação fisioterapêutica em membros superiores ou inferiores (Adaptado de: KARL, 2012).

Objetivos

O objetivo geral proposto nesse trabalho é desenvolver um protótipo de um sistema capaz de extrair os sinais elétricos do músculo, analisá-los, e através de um processo de gamificação motivar e auxiliar a reabilitação de pacientes com problemas em membros superiores ou inferiores, com auxílio de profissional de fisioterapia.

Os objetivos específicos para desenvolver o sistema de coleta, análise e gamificação dos sinais mioelétricos são:

- Projetar filtros para eliminar ruídos indesejáveis, permitindo captar apenas o ruído muscular na faixa 10Hz a 500Hz;
- Elaborar circuito amplificador diferencial com o foco na amplificação dos sinais elétricos (10Hz a 500Hz) gerados pelo músculo, utilizando amplificadores operacionais de alta sensibilidade;
- Desenvolver utilizando a linguagem ***java*** um software capaz de analisar processar o sinais;
- Desenvolver ***games*** adequados para cada tipo exercício necessários na reabilitação do paciente.

Metodologia

Coletar, analisar e processar os sinais mioelétricos é de suma importância para o desenvolvimento do trabalho proposto, pois após as tratativas de sinais, o conceito da gamificação é aplicado, permitindo ao paciente em recuperação fisioterapêutica efetuar exercícios com auxílio de ***games*** que estimulem sua recuperação. Nesse cenário, como metodologia para desenvolvimento do trabalho proposto, o sistema é dividido nas seguintes etapas:

- A **primeira etapa** é formada pelo estudo e captação dos sinais mioelétricos do músculo analisado, com utilização de eletrodos de superfície. Os eletrodos são dispositivos de entrada e saída de corrente em um sistema elétrico.
- A **segunda etapa** é embasada no desenvolvimento de um circuito condicionador, que é constituído por filtros e amplificadores operacionais diferencias, responsáveis pela filtragem e amplificação do sinal de atividade muscular. Nessa fase, os sinais coletados do

músculo em análise serão filtrados na faixa de 10Hz a 500Hz e após esse processo serão amplificados como base em amplificadores operacionais de alta sensibilidade, especificamente o operacional da família TLC274CN;

- A **terceira etapa** consiste no desenvolvimento de um software baseado em linguagem ***java***, capaz de receber os sinais coletados pelo EMG, processar matematicamente, apresentar sua forma de onda na tela do computador e apresentar de forma audível o ruído gerado pelo músculo em análise. O software prevê também mecanismos que permite mensurar a intensidade da força exercida pelo músculo e possibilidade de eliminar ruído fora da faixa de 10Hz a 500Hz do sinal EMG;
- Na **quarta etapa** o conceito da gamificação é aplicado para o desenvolvimento de jogos (games) que vão interagir com os sinais coletados pelo EMG e processados pelo software analisador de sinais. Por enquanto, estamos usando o Unity, que permite o desenvolvimento dos scripts dos jogos em linguagem C, ***Javascript***. e para o protótipo inicial apresentamos dois jogos que são referentes a exercícios de força recomendados por fisioterapeuta;
- Por fim, a **quinta etapa** consiste em realizar testes no hardware do EMG e testes de interação dos ***games*** com o EMG e validação dos testes. Complementando a metodologia utilizamos o simulador de circuitos Proteus (PROTEUS,2010) , multímetro digital, Eletrodos de superfície e computador para auxiliar os testes.

Desenvolvimento

Para desenvolver o sistema proposto é necessário analisar como os sinais eletromiográficos no corpo são transmitidos e a partir desse conhecimento desenvolver o protótipo para captação e amplificação das atividades musculares. Após o hardware EMG desenvolvido é possível de projetar os softwares para analise e geração de ***games*** de interação com o paciente. Os formados de onda EMG são processados por um amplificador diferencial com base no amplificador operacional TLC274CN de alta sensibilidade, impedância e ganho.

Sinais Eletromiográficos

São sinais mioelétricos gerados pelas contrações de nervos e músculos. Esses sinais apresentam tensões em níveis muito baixos, tipicamente variando entre 100 μv a 2 mv, com alta impedância e propensos a altos níveis de interferência de sinal e ruído (RICCIOTTI, 2006). O sinal EMG (Eletromiográfico) pode ser definido pela equação (1) (ANDRADE, 2007):

$$EMG(t) = \sum_{j=1}^{Nm} SPAUM(t) + n(t) \qquad \text{Eq.(1)}$$

Onde: **SPAUM** – série de potenciais de ação da unidade motora de 10Hz a 500Hz; **n(t)** – Ruído, **t** - é o tempo de amostra.

O software que é desenvolvido para processar o sinal EMG da equação 1 realiza uma operação matemática com o sinal, que é a transformada de ***Wavelet.*** A transformada de wavelet decompõe uma função definida no domínio do tempo em outra função, definida no domínio do tempo e no domínio da frequência. Ela é indicada para processamento de sinais mioelétricos e que não entraremos em detalhes nesse trabalho e apresentaremos o software com essa funcionalidade embutida.

Desenvolvimento do Circuito Condicionar de Sinais Mioelétricos

Os sinais Eletromiográficos são captados através dos eletrodos de superfície, que são dispositivos transdutores de sinais. Os sinais captados pelos eletrodos de superfície precisam ser amplificados devido a sua amplitude ser muito baixa. Os ruídos provenientes dos batimentos cardíacos, aparelhos eletrônicos podem provocar interferências no circuito condicionador. A figura 1 apresenta um eletrodo de superfície que utilizamos na captura dos sinais.

Figura 1: Eletrodo para captação do sinal EMG

Nesse cenário, é necessário desenvolver filtros de sinais na faixa de 10Hz e 500 Hz em conjunto com amplificadores diferenciais e amplificadores de áudio, esse nos permite ouvir a atividade muscular. A figura 2 abaixo apresenta o diagrama em blocos do circuito condicionador com amplificador de áudio e filtros.

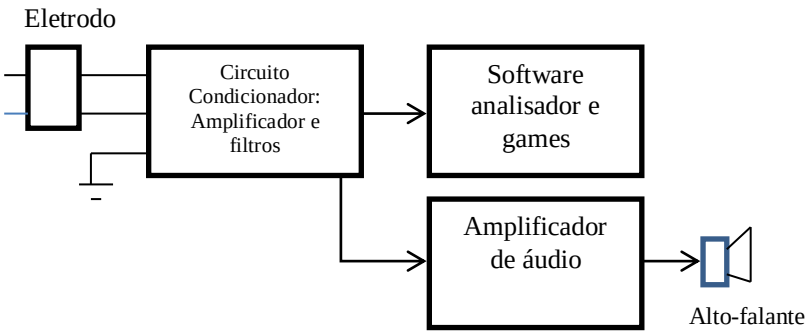


Figura 2: Diagrama em blocos do sistema proposto

O sinal captado pelo eletrodo é aplicado em um circuito amplificador diferencial com filtros na faixa de 10Hz a 500Hz, faixa dos sinais EMG (SUSAN,2008).Após amplificados, são enviados para o software analisador e amplificador de áudio. O software analisador é responsável por exibir na tela do computador a forma de onda do sinal EMG capturado no músculo em estudo e também apresenta funções de filtragem, mediações de intensidade força e tela personalizada que possibilita ao profissional de fisioterapia fazer diagnósticos em membros superiores ou inferiores. A tela do software para geração do ***game*** é apresentado apenas para o paciente em outro computador, no entanto, quem seleciona o ***game*** é o terapeuta e sua escolha é função do exercício. O circuito condicionador é projetado baseado no amplificador operacional TLC274CN, que possui alta impedância e ganho. A filtragem do sinal ocorre na faixa de 10Hz a 500Hz. O amplificador diferencial (TLC274CN) com os filtros RC na entrada que tem a função de eliminar os ruídos de atividades provindas dos outros músculos, os sinais cardíacos e outros tipos de interferências eletromagnéticas deixando passar apenas os sinais de EMG do músculo analisado. O filtro de RC pode ser obtido através da equação 2:

$$f = \frac{1}{2\pi * R * C} \qquad \text{(Eq.2)}$$

Onde: f - é a frequência de corte, R - Resistência em homs (Ω), Capacitância em Faraday

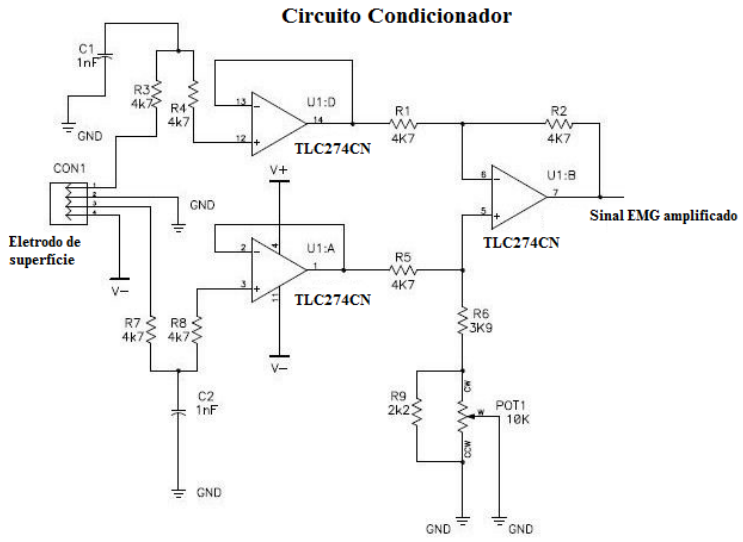


Figura 3: Circuito condicionador

(adaptado: TLC 274, 2012)

A figura 4 apresenta o projeto do amplificador de áudio, cuja função é amplificar o ruído da atividade muscular. O projeto desse módulo é utilizado um chip TDA7052B, onde as funções de amplificação são integradas. Os capacitores e resistores externos são indicados pelo fabricante para filtragem de ruídos externos (DATASHEET TDA7052, 2012).

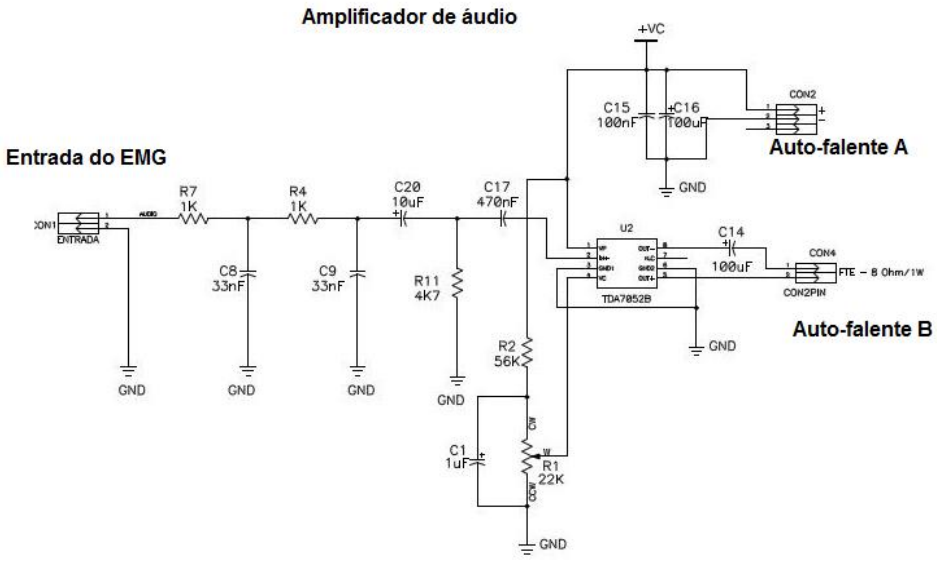


Figura 4: Amplificador de áudio

Fonte: (Adaptado de: DATASHEET TDA7052,2010; PERTENCE,2006).

