TRABALHO DE GRADUAÇÃO

SISTEMA DE AQUISIÇÃO E PROCESSAMENTO DE SINAIS MIOELÉTRICOS PARA CONTROLE DE UM BRAÇO ROBÓTICO UTILIZANDO O MYO ARMBAND

PHILIPPE DIAS ARAÚJO

Brasília, 30 de Janeiro, 2019

UNIVERSIDADE DE BRASÍLIA – UnB

FACULDADE DE TECNOLOGIA

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

UNIVERSIDADE DE BRASÍLIA – UnB

FACULDADE DE TECNOLOGIA

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASÍLIA
Faculdade de Tecnologia
Departamento de Engenharia Elétrica

PHILIPPE DIAS ARAÚJO

SISTEMA DE AQUISIÇÃO E PROCESSAMENTO DE SINAIS MIOELÉTRICOS PARA CONTROLE DE UM BRAÇO ROBÓTICO UTILIZANDO O MYO ARMBAND

UNIVERSIDADE DE BRASÍLIA
Faculdade de Tecnologia
Departamento de Engenharia Elétrica

PHILIPPE DIAS ARAÚJO

SISTEMA DE AQUISIÇÃO E PROCESSAMENTO DE SINAIS MIOELÉTRICOS PARA CONTROLE DE UM BRAÇO ROBÓTICO UTILIZANDO O MYO ARMBAND

Trabalho final de graduação submetido ao Departamento de Engenharia Elétrica da Faculdade de Tecnologia da Universidade de Brasília como parte dos requisitos necessários para a obtenção do grau de Engenheiro Eletricista.

APROVADO POR:

_____

Prof. Adson Ferreira da Rocha, Dr. (ENE/UnB)
(Orientador)

_____

Luiz Barbosa
(Examinador interno)

_____

Pedro Inazawa
(Examinador interno)

Brasília, 30 de Janeiro de 2019.

FICHA CATALOGRÁFICA

ARAUJO, PHILIPPE DIAS

Sistema de Aquisição e Processamento de Sinais Mioelétricos Para Controle de Um Braço Robótico Utilizando o Myo Armband. [Distrito Federal] 2019.

xi, 74p., 210 x 297 mm (ENE/FT/UnB, Engenheiro Eletricista, Engenharia Elétrica, 2019). Trabalho de Graduação Universidade de Brasília, Faculdade de Tecnologia.

Departamento de Engenharia Elétrica

1. EMG
2. Arduino DUE
3. Interação Homem Máquina.
4. Braço Robótico
5. Myo Armband
6. Unity 3D

I. ENE/FT/UnB
II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

ARAUJO, P. D. (2019). Sistema de Aquisição e Processamento de Sinais Mioelétricos Para Controle de Um Braço Robótico Utilizando o Myo Armband. Trabalho de Graduação em Engenharia Elétrica, Publicação 2018, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 74p.

CESSÃO DE DIREITOS

AUTOR: Philippe Dias Araújo.

TÍTULO: Sistema de Aquisição e Processamento de Sinais Mioelétricos Para Controle de Um Braço Robótico Utilizando o Myo Armband.

GRAU: Graduação em Engenharia Elétrica ANO: 2019.

_____

Philippe Dias Araújo
Departamento de Eng. Elétrica (ENE) - FT
Universidade de Brasília (UnB)
Campus Darcy Ribeiro
CEP 70919-970 - Brasília - DF – Brasil

AGRADECIMENTOS

Agradeço primeiramente a Deus, por tanta luz, por tantas bênçãos.

Agradeço aos meus heróis, Sr. Antônio de Jesus Araújo Pereira, Dona Maria de Lourdes da Silva Pereira e Sara Araújo Pereira, que fizeram o impossível, durante 24 anos, para me fornecer as melhores oportunidades possíveis. Todo meu esforço é pouco comparado ao que vocês fazem por mim. Essa conquista é nossa.