Projeto do software Analisador e Games

O software analisador é desenvolvido em linguagem ***java*** e apresenta a função de processador de sinais, que são enviados pelo circuito condicionador e capturados pelo software. Ele apresenta os resultados em uma

tela de computador, com funções de medição de força, gravação dos sinais, reprodução, filtragem dos sinais e o processamento é feito via transformada de ***Wavelet***. A figura 5 abaixo apresenta uma tela do software processando um determinado sinal EMG capturado pelo circuito condicionador em um membro superior.

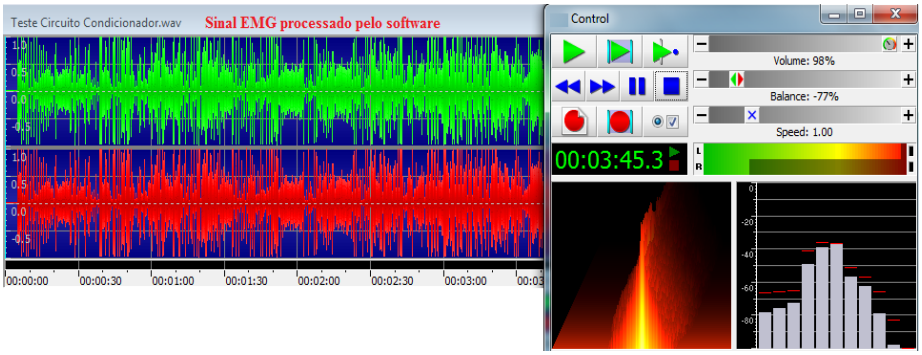


Figura 5: Tela de um sinal EMG processado pelo software desenvolvido

O sinal EMG capturado e processado pelo software é enviado para o game, que conforme o movimento do paciente opera o game especificado pelo terapeuta. Os games são desenvolvidos na plataforma Unity, utilizando o suporte nativo a captura de sinal de microfone para capturar o sinal mioelétrico amplificado. O sinal então é utilizado como controlador do jogo e o sinai EMG capturado pelo circuito condicionador e analisado pelo terapeuta. A figura 6 representa o menu dos jogos desenvolvidos que o terapeuta pode indicar aos pacientes.

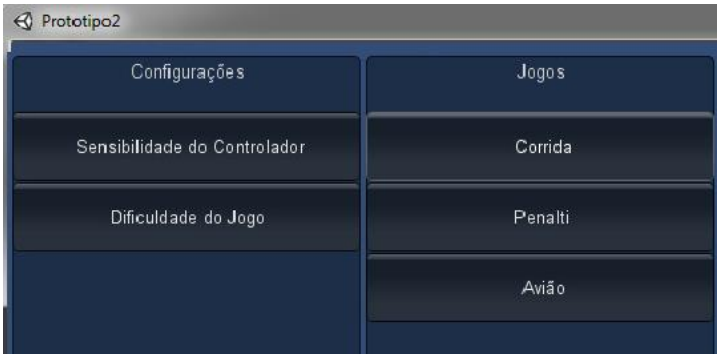


Figura 6: Tela do software para escolha de jogos

No primeiro game, de corrida, o esforço do paciente é traduzido na velocidade do personagem, que no primeiro protótipo é representado por uma bola branca competindo com uma vermelha que é acelerada por uma força constante. A figura 7 apresenta a tela do jogo de corrida.

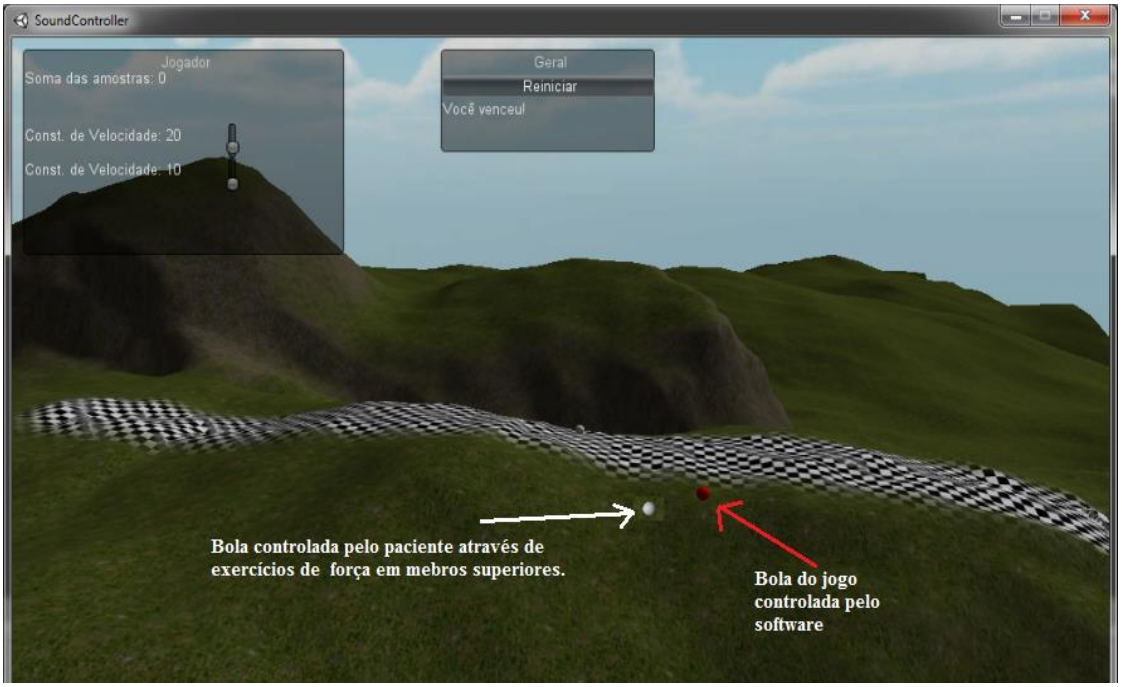


Figura 7: Jogo de corrida para exercícios de força em membros em recuperação

A dificuldade do jogo é definida pela distância e relevo do terreno do cenário escolhido e por parâmetros configurados pelo terapeuta, como a sensibilidade da captura, limiar de ativação do controlador e velocidade do oponente. No segundo jogo, que simula uma cobrança de pênalti em um jogo de futebol, o esforço do paciente define a força do chute e deve ser concentrado em um tempo limitado. Os parâmetros controlados pelo terapeuta incluem a velocidade de movimento do goleiro, que no protótipo é representado por uma parede, o tempo disponível para o paciente encher a barra de força do chute, a quantidade de cobranças de pênalti realizadas, a sensibilidade da captura e limiar de ativação do controlador.

Resultados

O protótipo apresenta resultados satisfatórios, no entanto, ainda encontra-se fase de ajustes finais e desenvolvimento. O sistema já é capaz de captar o sinal mioelétrico, processar via software, interagir com o ***game*** de corrida. Testes em membros superiores foram realizados e obtivemos boa resposta no quesito qualidade do sinal, processamento e interação com o game. A figura 8 apresenta o protótipo pronto e em teste em um membro superior (braço) e com sua interação com o ***game***.

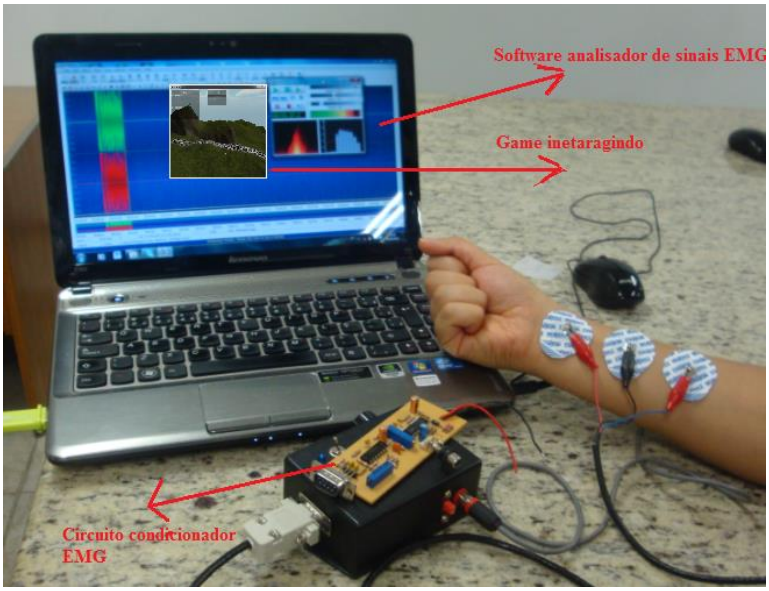


Figura 8: Protótipo do sistema em teste de membro superior

Considerações Finais

Com a aquisição do sinal mioelétrico, que é amplificado, filtrado e processado pelo software, a resposta medida atende as expectativas esperadas e também permitiu boa interação com o ***game***. O fisioterapeuta pode com o auxílio do protótipo, estudar e analisar o comportamento de uma musculatura qualquer em pacientes em reabilitação ou não. Por outro lado, o paciente obtém um feedback de sua recuperação através do ***game*** motivando sua reabilitação. A obtenção do sinal mioelétrico dependente muito de cada paciente, pois alguns possuem sinais Eletromiográficos (EMG) mais forte que outros em seu organismo. No entanto, o circuito condicionador projetado apresenta ganho ajustável, compensando as perdas que ocorrem de pessoa para pessoa.

Fontes Consultadas

AMADIO, A. C; ARAÚJO, R. C. **Análise biomecânica da ativação das porções superficiais do músculo**. Revista Brasileira de Fisioterapia, 2006.

ANDRADE, Nei Augusto., "**Desenvolvimento de um sistema de aquisição e processamento de sinais eletromiográficos de superfície para a utilização no controle de próteses motoras ativas",** Dissertação de Mestrado, UnB - Universidade de Brasília, 2007.

De Luca, C. J. **Surface electromyography: detection and recording**. 2006.

DATASHEET TDA7052, **Manual Técnico da NXP Semicondutores**, Disponívelem:<http://www.nxp.com/documents/data_sheet/TDA7052B.pdf**>.**Acesso em:Abr. 2010.

DATASHEET LM324, **Manual Técnico da National**, Disponível em: <http://www.national.com/ds/LM/LM124.pdf>. Acesso em : Jun. 2010.

Karl M. Kapp, Ed.D, The Gamification of Learning and Instruction, Editor Pfeiffer, 2012.

PERTENCE, ANTONIO. JR, **Eletrônica Analógica – Amplificadores Operacionais e Filtros Ativos**, 6ª Edição. São Paulo: Bookman, 2003.

RICCIOTTI, Duarte; **Utilização de Wavelets no processamento de sinais EMG**, 2006.

SUSAN, B.O, Thomas J. Schmitz, Fiseoterapia: Avaliação e tratamente, Ed.5, 2008.

PROTEUS - ***Labcenter Electronics*** – Professional PCB Design and Simulation. Disponível em < http://www.labcenter.com/>. Acesso em: Agosto 2012.