Agradeço minha namorada Brenda Vitória pelo apoio e amizade em todos os momentos ao longo dos anos. Obrigado por ser minha amiga, companheira e parceira de vida.

Agradeço aos professores que contribuíram para que essa caminhada fosse frutífera e valiosa.

Agradeço aos amigos e colegas que tornaram essa jornada mais leve ao longo dos anos.

Philippe Dias Araújo.

RESUMO

Testemunhamos uma verdadeira revolução na forma de interagir com as máquinas na nos últimos anos. Muitas invenções revolucionárias das décadas passadas foram vencidas pelo tempo ou se reduziram a relíquias. Cada vez mais, os dispositivos estão menores e melhores. Passamos mais tempo em contato com determinadas tecnologias do que com familiares. Sendo assim, essa nova era demanda cada vez mais interfaces intuitivas, eficientes e práticas. Este projeto tem o intuito de desenvolver um sistema de baixo custo para controle de um braço robótico em tempo real, que não utilize rotinas pré-programadas, e sim gestos e movimentos feitos com o braço. Utiliza-se o sensor Myo Armband, da Thalmic Labs, para extrair dados eletromiográficos e espaciais, a partir da sua Unidade de Medida Inercial (IMU). Realiza-se o processamento pela ferramenta de criação de jogos Unity 3D e atuação com o Microcontrolador Arduino DUE. O sistema apresenta resultados promissores ao conseguir realizar tarefas que exigem determinada destreza.

Palavras-chave: Interação Homem-Máquina, Myo Armband, EMG, Arduino, Unity 3D, Braço Robótico.

ABSTRACT

We witnessed a real revolution in the way we interacted with machines in recent years. Many revolutionary inventions of past decades have been weather-beaten or reduced to relics. Increasingly, the devices are smaller and better. We spend more time in touch with certain technologies than with family members. As such, this new era increasingly demands intuitive, efficient and practical interfaces. This project aims to develop a low cost system to control a robotic arm in real time, not using preprogrammed routines, but gestures and movements made with the arm. The Myo Armband sensor from Thalmic Labs is used to extract electromyographic and spatial data from its Inertial Measurement Unit (IMU). The processing is done by the game engine tool Unity 3D and actuating with the Arduino DUE Microcontroller. The system presents promising results by being able to perform tasks that require certain skill.

Keywords: Man-Machine interaction, Myo Armband, EMG, Arduino, Unity 3D, Robotic Arm.

SUMÁRIO

LISTA DE FIGURAS

1. INTRODUÇÃO

Vivemos em um mundo seletivamente integrado. Notícias, informações sobre acontecimentos ocorrendo nesse exato momento, em um país do outro lado do globo terrestre, podem ser transmitidas de forma simultânea para todo o mundo. A velocidade sem precedentes com que o conjunto hardware e software vem se desenvolvendo, nos permite viver de forma conectada em tempo integral, consumindo, gerando e transmitindo dados de forma consciente e inconsciente. Tamanha evolução não traz consigo somente benefícios. Nem mesmo os maiores pensadores do século passado seriam capazes de prever, por exemplo, que a divulgação de dados em determinadas plataformas digitais, de forma muitas vezes parcial, visando determinado fim, teria influência e poderio suficiente para influenciar, de forma ávida, o pensamento de uma nação inteira na eleição de um candidato a presidência da república. A forma com que a tecnologia é empregada e a interface com o ser humano é, e sempre será, tema de reflexão, de engenheiros à filósofos [LAWLER *et al.*,2005).

O surgimento de inúmeras ferramentas tecnológicas, com diversos propósitos e aplicações, revolucionou não só a forma com que nos comunicamos, mas também como vemos o mundo. Dentre os frutos do que se tem chamado de *A Quarta Revolução Industrial*, podemos citar smartphones capazes de desarmar disjuntores gerais de residências a partir de qualquer lugar do mundo (PARKER *et al.,* 2016) e drones que antecipam e controlam pragas em largas áreas de cultivos nos campos (PERRYMAN *et al.,* 2017), além da inesgotável variedade de conteúdo online para entreter os mais diversos perfis de usuários. Mas o fruto de maior valor pode estar em um outro segmento: medicina. Ou melhor, no corpo humano.