APÊNDICE C – Código-Fonte

```
1  package br.com.tomas.emgprocessing.rna;
2
3  import br.com.tomas.emgprocessing.sinal.mat.Complex;
4  import br.com.tomas.emgprocessing.sinal.mat.FFT;
5
6  public class ExtratorCaracteristicas {
7    private boolean normalizar;
8    private double inicio;
9    private double fim;
10
11   public ExtratorCaracteristicas(double inicio, double fim) {
12     this.normalizar = true;
13     this.inicio = inicio;
14     this.fim = fim;
15   }
16
17   public ExtratorCaracteristicas(boolean normalizar) {
18     this.normalizar = normalizar;
19   }
20
21   public ExtratorCaracteristicas() {
22     this.normalizar = false;
23   }
24
25   public double calcularIEMG(Integer[] amostras, int tamanho) {
26     double iemg = 0.0;
27
28     for (int i = 0; i < tamanho; i++) {
29       iemg += Math.abs(amostras[i]);
30     }
31
32     if (normalizar) {
```

```
33      iemg = normalizar(iemg, 0.0, tamanho * Math.pow(2, 14));
34    }
35
36    System.out.println("IEMG: " + iemg);
37    return iemg;
38  }
39
40  public double normalizar(double valor, double inicio, double
       fim) {
41    return (valor - inicio) / (fim - inicio);
42  }
43
44  public double calcularMAV(Integer[] amostras, int tamanho) {
45    double mav = 0.0;
46
47    for (int i = 0; i < tamanho; i++) {
48      mav += Math.abs(amostras[i]);
49    }
50
51    mav /= tamanho;
52
53    if (normalizar) {
54      mav = normalizar(mav, 0.0, 10000.0);
55    }
56
57    System.out.println("MAV: " + mav);
58    return mav;
59  }
60
61  public double calcularMNF(Integer[] amostras, int tamanho,
       int taxaAmostragem) {
62    Complex[] comps = new Complex[tamanho];
63    for (int i = 0; i < tamanho; i++) {
64      comps[i] = new Complex(amostras[i], 0);
65    }
66    comps = FFT.fft(comps);
67
68    double intervaloFrequencia = ((double)taxaAmostragem) /
        tamanho;
```

```
69
70    double mnf = 0.0;
71    double denominador = 0.0;
72
73    int meiaDft = tamanho / 2;
74    for (int i = 0; i < meiaDft / 2; i++) {
75      mnf += i * intervaloFrequencia * comps[i].abs();
76      denominador += comps[i].abs();
77    }
78
79    mnf /= denominador;
80
81    if (normalizar) {
82      mnf = normalizar(mnf, 0.0, 5000);
83    }
84
85    System.out.println("MNF:␣" + mnf);
86    return mnf;
87  }
88
89  public double calcularMDF(Integer[] amostras, int tamanho) {
90    Complex[] comps = new Complex[tamanho];
91    for (int i = 0; i < tamanho; i++) {
92      comps[i] = new Complex(amostras[i], 0);
93    }
94    comps = FFT.fft(comps);
95
96    double ipt = 0.0;
97
98    int meiaDft = tamanho / 2;
99    for (int i = 0; i < meiaDft; i++) {
100     ipt += comps[i].abs();
101   }
102
103   double meioIpt = ipt / 2;
104
105   double mdf = 0.0;
106   ipt = 0.0;
107   for (int i = 0; i < meiaDft; i++) {
```

```
108       ipt += comps[i].abs();
109       if (ipt > meioIpt) {
110         mdf = i;
111         break;
112       }
113     }
114
115     if (normalizar) {
116       mdf = normalizar(mdf, 0.0, 4000.0);
117     }
118
119     System.out.println("MDF:␣" + mdf);
120     return mdf;
121   }
122 }
```

```
1  package br.com.tomas.emgprocessing.rna;
2
3  import java.util.ArrayList;
4  import java.util.List;
5  import java.util.Map;
6
7  import org.encog.engine.network.activation.ActivationSigmoid;
8  import org.encog.ml.data.MLData;
9  import org.encog.ml.data.MLDataPair;
10 import org.encog.ml.data.MLDataSet;
11 import org.encog.ml.data.basic.BasicMLDataSet;
12 import org.encog.neural.networks.BasicNetwork;
13 import org.encog.neural.networks.layers.BasicLayer;
14 import org.encog.neural.networks.training.propagation.back.
     Backpropagation;
15
16 import br.com.tomas.emgprocessing.g3d.Comando;
17
18 public class RedeNeuralSEMG {
19
20   private BasicNetwork network;
21   private Map<Integer, Comando> mapeamento;
22
```

```
23  private List<AmostraRedeNeural> amostras = new ArrayList<
       AmostraRedeNeural>();
24  private double iemgMax;
25  private double iemgMin;
26  private double mavMax;
27  private double mavMin;
28  private double mnfMin;
29  private double mnfMax;
30  private double mdfMin;
31  private double mdfMax;
32
33  public RedeNeuralSEMG() {
34
35  }
36
37  public void adicionarAmostraRedeNeural(AmostraRedeNeural arn)
        {
38    amostras.add(arn);
39  }
40
41  public void treinar() {
42    network = new BasicNetwork();
43    network.addLayer(new BasicLayer(null, true, 4));
44    network.addLayer(new BasicLayer(new ActivationSigmoid(),
        true, 7));
45    //network.addLayer(new BasicLayer(new ActivationSigmoid(),
        false, 4));
46    network.addLayer(new BasicLayer(new ActivationSigmoid(),
        false, mapeamento.size()));
47    network.getStructure().finalizeStructure();
48    network.reset();
49
50    MLDataSet trainingSet = new BasicMLDataSet(getEntrada(),
        getSaida());
51
52    final Backpropagation train = new Backpropagation(network,
        trainingSet);
53
54    int epoch = 1;
```

```
55
56    do {
57      train.iteration();
58      System.out
59          .println("Epoch␣#" + epoch + "␣Error:" + train.getError
              ());
60      epoch++;
61    } while ((epoch < 5000) && (train.getError() > 0.001));
62
63  }
64
65  private double[][] getSaida() {
66    double[][] saidas = new double[amostras.size()][mapeamento.
        size()];
67
68    int saida;
69    for (int i = 0; i < saidas.length; i++) {
70      saida = amostras.get(i).getSaidaEsperada();
71      for (int j = 0; j < mapeamento.size(); j++) {
72        if (j == saida) {
73          saidas[i][j] = 1.0;
74        } else {
75          saidas[i][j] = 0.0;
76        }
77      }
78    }
79
80    return saidas;
81  }
82
83  private double[][] getEntrada() {
84    double[][] entradas = new double[amostras.size()][4];
85    ExtratorCaracteristicas ec = new ExtratorCaracteristicas
        (0.0, 1.0);
86
87    Integer[] seq = null;
88
89    ExtratorCaracteristicas ecSemNormalizar = new
        ExtratorCaracteristicas(false);
```

```
90    double iemgMax = Integer.MIN_VALUE, iemgMin = Integer.
        MAX_VALUE, iemg;
91    double mavMax = Integer.MIN_VALUE, mavMin = Integer.
        MAX_VALUE, mav;
92    double mnfMax = Integer.MIN_VALUE, mnfMin = Integer.
        MAX_VALUE, mnf;
93    double mdfMax = Integer.MIN_VALUE, mdfMin = Integer.
        MAX_VALUE, mdf;
94    for (int i = 0; i < entradas.length; i++) {
95      seq = amostras.get(i).getSeq();
96      iemg = ecSemNormalizar.calcularIEMG(seq, seq.length);
97      entradas[i][0] = iemg;
98      System.out.println("iEMG:" + iemg);
99      if (iemg > iemgMax) {
100       iemgMax = iemg;
101     }
102     if (iemg < iemgMin) {
103       iemgMin = iemg;
104     }
105
106     mav = ecSemNormalizar.calcularMAV(seq, seq.length);
107     entradas[i][1] = mav;
108     System.out.println("mAV:" + mav);
109     if (mav > mavMax) {
110       mavMax = mav;
111     }
112     if (mav < mavMin) {
113       mavMin = mav;
114     }
115
116     int pot2proxima = (int) Math.pow(2, Math.floor(Math.log(
          seq.length) / Math.log(2)));
117     mnf = ecSemNormalizar.calcularMNF(seq, pot2proxima, 8000);
118     entradas[i][2] = mnf;
119     System.out.println("mnf:" + mnf);
120     if (mnf > mnfMax) {
121       mnfMax = mnf;
122     }
123     if (mnf < mnfMin) {
```

```
124       mnfMin = mnf;
125      }
126
127      mdf = ecSemNormalizar.calcularMDF(seq, pot2proxima);
128      entradas[i][3] = mdf;
129      System.out.println("mdf:" + mdf);
130      if (mdf > mdfMax) {
131       mdfMax = mdf;
132      }
133      if (mdf < mdfMin) {
134       mdfMin = mdf;
135      }
136    }
137    System.out.println("iEMGmAX:" + iemgMax);
138    System.out.println("iEMGmIN:" + iemgMin);
139    System.out.println("mAVmAX:" + mavMax);
140    System.out.println("mAVmIN:" + mavMin);
141
142    System.out.println("mNFmAX:" + mnfMax);
143    System.out.println("mNFmIN:" + mnfMin);
144    System.out.println("mDFmAX:" + mdfMax);
145    System.out.println("mDFmIN:" + mdfMin);
146
147    this.iemgMax = iemgMax;
148    this.iemgMin = iemgMin;
149    this.mavMax = mavMax;
150    this.mavMin = mavMin;
151    this.mnfMin = mnfMin;
152    this.mnfMax = mnfMax;
153    this.mdfMin = mdfMin;
154    this.mdfMax = mdfMax;
155
156
157    for (int i = 0; i < entradas.length; i++) {
158      entradas[i][0] = ecSemNormalizar.normalizar(entradas[i
          ][0], iemgMin, iemgMax);
159      entradas[i][1] = ecSemNormalizar.normalizar(entradas[i
          ][1], mavMin, mavMax);
160
```

```
161     entradas[i][2] = ecSemNormalizar.normalizar(entradas[i
          ][2], mnfMin, mnfMax);
162     entradas[i][3] = ecSemNormalizar.normalizar(entradas[i
          ][3], mdfMin, mdfMax);
163   }
164
165   /*for (int i = 0; i < entradas.length; i++) {
166     seq = amostras.get(i).getSeq();
167     entradas[i][0] = ec.calcularIEMG(seq, seq.length);
168     entradas[i][1] = ec.calcularMAV(seq, seq.length);
169
170     int pot2proxima = (int) Math.pow(2, Math.floor(Math.log(
          seq.length) / Math.log(2)));
171     entradas[i][2] = ec.calcularMNF(seq, pot2proxima, 8000);
172     entradas[i][3] = ec.calcularMDF(seq, pot2proxima);
173   }*/
174
175   return entradas;
176 }
177
178 public Comando executar(Integer[] amostras, int saidaEsperada
      ) {
179   ExtratorCaracteristicas ec = new ExtratorCaracteristicas
        (0.0, 1.0);
180   ExtratorCaracteristicas ecSN = new ExtratorCaracteristicas(
        false);
181   double[][] entrada = new double[1][4];
182   double[][] saida = new double[1][4];
183
184   entrada[0][0] = ecSN.normalizar(ecSN.calcularIEMG(amostras,
        amostras.length), this.iemgMin, this.iemgMax);
185   entrada[0][1] = ecSN.normalizar(ecSN.calcularMAV(amostras,
        amostras.length), this.mavMin, this.mavMax);
186
187   int pot2proxima = (int) Math.pow(2, Math.floor(Math.log(
        amostras.length) / Math.log(2)));
188   entrada[0][2] = ecSN.normalizar(ecSN.calcularMNF(amostras,
        pot2proxima, 8000), this.mnfMin, this.mnfMax);
```

```
189   entrada[0][3] = ecSN.normalizar(ecSN.calcularMDF(amostras,
          pot2proxima), this.mdfMin, this.mdfMax);
190
191   for (int j = 0; j < 4; j++) {
192     if (j == saidaEsperada) {
193       saida[0][j] = 1.0;
194     } else {
195       saida[0][j] = 0.0;
196     }
197   }
198
199   MLDataSet testSet = new BasicMLDataSet(entrada, saida);
200   System.out.println("Neural Network Results:");
201   int i = 0;
202   double valorMaior = -1.0;
203   int maior = 0;
204   for (MLDataPair pair : testSet) {
205     final MLData output = network.compute(pair.getInput());
206     System.out.print(i++);
207     System.out.print(pair.getInput().getData(0) + ","
208         + pair.getInput().getData(1) + ","
209           + pair.getInput().getData(2) + ","
210             + pair.getInput().getData(3));
211     System.out.print("|real=");
212     for (Integer in : mapeamento.keySet()) {
213       System.out.print(output.getData(in) + ",");
214     }
215
216     System.out.print("|ideal=");
217     for (Integer in : mapeamento.keySet()) {
218       System.out.print(pair.getIdeal().getData(in) + ",");
219     }
220
221     valorMaior = -1.0;
222     maior = 0;
223
224     for (Integer in : mapeamento.keySet()) {
225       if (output.getData(in) > valorMaior) {
226         maior = in;
```

```
227         valorMaior = output.getData(in);
228       }
229     }
230   }
231
232   return mapeamento.get(maior);
233  }
234
235  public void carregarAmostras(List<AmostraRedeNeural> amostras
        ) {
236   this.amostras = amostras;
237  }
238
239  public List<AmostraRedeNeural> getAmostras() {
240   return amostras;
241  }
242
243  public void setMapeamento(Map<Integer, Comando> mapeamento) {
244   this.mapeamento = mapeamento;
245  }
246 }
```

```
1 package br.com.tomas.emgprocessing.sinal;
2
3 public interface AmostraListener {
4  void amostrasChegaram(Integer[] amostras);
5 }
```

```
1 package br.com.tomas.emgprocessing.sinal;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 public abstract class LeitorDeAmostras {
7  protected List<AmostraListener> listeners = new ArrayList<
      AmostraListener>();
8  protected AmostraInputStream ais;
9
10  public LeitorDeAmostras(AmostraInputStream ais) {
11   this.ais = ais;
```

```
12   }
13
14   public void addAmostraListener(AmostraListener al) {
15     this.listeners.add(al);
16   }
17
18   public void removeAmostraListener(AmostraListener al) {
19     this.listeners.remove(al);
20   }
21
22   public abstract void iniciarLeitura();
23
24   public abstract void parar();
25
26 }
```

```
1 package br.com.tomas.emgprocessing.sinal;
2
3 import java.io.IOException;
4 import java.util.ArrayList;
5 import java.util.List;
6
7
8 public class LeitorDeAmostrasDetectorDeSinal extends
     LeitorDeAmostras {
9   private boolean parar = false;
10
11   public LeitorDeAmostrasDetectorDeSinal(AmostraInputStream ais
       ) {
12     super(ais);
13     this.ais = ais;
14   }
15
16   public void iniciarLeitura() {
17     new Thread(new Runnable() {
18
19       @Override
20       public void run() {
21         try {
22           List<Integer> bufferSinal = new ArrayList<Integer>();
```

```
23          int periodoMedia = 5;
24          int[] mediaBuffer = new int[periodoMedia];
25          int amplitude;
26          int limiar = 180;
27          boolean temSinal = false;
28          double media = 0.0;
29          int i = 0;
30          while (!parar) {
31            amplitude = ais.lerAmostra();
32
33            if (temSinal) {
34              bufferSinal.add(amplitude);
35            } else {
36              mediaBuffer[i % periodoMedia] = amplitude;
37            }
38
39            if (i % periodoMedia == 0) {
40              media /= periodoMedia;
41              //System.out.println("Media: " + media);
42
43              if (!temSinal && media > limiar) {
44                System.out.println("Tem␣sinal");
45                temSinal = true;
46                for (int k = 0; k < mediaBuffer.length; k++) {
47                  if (Math.abs(mediaBuffer[k]) > limiar) {
48                    bufferSinal.add(mediaBuffer[k]);
49                  }
50                }
51              } else if (temSinal && media < limiar) {
52                temSinal = false;
53                for (int k = 0; k < mediaBuffer.length; k++) {
54                  if (Math.abs(mediaBuffer[k]) > limiar) {
55                    bufferSinal.add(mediaBuffer[k]);
56                  }
57                }
58                System.out.println("Acabou␣o␣sinal");
59
60
61                for (AmostraListener al: listeners) {
```

```
62                 al.amostrasChegaram(bufferSinal.toArray(new
                       Integer[0]));
63               }
64               bufferSinal = new ArrayList<Integer>();
65             }
66
67             media = 0.0;
68           }
69           media += Math.abs(amplitude);
70           i++;
71         }
72       } catch (IOException e) {
73         e.printStackTrace();
74       }
75     }
76   }).start();
77  }
78
79  @Override
80  public void parar() {
81    parar = true;
82  }
83 }
```

```
1 package br.com.tomas.emgprocessing.sinal;
2
3 import java.io.IOException;
4
5 public class LeitorDeAmostrasTamanhoFixo extends
    LeitorDeAmostras {
6  private AmostraInputStream ais;
7  private int bufferSize;
8  private boolean parar = false;
9
10 public LeitorDeAmostrasTamanhoFixo(AmostraInputStream ais,
     int bufferSize) {
11   super(ais);
12   this.ais = ais;
13   this.bufferSize = bufferSize;
14  }
```

```
15
16  @Override
17  public void iniciarLeitura() {
18    new Thread(new Runnable() {
19
20      @Override
21      public void run() {
22        try {
23          int b;
24          Integer[] buffer = new Integer[bufferSize];
25          int i = 0;
26          while (!parar) {
27            b = ais.lerAmostra();
28            buffer[i] = b;
29            i++;
30            if (i == bufferSize) {
31              for (AmostraListener al : listeners) {
32                al.amostrasChegaram(buffer);
33              }
34
35              i = 0;
36            }
37          }
38        } catch (IOException e) {
39          e.printStackTrace();
40        }
41      }
42    }).start();
43  }
44
45  @Override
46  public void parar() {
47    parar = true;
48  }
49 }
```

```
1 package br.com.tomas.emgprocessing.sinal;
2
3 import java.io.InputStream;
4
```

```
5 public interface LeitorDeSinal {
6   InputStream comecarLeitura() throws LeituraDeSinalException;
7   void pararLeitura();
8 }
```

```
1 package br.com.tomas.emgprocessing.sinal;
2
3 import java.io.IOException;
4 import java.io.InputStream;
5 import java.io.PipedInputStream;
6 import java.io.PipedOutputStream;
7
8 import javax.sound.sampled.AudioFormat;
9 import javax.sound.sampled.AudioSystem;
10 import javax.sound.sampled.DataLine;
11 import javax.sound.sampled.LineUnavailableException;
12 import javax.sound.sampled.TargetDataLine;
13
14 /**
15  * Classe responsável por ler o sinal na entrada de áudio.
16  *
17  * @author Tomás Godoi
18  */
19 public class LeitorDeSinalDeAudio implements LeitorDeSinal,
      Runnable {
20   private TargetDataLine line;
21   private AudioFormat format;
22   private PipedOutputStream out;
23   private int bufferSize;
24   private boolean parar;
25
26   public LeitorDeSinalDeAudio(AudioFormat af, int bufferSize)
        throws LineUnavailableException {
27     this.format = af;
28     this.bufferSize = bufferSize;
29
30     DataLine.Info info = new DataLine.Info(TargetDataLine.class,
           format);
31
32     if (!AudioSystem.isLineSupported(info)) {
```

```
33      System.out.println("Não␣tem␣suporte");
34    } else {
35      System.out.println("Tem␣suporte");
36    }
37
38    line = (TargetDataLine) AudioSystem.getLine(info);
39  }
40
41  public LeitorDeSinalDeAudio() throws LineUnavailableException
        {
42    this(new AudioFormat(8000.0f, 16, 1, true, true), 2048);
43  }
44
45  public InputStream comecarLeitura() throws
       LeituraDeSinalException {
46    try {
47      line.open(format);
48
49      System.out.println("Tamanho␣do␣buffer:␣" + line.
          getBufferSize());
50      System.out.println("Tamanho␣do␣frame:␣"
51          + line.getFormat().getFrameSize());
52      System.out.println("Qtde␣canais:␣"
53          + line.getFormat().getChannels());
54
55      PipedInputStream in = new PipedInputStream(this.bufferSize
          );
56
57      out = new PipedOutputStream(in);
58
59      Thread threadLeitora = new Thread(this, "LeitoraDeSinal");
60      threadLeitora.start();
61
62      return in;
63    } catch (LineUnavailableException e) {
64      throw new LeituraDeSinalException(e);
65    } catch (IOException e) {
66      throw new LeituraDeSinalException(e);
67    }
```

```
68   }
69
70   @Override
71   public void run() {
72     int qtdeBytesLidos;
73     byte[] data = new byte[this.bufferSize];
74
75     line.start();
76
77     System.out.println("Iniciou a linha...");
78
79     try {
80       while (!parar) {
81         qtdeBytesLidos = line.read(data, 0, data.length);
82         out.write(data, 0, qtdeBytesLidos);
83       }
84       line.close();
85     } catch (IOException e) {
86       e.printStackTrace();
87     }
88   }
89
90   @Override
91   public void pararLeitura() {
92     this.parar = true;
93   }
94 }
```

```
1 package br.com.tomas.emgprocessing.sinal.ui;
2
3 import javax.swing.JComponent;
4
5 public abstract class RenderizadorSEMG extends JComponent {
6   private static final long serialVersionUID = 1L;
7
8 }
```

```
1 package br.com.tomas.emgprocessing.sinal.ui;
2
3 import java.awt.Color;
```

```
4  import java.awt.Dimension;
5  import java.awt.Graphics;
6  import java.awt.Graphics2D;
7
8  import br.com.tomas.emgprocessing.sinal.AmostraListener;
9
10 public abstract class RenderizadorTamanhoFixo extends
       RenderizadorSEMG implements AmostraListener {
11   private static final long serialVersionUID = 1L;
12   private Dimension dimension;
13   protected Integer[] amostras;
14   protected int fatorEscalaVertical;
15
16   public RenderizadorTamanhoFixo(Dimension dimension, int
       fatorVertical) {
17     this.dimension = dimension;
18     this.fatorEscalaVertical = fatorVertical;
19     setOpaque(true);
20     setBackground(Color.WHITE);
21
22     amostras = new Integer[dimension.width];
23     for (int i = 0; i < amostras.length; i++) {
24       amostras[i] = 0;
25     }
26   }
27
28   public void setFatorEscalaVertical(int novoFator) {
29     fatorEscalaVertical = novoFator;
30   }
31
32   public Dimension getPreferredSize() {
33     return dimension;
34   }
35
36   @Override
37   protected void paintComponent(Graphics g) {
38     super.paintComponent(g);
39
40     Graphics2D g2 = (Graphics2D)g.create();
```

```
41
42    int w = getWidth() - getInsets().left - getInsets().right;
43    int h = getHeight() - getInsets().top - getInsets().bottom;
44
45    if (isOpaque()) {
46      g2.setPaint(getBackground());
47      g2.fillRect(0, 0, getWidth(), getHeight());
48    }
49
50    g2.setPaint(Color.BLACK);
51
52    g2.drawLine(0, h / 2, w, h / 2);
53
54    int dottedLineWidth = 3;
55    int iterations = h / (2 * dottedLineWidth);
56
57    for (int i = 0; i < iterations; i++) {
58      int startY = i * 2 * dottedLineWidth;
59      int endY = startY + dottedLineWidth;
60      g2.drawLine(w / 2, startY, w / 2, endY);
61    }
62
63    atualizarSinal(g2, w, h);
64  }
65
66  protected abstract void atualizarSinal(Graphics2D g2, int
       width, int heigth);
67
68  @Override
69  public void amostrasChegaram(Integer[] amostras) {
70    this.amostras = amostras;
71    this.repaint();
72  }
73 }
```

```
1 package br.com.tomas.emgprocessing.sinal.ui;
2
3 import java.awt.Dimension;
4 import java.awt.Graphics2D;
5
```

```
6 public class RenderizadorTamanhoFixoTempo extends
    RenderizadorTamanhoFixo {
7  private static final long serialVersionUID = 1L;
8
9  public RenderizadorTamanhoFixoTempo(Dimension dimension, int
     fatorVertical) {
10   super(dimension, fatorVertical);
11  }
12
13  @Override
14  protected void atualizarSinal(Graphics2D g2, int width, int
     heigth) {
15   int amplitude;
16   int yMeio = heigth / 2;
17   int amplitudeAnterior = 0;
18   for (int i = 0; i < amostras.length; i++) {
19    amplitude = amostras[i];
20
21    g2.drawLine(i, yMeio - amplitudeAnterior /
        fatorEscalaVertical, i,
22        yMeio - amplitude / fatorEscalaVertical);
23
24    amplitudeAnterior = amplitude;
25   }
26  }
27 }
```

```
1 package br.com.tomas.emgprocessing.sinal.ui;
2
3 import java.awt.Color;
4 import java.awt.Dimension;
5 import java.awt.Graphics;
6 import java.awt.Graphics2D;
7 import java.awt.Rectangle;
8 import java.util.ArrayList;
9 import java.util.List;
10
11 import javax.swing.SwingUtilities;
12
13 import br.com.tomas.emgprocessing.sinal.AmostraListener;
```

```
14
15 public class RenderizadorTamanhoVariavel extends
     RenderizadorSEMG implements
16    AmostraListener {
17   private static final long serialVersionUID = 1L;
18   private Dimension dimension;
19   protected List<Integer> amostras;
20   protected int fatorEscalaVertical;
21
22   public RenderizadorTamanhoVariavel(Dimension dimension, int
       fatorVertical) {
23     dimension.width = 0;
24     this.dimension = dimension;
25     this.fatorEscalaVertical = fatorVertical;
26     setOpaque(true);
27     setBackground(Color.WHITE);
28
29     amostras = new ArrayList<Integer>();
30   }
31
32   public Dimension getPreferredSize() {
33     return dimension;
34   }
35
36   public void setFatorEscalaVertical(int novoFator) {
37     fatorEscalaVertical = novoFator;
38   }
39
40   @Override
41   protected void paintComponent(Graphics g) {
42     super.paintComponent(g);
43
44     Graphics2D g2 = (Graphics2D) g.create();
45
46     int w = getWidth() - getInsets().left - getInsets().right;
47     int h = getHeight() - getInsets().top - getInsets().bottom;
48
49     if (isOpaque()) {
50       g2.setPaint(getBackground());
```

```
51      g2.fillRect(0, 0, getWidth(), getHeight());
52    }
53
54    g2.setPaint(Color.BLACK);
55
56    g2.drawLine(0, h / 2, w, h / 2);
57
58    atualizarSinal(g2, w, h);
59  }
60
61  protected void atualizarSinal(Graphics2D g2, int width, int
       height) {
62    int amplitude;
63    int yMeio = height / 2;
64    int amplitudeAnterior = 0;
65    for (int i = 0; i < amostras.size(); i++) {
66      amplitude = amostras.get(i);
67
68      g2.drawLine(i, yMeio - amplitudeAnterior /
           fatorEscalaVertical, i,
69          yMeio - amplitude / fatorEscalaVertical);
70
71      amplitudeAnterior = amplitude;
72    }
73  }
74
75  @Override
76  public void amostrasChegaram(Integer[] amostras) {
77    for (int i = 0; i < amostras.length; i++) {
78      this.amostras.add(amostras[i]);
79    }
80    this.dimension.width = this.amostras.size();
81    System.out.println(this.amostras.size());
82    this.repaint();
83    this.revalidate();
84    SwingUtilities.invokeLater(new Runnable() {
85      @Override
86      public void run() {
```

```
87          scrollRectToVisible(new Rectangle(getWidth() - getParent
                ().getWidth(), 0, getWidth(), getHeight()));
88        }
89      });
90    }
91
92    public void zerarAmostras() {
93      this.amostras = new ArrayList<Integer>();
94      this.dimension.width = this.amostras.size();
95      this.repaint();
96      this.revalidate();
97    }
98
99    public List<Integer> getAmostras() {
100     return this.amostras;
101   }
102 }
```