Fármacos, vacinas, tratamentos. São incontáveis os exemplos em que o desenvolvimento de dispositivos e processos mais robustos, com funções variadas, vem auxiliando profissionais da saúde. Esses que, muitas vezes com poucos recursos, se vêem fadados a tomarem decisões de vida ou morte, sem o devido embasamento técnico. Embasamento que hoje, devido a aplicações de modelos matemáticos, físicos e químicos, geram um ferramental enorme para combater condições que há dez anos atrás eram tidas como irremediáveis.

No escopo deste trabalho, o braço humano é um gerador de dados. Os dados gerados são denominados sinais mioelétricos de superfície. Um bracelete, posicionado logo abaixo do

cotovelo é a primeira ferramenta tecnológica do projeto. Ele é capaz de ler não só os sinais mioelétricos mas também dados inerciais, como a rotação do punho, e enviá-los via bluetooth para a segunda ferramenta, o computador. Esse é capaz de armazenar esses dados e aplicar modelos estatísticos, ou redes neurais, capazes de reconhecer e gerar padrões para o controle da terceira ferramenta, um braço robótico, ou prótese. Essa última não é, ainda, capaz de trazer de volta ao usuário todo sua mobilidade original, e nem de longe é capaz de remediar o dano psicológico causado por tal trauma, mas é um passo inicial da ciência em direção a um futuro de maior mobilidade a todos.

1. 1 CONTEXTUALIZAÇÃO

O controle de um dispositivo externo a partir de sinais fisiológicos, esteja ele acoplado ao corpo humano ou não, é um conceito relativamente novo, com alguns estudos e experimentos documentados nas décadas de 50 e 60 (FARINA *et al.*, 2014). A motivação ao redor da pesquisa e desenvolvimento nesse segmento deriva, dentre outras fontes, de uma enorme aplicabilidade na área de saúde, onde próteses, por exemplo, se mostram muitas vezes como a última solução viável às consequências de acidentes e estágios avançados de doenças agressivas. Dados da Access Prosthethics apontam que:

- Nos Estados Unidos da América existam 2.1 milhões de pessoas convivendo com a perda de algum membro, número que dobrará até 2050
- 185 mil pessoas são amputadas todos anos, uma média de 300 a 500 amputações por dia.
- Por volta de 30% das pessoas que sofreram alguma amputação sofrem de depressão e/ou ansiedade.
- O custo de tratamentos médicos por toda a vida de pessoas amputadas é estimado em $509.275,00 dólares, contra $361.200,00 de pessoas sem amputações.
- O custo estimado para convênios de saúde públicos e privados é por volta de 12 bilhões de dólares, anualmente.
- O número amputações causadas por diabetes cresceu em 24% entre 1988 e 2009.

As estatísticas são alarmantes e não representam uma realidade exclusivamente norte-americana. No Brasil, a diabetes foi a causa de 11.580 amputações em 2016, subindo para 12.478 em 2017 (SOARES, 2018) e segue o padrão de crescimento no ano de 2018. Os dados

levantados pela pesquisa ajudam a corroborar o que famílias e pacientes relatam: após a perda de um membro, em geral, a vida torna-se mais cara, as oportunidades de trabalho podem se limitar, atividades do dia-a-dia podem se tornar complexas, afetando diretamente a confiança e o bem-estar do paciente.

Dentro desse contexto, é extremamente necessário que governos, indústrias e universidades trabalhem juntos para prevenir as condições causadoras da perda de membros e minimizar os danos quando a situação é irreversível. Além disso, o resultado dessa colaboração precisa ser socialmente, demograficamente e economicamente abrangente. Muitos pacientes, possivelmente a maioria deles, não detêm de recursos suficientes para se locomover em direção a grandes centros comerciais em busca de um tratamento personalizado e eficiente.