Anexos

ANEXO A – *Datasheets*

Este anexo apresenta algumas partes importantes dos *datasheets* dos dois CIs utilizados neste projeto, o AD620 e o TL074.

Low Cost Low Power Instrumentation Amplifier

AD620

FEATURES

- Easy to use
  - Gain set with one external resistor
    - (Gain range 1 to 10,000)
  - Wide power supply range (±2.3 V to ±18 V)
  - Higher performance than 3 op amp IA designs
  - Available in 8-lead DIP and SOIC packaging
  - Low power, 1.3 mA max supply current
- Excellent dc performance (B grade)
  - 50 µV max, input offset voltage
  - 0.6 µV/°C max, input offset drift
  - 1.0 nA max, input bias current
  - 100 dB min common-mode rejection ratio (G = 10)
- Low noise
  - 9 nV/√Hz @ 1 kHz, input voltage noise
  - 0.28 µV p-p noise (0.1 Hz to 10 Hz)
- Excellent ac specifications
  - 120 kHz bandwidth (G = 100)
  - 15 µs settling time to 0.01%

APPLICATIONS

- Weigh scales
- ECG and medical instrumentation
- Transducer interface
- Data acquisition systems
- Industrial process controls
- Battery-powered and portable equipment

Table 1. Next Generation Upgrades for AD620

Part	Comment
AD8221	Better specs at lower price
AD8222	Dual channel or differential out
AD8226	Low power, wide input range
AD8220	JFET input
AD8228	Best gain accuracy
AD8295	+2 precision op amps or differential out
AD8429	Ultra low noise

CONNECTION DIAGRAM

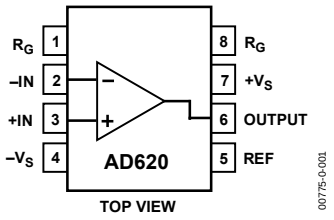


Figure 1. 8-Lead PDIP (N), CERDIP (Q), and SOIC (R) Packages

PRODUCT DESCRIPTION

The AD620 is a low cost, high accuracy instrumentation amplifier that requires only one external resistor to set gains of 1 to 10,000. Furthermore, the AD620 features 8-lead SOIC and DIP packaging that is smaller than discrete designs and offers lower power (only 1.3 mA max supply current), making it a good fit for battery-powered, portable (or remote) applications.

The AD620, with its high accuracy of 40 ppm maximum nonlinearity, low offset voltage of 50 µV max, and offset drift of 0.6 µV/°C max, is ideal for use in precision data acquisition systems, such as weigh scales and transducer interfaces. Furthermore, the low noise, low input bias current, and low power of the AD620 make it well suited for medical applications, such as ECG and noninvasive blood pressure monitors.

The low input bias current of 1.0 nA max is made possible with the use of Superβeta processing in the input stage. The AD620 works well as a preamplifier due to its low input voltage noise of 9 nV/√Hz at 1 kHz, 0.28 µV p-p in the 0.1 Hz to 10 Hz band, and 0.1 pA/√Hz input current noise. Also, the AD620 is well suited for multiplexed applications with its settling time of 15 µs to 0.01%, and its cost is low enough to enable designs with one in-amp per channel.

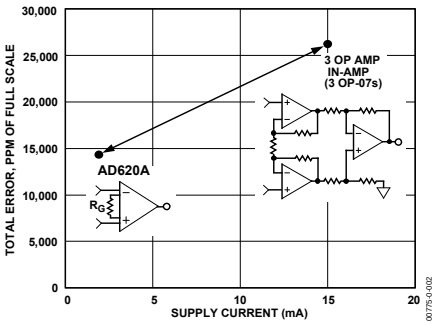


Figure 2. Three Op Amp IA Designs vs. AD620

Rev. H

One Technology Way, P.O. Box 9106, Norwood, MA 02062-9106, U.S.A.
Tel: 781.329.4700 **www.analog.com**
Fax: 781.326.8703

SPECIFICATIONS

Typical @ 25°C, V_S = ±15 V, and R_L = 2 kΩ, unless otherwise noted.

Table 2.

		AD620A			AD620B			AD620S[1]			
Parameter	**Conditions**	**Min**	**Typ**	**Max**	**Min**	**Typ**	**Max**	**Min**	**Typ**	**Max**	**Unit**
GAIN	G = 1 + (49.4 kΩ/R_G)										
Gain Range		1		10,000	1		10,000	1		10,000	
Gain Error[2]	V_{OUT} = ±10 V										
G = 1			0.03	0.10		0.01	0.02		0.03	0.10	%
G = 10			0.15	0.30		0.10	0.15		0.15	0.30	%
G = 100			0.15	0.30		0.10	0.15		0.15	0.30	%
G = 1000			0.40	0.70		0.35	0.50		0.40	0.70	%
Nonlinearity	V_{OUT} = −10 V to +10 V										
G = 1–1000	R_L = 10 kΩ		10	40		10	40		10	40	ppm
G = 1–100	R_L = 2 kΩ		10	95		10	95		10	95	ppm
Gain vs. Temperature											
	G = 1			10			10			10	ppm/°C
	Gain >1[2]			−50			−50			−50	ppm/°C
VOLTAGE OFFSET	(Total RTI Error = V_{OSI} + V_{OSO}/G)										
Input Offset, V_{OSI}	V_S = ±5 V to ± 15 V		30	125		15	50		30	125	µV
Overtemperature	V_S = ±5 V to ± 15 V			185			85			225	µV
Average TC	V_S = ±5 V to ± 15 V		0.3	1.0		0.1	0.6		0.3	1.0	µV/°C
Output Offset, V_{OSO}	V_S = ±15 V		400	1000		200	500		400	1000	µV
	V_S = ± 5 V			1500			750			1500	µV
Overtemperature	V_S = ±5 V to ± 15 V			2000			1000			2000	µV
Average TC	V_S = ±5 V to ± 15 V		5.0	15		2.5	7.0		5.0	15	µV/°C
Offset Referred to the Input vs. Supply (PSR)	V_S = ±2.3 V to ±18 V										
G = 1		80	100		80	100		80	100		dB
G = 10		95	120		100	120		95	120		dB
G = 100		110	140		120	140		110	140		dB
G = 1000		110	140		120	140		110	140		dB
INPUT CURRENT											
Input Bias Current			0.5	2.0		0.5	1.0		0.5	2	nA
Overtemperature				2.5			1.5			4	nA
Average TC			3.0			3.0			8.0		pA/°C
Input Offset Current			0.3	1.0		0.3	0.5		0.3	1.0	nA
Overtemperature				1.5			0.75			2.0	nA
Average TC			1.5			1.5			8.0		pA/°C
INPUT											
Input Impedance											
Differential			10||2			10||2			10||2		GΩ_pF
Common-Mode			10||2			10||2			10||2		GΩ_pF
Input Voltage Range[3]	V_S = ±2.3 V to ±5 V	$-V_S$ + 1.9		$+V_S$ − 1.2	$-V_S$ + 1.9		$+V_S$ − 1.2	$-V_S$ + 1.9		$+V_S$ − 1.2	V
Overtemperature		$-V_S$ + 2.1		$+V_S$ − 1.3	$-V_S$ + 2.1		$+V_S$ − 1.3	$-V_S$ + 2.1		$+V_S$ − 1.3	V
	V_S = ± 5 V to ±18 V	$-V_S$ + 1.9		$+V_S$ − 1.4	$-V_S$ + 1.9		$+V_S$ − 1.4	$-V_S$ + 1.9		$+V_S$ − 1.4	V
Overtemperature		$-V_S$ + 2.1		$+V_S$ − 1.4	$-V_S$ + 2.1		$+V_S$ + 2.1	$-V_S$ + 2.3		$+V_S$ − 1.4	V

Parameter	Conditions	AD620A Min	AD620A Typ	AD620A Max	AD620B Min	AD620B Typ	AD620B Max	AD620S[1] Min	AD620S[1] Typ	AD620S[1] Max	Unit
Common-Mode Rejection											
Ratio DC to 60 Hz with											
1 kΩ Source Imbalance	V_{CM} = 0 V to ± 10 V										
G = 1		73	90		80	90		73	90		dB
G = 10		93	110		100	110		93	110		dB
G = 100		110	130		120	130		110	130		dB
G = 1000		110	130		120	130		110	130		dB
OUTPUT											
Output Swing	R_L = 10 kΩ										
	V_S = ±2.3 V to ± 5 V	$-V_S$ + 1.1		$+V_S$ – 1.2	$-V_S$ + 1.1		$+V_S$ – 1.2	$-V_S$ + 1.1		$+V_S$ – 1.2	V
Overtemperature		$-V_S$ + 1.4		$+V_S$ – 1.3	$-V_S$ + 1.4		$+V_S$ – 1.3	$-V_S$ + 1.6		$+V_S$ – 1.3	V
	V_S = ±5 V to ± 18 V	$-V_S$ + 1.2		$+V_S$ – 1.4	$-V_S$ + 1.2		$+V_S$ – 1.4	$-V_S$ + 1.2		$+V_S$ – 1.4	V
Overtemperature		$-V_S$ + 1.6		$+V_S$ – 1.5	$-V_S$ + 1.6		$+V_S$ – 1.5	$-V_S$ + 2.3		$+V_S$ – 1.5	V
Short Circuit Current			±18			±18			±18		mA
DYNAMIC RESPONSE											
Small Signal –3 dB Bandwidth											
G = 1			1000			1000			1000		kHz
G = 10			800			800			800		kHz
G = 100			120			120			120		kHz
G = 1000			12			12			12		kHz
Slew Rate		0.75	1.2		0.75	1.2		0.75	1.2		V/µs
Settling Time to 0.01%	10 V Step										
G = 1–100			15			15			15		µs
G = 1000			150			150			150		µs
NOISE											
Voltage Noise, 1 kHz	$Total\ RTI\ Noise = \sqrt{(e^2_{ni})} + (e_{no}/G)^2$										
Input, Voltage Noise, e_{ni}			9	13		9	13		9	13	nV/√Hz
Output, Voltage Noise, e_{no}			72	100		72	100		72	100	nV/√Hz
RTI, 0.1 Hz to 10 Hz											
G = 1			3.0			3.0	6.0		3.0	6.0	µV p-p
G = 10			0.55			0.55	0.8		0.55	0.8	µV p-p
G = 100–1000			0.28			0.28	0.4		0.28	0.4	µV p-p
Current Noise	f = 1 kHz		100			100			100		fA/√Hz
0.1 Hz to 10 Hz			10			10			10		pA p-p
REFERENCE INPUT											
R_{IN}			20			20			20		kΩ
I_{IN}	V_{IN+}, V_{REF} = 0		50	60		50	60		50	60	µA
Voltage Range		$-V_S$ + 1.6		$+V_S$ – 1.6	$-V_S$ + 1.6		$+V_S$ – 1.6	$-V_S$ + 1.6		$+V_S$ – 1.6	V
Gain to Output		1 ± 0.0001			1 ± 0.0001			1 ± 0.0001			
POWER SUPPLY											
Operating Range[4]		±2.3		±18	±2.3		±18	±2.3		±18	V
Quiescent Current	V_S = ±2.3 V to ±18 V		0.9	1.3		0.9	1.3		0.9	1.3	mA
Overtemperature			1.1	1.6		1.1	1.6		1.1	1.6	mA
TEMPERATURE RANGE											
For Specified Performance		–40 to +85			–40 to +85			–55 to +125			°C

[1] See Analog Devices military data sheet for 883B tested specifications.
[2] Does not include effects of external resistor R_G.
[3] One input grounded. G = 1.
[4] This is defined as the same supply range that is used to specify PSR.

THEORY OF OPERATION

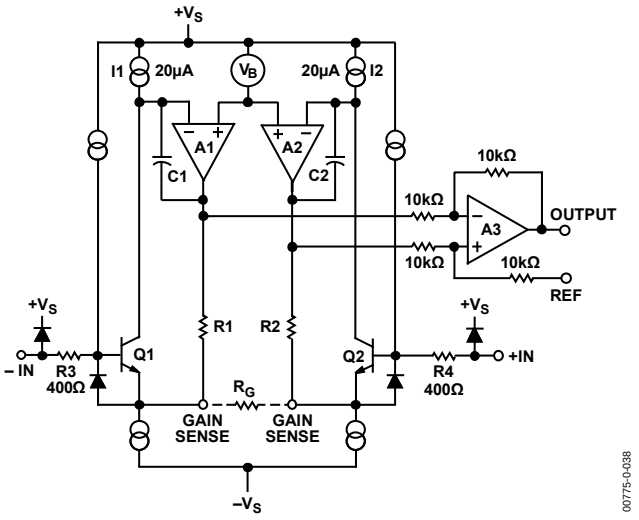


Figure 36. Simplified Schematic of AD620

The AD620 is a monolithic instrumentation amplifier based on a modification of the classic three op amp approach. Absolute value trimming allows the user to program gain *accurately* (to 0.15% at G = 100) with only one resistor. Monolithic construction and laser wafer trimming allow the tight matching and tracking of circuit components, thus ensuring the high level of performance inherent in this circuit.

The input transistors Q1 and Q2 provide a single differential-pair bipolar input for high precision (Figure 36), yet offer 10× lower input bias current thanks to Superβeta processing. Feedback through the Q1-A1-R1 loop and the Q2-A2-R2 loop maintains constant collector current of the input devices Q1 and Q2, thereby impressing the input voltage across the external gain setting resistor R_G. This creates a differential gain from the inputs to the A1/A2 outputs given by $G = (R1 + R2)/R_G + 1$. The unity-gain subtractor, A3, removes any common-mode signal, yielding a single-ended output referred to the REF pin potential.

The value of R_G also determines the transconductance of the preamp stage. As R_G is reduced for larger gains, the transconductance increases asymptotically to that of the input transistors. This has three important advantages: (a) Open-loop gain is boosted for increasing programmed gain, thus reducing gain related errors. (b) The gain-bandwidth product (determined by C1 and C2 and the preamp transconductance) increases with programmed gain, thus optimizing frequency response. (c) The input voltage noise is reduced to a value of 9 nV/√Hz, determined mainly by the collector current and base resistance of the input devices.

The internal gain resistors, R1 and R2, are trimmed to an absolute value of 24.7 kΩ, allowing the gain to be programmed accurately with a single external resistor.

The gain equation is then

$$G = \frac{49.4k\Omega}{R_G} + 1$$

$$R_G = \frac{49.4k\Omega}{G - 1}$$

Make vs. Buy: a Typical Bridge Application Error Budget

The AD620 offers improved performance over "homebrew" three op amp IA designs, along with smaller size, fewer components, and 10× lower supply current. In the typical application, shown in Figure 37, a gain of 100 is required to amplify a bridge output of 20 mV full-scale over the industrial temperature range of −40°C to +85°C. Table 4 shows how to calculate the effect various error sources have on circuit accuracy.

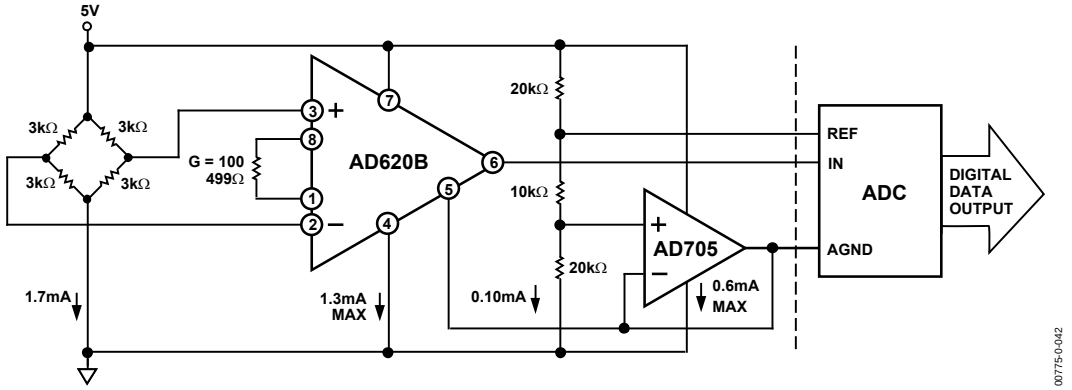


Figure 38. A Pressure Monitor Circuit that Operates on a 5 V Single Supply

Pressure Measurement

Although useful in many bridge applications, such as weigh scales, the AD620 is especially suitable for higher resistance pressure sensors powered at lower voltages where small size and low power become more significant.

Figure 38 shows a 3 kΩ pressure transducer bridge powered from 5 V. In such a circuit, the bridge consumes only 1.7 mA. Adding the AD620 and a buffered voltage divider allows the signal to be conditioned for only 3.8 mA of total supply current.

Small size and low cost make the AD620 especially attractive for voltage output pressure transducers. Since it delivers low noise and drift, it also serves applications such as diagnostic noninvasive blood pressure measurement.

Medical ECG

The low current noise of the AD620 allows its use in ECG monitors (Figure 39) where high source resistances of 1 MΩ or higher are not uncommon. The AD620's low power, low supply voltage requirements, and space-saving 8-lead mini-DIP and SOIC package offerings make it an excellent choice for battery-powered data recorders.

Furthermore, the low bias currents and low current noise, coupled with the low voltage noise of the AD620, improve the dynamic range for better performance.

The value of capacitor C1 is chosen to maintain stability of the right leg drive loop. Proper safeguards, such as isolation, must be added to this circuit to protect the patient from possible harm.

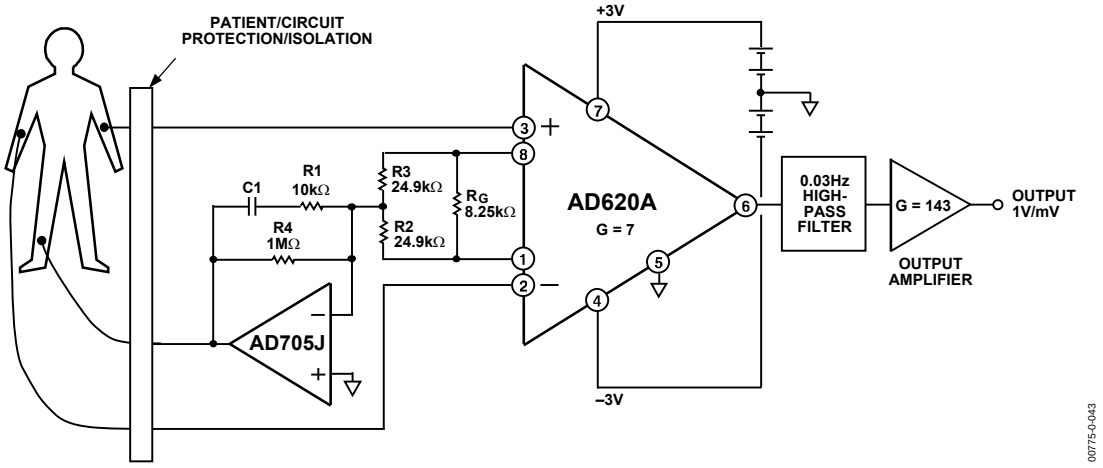


Figure 39. A Medical ECG Monitor Circuit

Precision V-I Converter

The AD620, along with another op amp and two resistors, makes a precision current source (Figure 40). The op amp buffers the reference terminal to maintain good CMR. The output voltage, V_X, of the AD620 appears across R1, which converts it to a current. This current, less only the input bias current of the op amp, then flows out to the load.

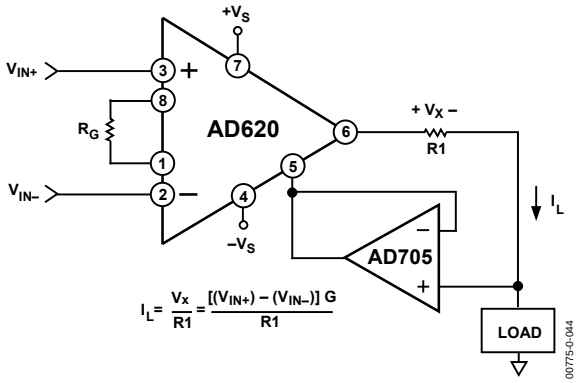


Figure 40. Precision Voltage-to-Current Converter (Operates on 1.8 mA, ±3 V)

GAIN SELECTION

The AD620 gain is resistor-programmed by R_G, or more precisely, by whatever impedance appears between Pins 1 and 8. The AD620 is designed to offer accurate gains using 0.1% to 1% resistors. Table 5 shows required values of R_G for various gains. Note that for G = 1, the R_G pins are unconnected ($R_G = \infty$). For any arbitrary gain, R_G can be calculated by using the formula:

$$R_G = \frac{49.4\,k\Omega}{G-1}$$

To minimize gain error, avoid high parasitic resistance in series with R_G; to minimize gain drift, R_G should have a low TC—less than 10 ppm/°C—for the best performance.