No âmbito acadêmico-científico, o advento de novas tecnologias e recursos nas áreas de processamentos de sinais, sensoriamento, prototipagem, protocolos de comunicação, circuitos integrados, manufatura de baixo custo com o auxílio de impressoras 3D, são alguns dos elementos que promoveram uma verdadeira revolução no desenvolvimento de dispositivos capazes de auxiliar pacientes em inúmeras aplicações.

Como exemplo do fruto da integração de elementos como os acima citados, temos o dispositivo de captação de sinais usado neste trabalho, o Myo Armband (THALMIC LABS, 2015). Trata-se de um bracelete tipicamente utilizado como ferramenta de controle de interfaces visuais, como apresentações em slideshow e navegação em menus interativos. Além dessas aplicações, o dispositivo da Thalmic Labs se apresenta como uma interessante opção em termos de custo, praticidade, mobilidade e desempenho, para aqueles que buscam um meio de adquirir sinais mioelétricos, processá-los e transmiti-los de forma rápida e confiável, independentemente do agente atuador.

1. 2 OBJETIVOS DO TRABALHO

O objetivo principal deste trabalho consiste no desenvolvimento de um sistema de baixo custo, com uma interface simples para controle de um braço robótico, incluindo a aquisição de sinais mioelétricos e espaciais e a utilização desses dados para a atuação com auxílio de um microcontrolador. Deseja-se elaborar uma interface gráfica limpa e amigável, utilizando uma

plataforma que permite um aperfeiçoamento da aplicação em projetos futuros e integração com ferramentas mais atuais. Para tanto, são definidos como objetivos específicos:

- Realizar uma revisão bibliográfica acerca do estado-da-arte dos métodos de interação homem-máquina.
- Utilizar um microcontrolador de baixo custo como ferramenta de atuação do sistema.
- Calibrar servomotore e associar o controle destes aos gestos performados pelo usuário.
- Extrair os dados de EMG e da IMU do Myo Armband.
- Desenvolver uma interface gráfica capaz de apresentar as informações extraídas do Myo Armband e disponibilizá-las ao atuador;
- Desenvolver um método de controle do braço robótico que utilize apenas os dados extraídos pelo sensor, sem nenhuma rotina pré-programada.

1.2 ORGANIZAÇÃO DO TEXTO

A estrutura de organização do texto para este trabalho é a seguinte:

- No capítulo 2 realizar-se-á a revisão bibliográfica do tema. Levanta-se o histórico e o estado-da-arte da Interação Homem-Máquina, além de comentar sobre EMG e IMUs .
- No capítulo 3 discute-se sobre o Myo Armband, comentando seus principais conceitos, limitações e aplicações.
- No capítulo 4 detalha-se a implementação do sistema todo, desde os requerimentos iniciais até a calibração dos servomotores.
- No capítulo 5, apresentam-se os resultados e alguns comentários acerca do funcionamento do sistema implementado.
- Por fim, no capítulo 6, conclui-se o tema e são sugeridas algumas aplicações para o futuro.

2. REVISÃO BIBLIOGRÁFICA

2.1 INTERAÇÃO HOMEM-MÁQUINA (IHM)

Também conhecida como interação homem-computador (IHC), essa é uma definição generalista que inclui uma ampla variedade de interfaces capazes de conectar uma pessoa à uma máquina, sistema ou dispositivo. De forma abrangente, esse é um conceito que inclui todo o histórico evolucionário da tecnologia que contribuiu para o desenvolvimento do computador, dispositivo referência para exemplificar uma IHM. Pode-se partir do ábaco, instrumento mecânico de origem chinesa do século V A.C, passando por John Napier no século XVII com a régua de cálculo, e seguindo com contribuições de diversos matemáticos, físicos, engenheiros, como Leibniz, Jacquard, Boole e Babbage.