Table 5. Required Values of Gain Resistors

1% Std Table Value of R_G(Ω)	Calculated Gain	0.1% Std Table Value of R_G(Ω)	Calculated Gain
49.9 k	1.990	49.3 k	2.002
12.4 k	4.984	12.4 k	4.984
5.49 k	9.998	5.49 k	9.998
2.61 k	19.93	2.61 k	19.93
1.00 k	50.40	1.01 k	49.91
499	100.0	499	100.0
249	199.4	249	199.4
100	495.0	98.8	501.0
49.9	991.0	49.3	1,003.0

INPUT AND OUTPUT OFFSET VOLTAGE

The low errors of the AD620 are attributed to two sources, input and output errors. The output error is divided by *G* when referred to the input. In practice, the input errors dominate at high gains, and the output errors dominate at low gains. The total V_{OS} for a given gain is calculated as

Total Error RTI = input error + (output error/G)

Total Error RTO = (input error × G) + output error

REFERENCE TERMINAL

The reference terminal potential defines the zero output voltage and is especially useful when the load does not share a precise ground with the rest of the system. It provides a direct means of injecting a precise offset to the output, with an allowable range of 2 V within the supply voltages. Parasitic resistance should be kept to a minimum for optimum CMR.

INPUT PROTECTION

The AD620 safely withstands an input current of ±60 mA for several hours at room temperature. This is true for all gains and power on and off, which is useful if the signal source and amplifier are powered separately. For longer time periods, the input current should not exceed 6 mA.

For input voltages beyond the supplies, a protection resistor should be placed in series with each input to limit the current to 6 mA. These can be the same resistors as those used in the RFI filter. High values of resistance can impact the noise and AC CMRR performance of the system. Low leakage diodes (such as the BAV199) can be placed at the inputs to reduce the required protection resistance.

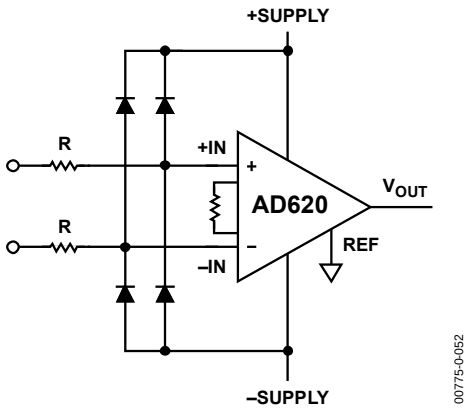


Figure 41. Diode Protection for Voltages Beyond Supply

RF INTERFERENCE

All instrumentation amplifiers rectify small out of band signals. The disturbance may appear as a small dc voltage offset. High frequency signals can be filtered with a low pass R-C network placed at the input of the instrumentation amplifier. Figure 42 demonstrates such a configuration. The filter limits the input

signal according to the following relationship:

$$FilterFreq_{DIFF} = \frac{1}{2\pi R(2C_D + C_C)}$$

$$FilterFreq_{CM} = \frac{1}{2\pi RC_C}$$

where $C_D \geq 10C_C$.

C_D affects the difference signal. C_C affects the common-mode signal. Any mismatch in $R \times C_C$ degrades the AD620 CMRR. To avoid inadvertently reducing CMRR-bandwidth performance, make sure that C_C is at least one magnitude smaller than C_D. The effect of mismatched C_Cs is reduced with a larger C_D:C_C ratio.

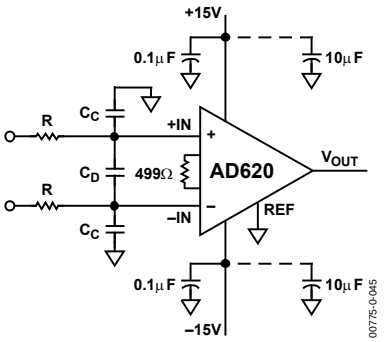


Figure 42. Circuit to Attenuate RF Interference

COMMON-MODE REJECTION

Instrumentation amplifiers, such as the AD620, offer high CMR, which is a measure of the change in output voltage when both inputs are changed by equal amounts. These specifications are usually given for a full-range input voltage change and a specified source imbalance.

For optimal CMR, the reference terminal should be tied to a low impedance point, and differences in capacitance and resistance should be kept to a minimum between the two inputs. In many applications, shielded cables are used to minimize noise; for best CMR over frequency, the shield should be properly driven. Figure 43 and Figure 44 show active data guards that are configured to improve ac common-mode rejections by "bootstrapping" the capacitances of input cable shields, thus minimizing the capacitance mismatch between the inputs.

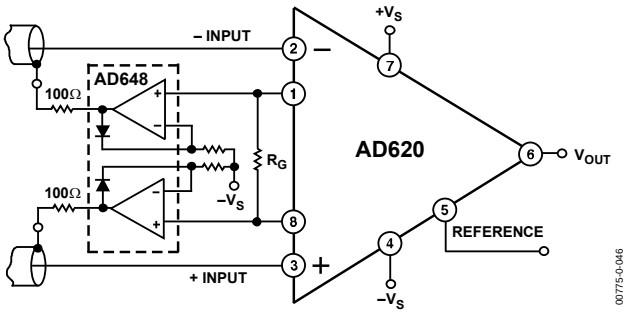


Figure 43. Differential Shield Driver

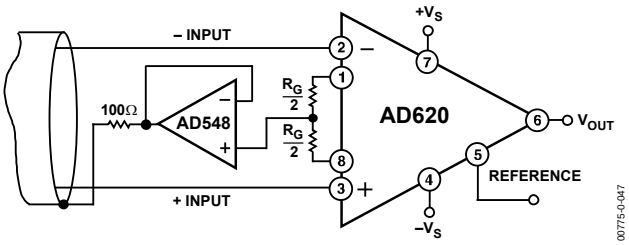


Figure 44. Common-Mode Shield Driver

GROUNDING

Since the AD620 output voltage is developed with respect to the potential on the reference terminal, it can solve many grounding problems by simply tying the REF pin to the appropriate "local ground."

To isolate low level analog signals from a noisy digital environment, many data-acquisition components have separate analog and digital ground pins (Figure 45). It would be convenient to use a single ground line; however, current through ground wires and PC runs of the circuit card can cause hundreds of millivolts of error. Therefore, separate ground returns should be provided to minimize the current flow from the sensitive points to the system ground. These ground returns must be tied together at some point, usually best at the ADC package shown in Figure 45.

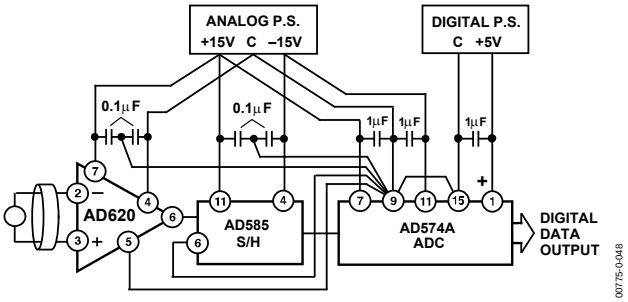


Figure 45. Basic Grounding Practice

- **Low Power Consumption**
- **Wide Common-Mode and Differential Voltage Ranges**
- **Low Input Bias and Offset Currents**
- **Output Short-Circuit Protection**
- **Low Total Harmonic Distortion . . . 0.003% Typ**
- **Low Noise**
  V_n = 18 nV/$\sqrt{\text{Hz}}$ Typ at f = 1 kHz
- **High Input Impedance . . . JFET Input Stage**
- **Internal Frequency Compensation**
- **Latch-Up-Free Operation**
- **High Slew Rate . . . 13 V/μs Typ**
- **Common-Mode Input Voltage Range Includes V_{CC+}**

description/ordering information

The JFET-input operational amplifiers in the TL07x series are similar to the TL08x series, with low input bias and offset currents and fast slew rate. The low harmonic distortion and low noise make the TL07x series ideally suited for high-fidelity and audio preamplifier applications. Each amplifier features JFET inputs (for high input impedance) coupled with bipolar output stages integrated on a single monolithic chip.

The C-suffix devices are characterized for operation from 0°C to 70°C. The I-suffix devices are characterized for operation from –40°C to 85°C. The M-suffix devices are characterized for operation over the full military temperature range of –55°C to 125°C.

description/ordering information (continued)

ORDERING INFORMATION

T_A	V_{IO}max AT 25°C	PACKAGE†		ORDERABLE PART NUMBER	TOP-SIDE MARKING
0°C to 70°C	10 mV	PDIP (P)	Tube of 50	TL071CP	TL071CP
			Tube of 50	TL072CP	TL072CP
		PDIP (N)	Tube of 25	TL074CN	TL074CN
		SOIC (D)	Tube of 75	TL071CD	TL071C
			Reel of 2500	TL071CDR	
			Tube of 75	TL072CD	TL072C
			Reel of 2500	TL072CDR	
			Tube of 50	TL074CD	TL074C
			Reel of 2500	TL074CDR	
		SOP (NS)	Reel of 2000	TL074CNSR	TL074
		SOP (PS)	Reel of 2000	TL071CPSR	TL071
			Reel of 2000	TL072CPSR	T072
		TSSOP (PW)	Reel of 2000	TL072CPWR	T072
			Tube of 90	TL074CPW	T074
			Reel of 2000	TL074CPWR	
	6 mV	PDIP (P)	Tube of 50	TL071ACP	TL071ACP
			Tube of 50	TL072ACP	TL072ACP
		PDIP (N)	Tube of 25	TL074ACN	TL074ACN
		SOIC (D)	Tube of 75	TL071ACD	071AC
			Reel of 2500	TL071ACDR	
			Tube of 75	TL072ACD	072AC
			Reel of 2500	TL072ACDR	
			Tube of 50	TL074ACD	TL074AC
			Reel of 2500	TL074ACDR	
		SOP (PS)	Reel of 2000	TL072ACPSR	T072A
		SOP (NS)	Reel of 2000	TL074ACNSR	TL074A
	3 mV	PDIP (P)	Tube of 50	TL071BCP	TL071BCP
			Tube of 50	TL072BCP	TL072BCP
		PDIP (N)	Tube of 25	TL074BCN	TL074BCN
		SOIC (D)	Tube of 75	TL071BCD	071BC
			Reel of 2500	TL071BCDR	
			Tube of 75	TL072BCD	072BC
			Reel of 2500	TL072BCDR	
			Tube of 50	TL074BCD	TL074BC
			Reel of 2500	TL074BCDR	
		SOP (NS)	Reel of 2000	TL074BCNSR	TL074B

† Package drawings, standard packing quantities, thermal data, symbolization, and PCB design guidelines are available at www.ti.com/sc/package.

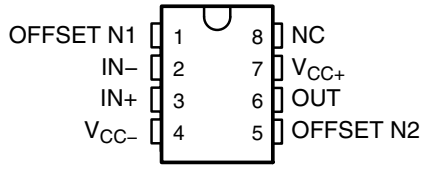
description/ordering information (continued)

ORDERING INFORMATION

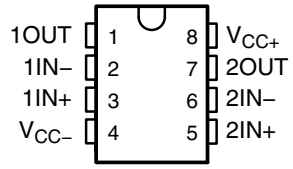
T_A	V_{IO}max AT 25°C	PACKAGE†		ORDERABLE PART NUMBER	TOP-SIDE MARKING
–40°C to 85°C	6 mV	PDIP (P)	Tube of 50	TL071IP	TL071IP
			Tube of 50	TL072IP	TL072IP
		PDIP (N)	Tube of 25	TL074IN	TL074IN
		SOIC (D)	Tube of 75	TL071ID	TL071I
			Reel of 2500	TL071IDR	
			Tube of 75	TL072ID	TL072I
			Reel of 2500	TL072IDR	
			Tube of 50	TL074ID	TL074I
			Reel of 2500	TL074IDR	
–55°C to 125°C	6 mV	CDIP (JG)	Tube of 50	TL072MJGB	TL072MJGB
		CFP (U)	Tube of 150	TL072MUB	TL072MUB
		LCCC (FK)	Tube of 55	TL072MFKB	TL072MFKB
	9 mV	CDIP (J)	Tube of 25	TL074MJB	TL074MJB
		CFP (W)	Tube of 25	TL074MWB	TL074MWB
		LCCC (FK)	Tube of 55	TL074MFKB	TL074MFKB

† Package drawings, standard packing quantities, thermal data, symbolization, and PCB design guidelines are available at www.ti.com/sc/package.