O computador pessoal (PC) a que estamos acostumados hoje em dia, tem sua história comumente dividida em quatro períodos. A primeira geração (1951-1959) pode ser exemplificada pelo ENIAC (Eletronic Numerical Integrator and Computer), que utilizava a tecnologia de tubos a vácuo nos seus circuitos, além de tambores magnéticos e cartões perfurados. Apesar de ser um dispositivo extremamente grande, com baixa capacidade de processamento e alto gasto energético (da ordem de 200 kW), o ENIAC é um marco na história da computação.

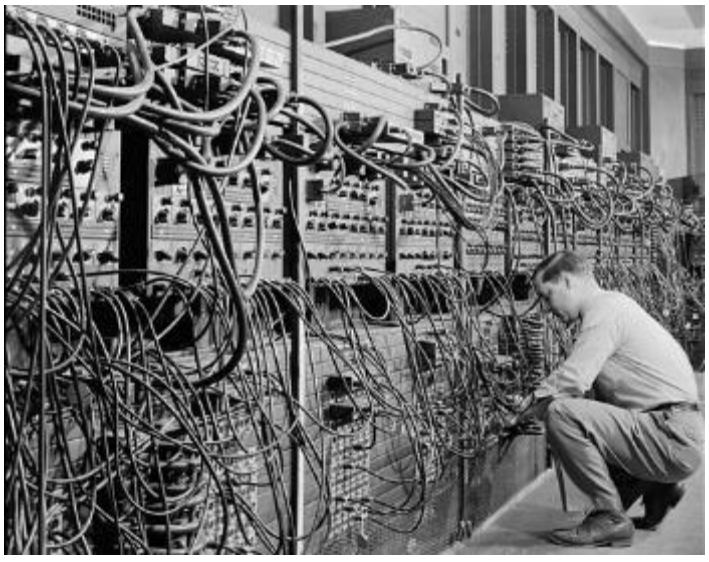

Figura 2.1 - Considerado como o primeiro computador eletrônico de propósito geral, o Integrador Numérico Eletrônico e o Computador (ENIAC) foi inicialmente comissionado para o uso na Segunda

Guerra Mundial, mas não foi concluído até um ano após o término da guerra. Instalado na Universidade da Pensilvânia, seus 40 racks separados de dois metros de altura e 18 mil tubos foram projetados para ajudar no cálculo de trajetórias balísticas. *Fonte: Revista Time; Créditos: Jerry Cooke/Corbis*

A segunda geração (1959 - 1965) é marcada pelo uso de transistores, substituindo os tubos a vácuo. Apesar de ainda expor os computadores a níveis prejudiciais de calor, os transistores tornaram os computadores menores, mais baratos e rápidos, utilizando menos energia elétrica. A linguagem binária era substituída por linguagens simbólicas, como *assembly*. Isso significava que os programadores poderiam criar instruções em palavras. Na mesma época, linguagens de programação de alto nível estavam sendo desenvolvidas (versões anteriores de COBOL e FORTRAN). As máquinas movidas a transistores foram os primeiros computadores a armazenar instruções em suas memórias, passando de tambores magnéticos para a tecnologia de núcleo magnético. As primeiras versões dessas máquinas foram desenvolvidas para a indústria de energia atômica.

A terceira geração (1965 - 1975) é marcada pela miniaturização dos transistores e utilização dos mesmos em chips de silício, nomeados de semicondutores. Essa nova tecnologia representou um enorme ganho em velocidade e eficiência para os computadores. Cartões perfurados foram substituídos por elementos como o monitor e o teclado, e pela primeira vez o usuário interagia com um sistema operacional, capaz de armazenar e reproduzir aplicações. A quarta geração (1975 - atualmente) marca o início da era dos microprocessadores, onde milhares de circuitos integrados eram armazenados em chips cada vez menores. Os microprocessadores extrapolaram o mundo dos computadores e começaram a ser utilizados em vários outros dispositivos do dia-a-dia. Além disso, contribuíram para o desenvolvimento de redes, uma vez que eles podiam se conectar e compartilhar informações. Outros grandes avanços da quarta geração foram a criação do mouse, desenvolvimento de interfaces gráficas do usuário (GUI) e mais recentemente os laptops e outros dispositivos móveis, como os smartphones.