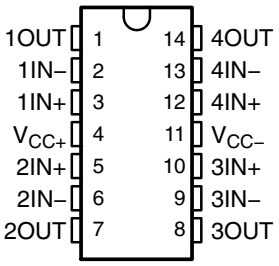
TL071, TL071A, TL071B
D, P, OR PS PACKAGE
(TOP VIEW)



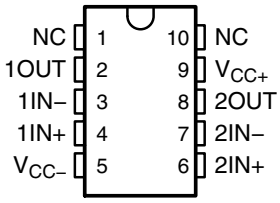
TL072, TL072A, TL072B
D, JG, P, PS, OR PW PACKAGE
(TOP VIEW)



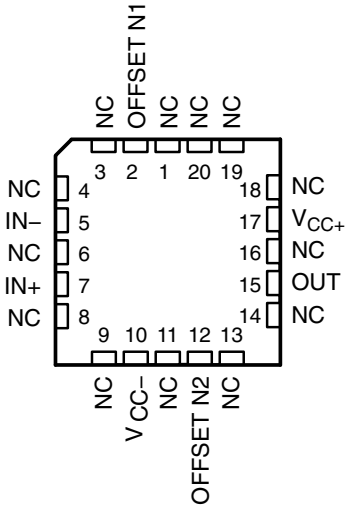
TL074A, TL074B
D, J, N, NS, OR PW PACKAGE
TL074 . . . D, J, N, NS, PW,
OR W PACKAGE
(TOP VIEW)



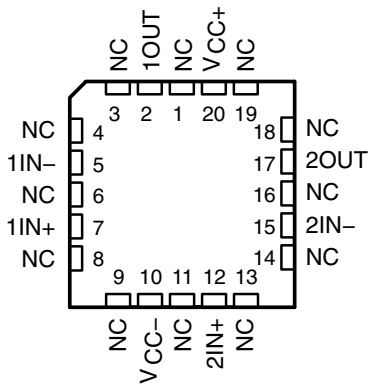
TL072
U PACKAGE
(TOP VIEW)



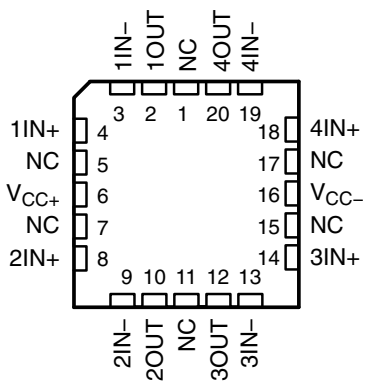
TL071
FK PACKAGE
(TOP VIEW)



TL072
FK PACKAGE
(TOP VIEW)



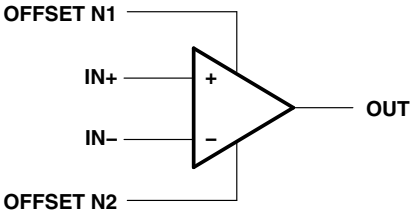
TL074
FK PACKAGE
(TOP VIEW)



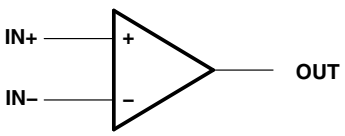
NC – No internal connection

symbols

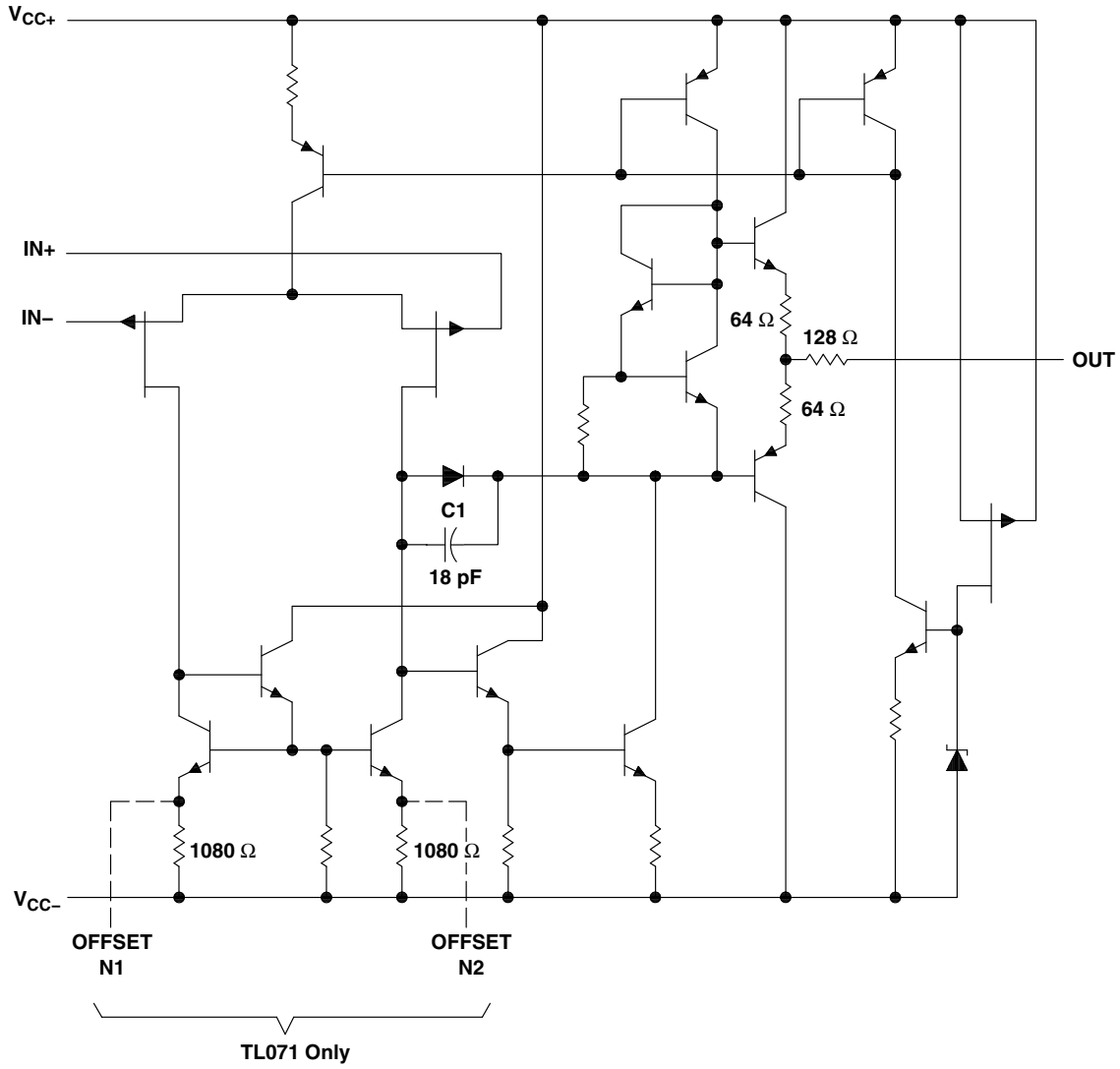
TL071



TL072 (each amplifier)
TL074 (each amplifier)



schematic (each amplifier)



All component values shown are nominal.

COMPONENT COUNT†			
COMPONENT TYPE	TL071	TL072	TL074
Resistors	11	22	44
Transistors	14	28	56
JFET	2	4	6
Diodes	1	2	4
Capacitors	1	2	4
epi-FET	1	2	4

† Includes bias and trim circuitry

APPLICATION INFORMATION

Table of Application Diagrams

APPLICATION DIAGRAM	PART NUMBER	FIGURE
0.5-Hz square-wave oscillator	TL071	23
High-Q notch filter	TL071	24
Audio-distribution amplifier	TL074	25
100-kHz quadrature oscillator	TL072	26
AC amplifier	TL071	27

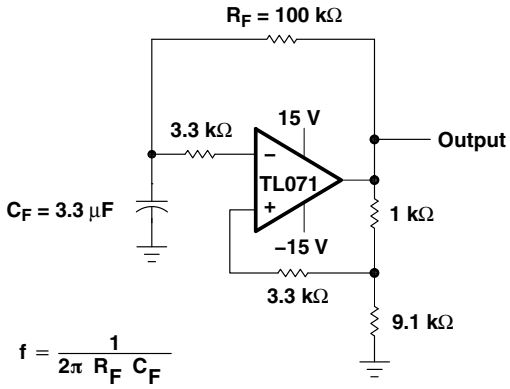


Figure 23. 0.5-Hz Square-Wave Oscillator

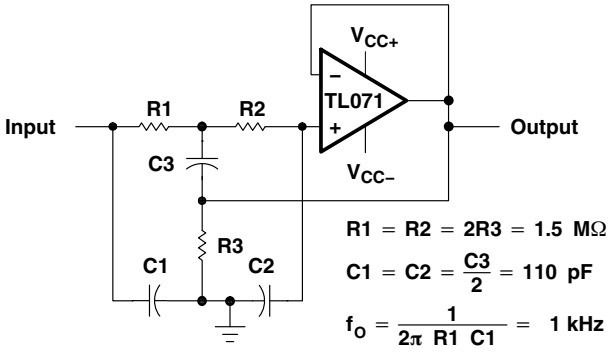


Figure 24. High-Q Notch Filter

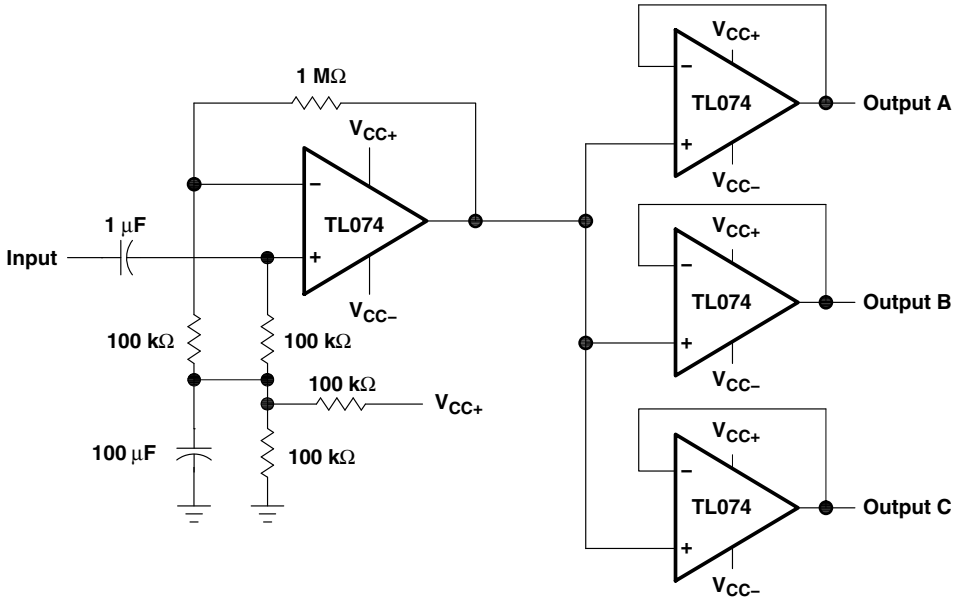


Figure 25. Audio-Distribution Amplifier