Figura 2.2 - IBM PC, 1981 (esquerda). Primeiro computador de grande sucesso comercial para uso corporativo e pessoal. Liderou e estimulou a concorrência de outros grandes fabricantes, como a Apple, referência em inovação com seus dispositivos móveis como o iPhone e o MacBook Pro, 2018 (direita). *Fonte: Wikipédia e Amazon*

Para alguns autores, já vivemos a época da quinta geração computacional, com o advento da inteligência artificial, processamento paralelo e supercondutores. Outros defendem que o próximo grande marco está nas mãos da computação quântica, quando o mundo viverá uma nova era de interação com a tecnologia, com máquinas processando, aprendendo e se organizando de forma autônoma, alcançando uma velocidade de processamento computacional muito superior à empregada em dispositivos atuais.

De forma geral, percebe-se que o computador a que hoje estamos acostumados está enraizado em propósitos militares e científicos antes de se tornar um produto indispensável nos mais variados contextos. Rapidamente os computadores se tornaram desktops e notebooks, deixaram de ser uma invenção distante da realidade da maioria da população para tornar-se um elemento facilitador no exercício de diversas atividades cotidianas, como fazer compras, trabalhar e interagir com familiares e amigos. Estes são utilizados inclusive como inesgotável fonte de entretenimento, com o intuito de descansar, muitas vezes da fadiga gerada pelo demasiado uso desses mesmos dispositivos, mas com outro propósito. Sendo assim, a forma com que interagimos com essa máquina se faz extremamente relevante. Aplicações comuns como editores de texto, reprodutores e visualizadores de mídia são satisfeitas, pela maioria dos usuários, com a utilização de teclados, mouses, canetas, joysticks, elementos que são frutos de um árduo trabalho de engenheiros e desenvolvedores do passado. Ainda assim, usuários portadores de necessidades especiais não usufruem com tanta facilidade desse tipo de interface. Além disso, os softwares do futuro, cada vez mais sofisticados e demandantes, vem exigindo interações mais elaboradas, onde as ferramentas que temos hoje não mais serão suficientes para fornecer uma experiência precisa, intuitiva e agradável. Cada vez mais o usuário final,

comum, com limitado poder aquisitivo, tem se tornado elemento de maior relevância durante o processo de desenvolvimento das novas tecnologias.

2. 1. 1 IHM: DISPOSITIVOS

As interações humanas são ricas em gestos e expressões, corporais e faciais. Em sua pesquisa, Sharma *et al.* (2015) comentam que um objetivo de longo prazo buscado por cientistas que realizam interações homem-máquina, tem sido de migrar movimentos naturais que humanos utilizam para se comunicarem rumo às novas interfaces que surgem. Um segmento bastante promissor, e objeto de escopo deste trabalho, é o reconhecimento de gestos do conjunto braço-mão. Com ele, conseguimos apontar para um objeto ou pessoa, expressar um sentimento, mover, modificar e transformar um objeto, gesticular de forma a ilustrar idéias. Com a devida interface, de forma a garantir a correta interpretação pelo computador, são inúmeras as aplicações que se beneficiam a partir de um sinal de entrada do sistema mais fiel à intenção do usuário. Juntamente com o reconhecimento de voz, outro segmento largamente explorado nos últimos anos, temos maior flexibilidade e acurácia em aplicações de realidade virtual (VR), automóveis inteligentes, telemedicina, edição de textos, videogames, interfaces de multimídia, etc.

Encontra-se na literatura alguns dispositivos comuns às mais diversas aplicações. Cabreira *et al.* (2015) elaboraram uma interessante comparação entre três das plataformas mais utilizadas atualmente para reconhecimento de gestos: Myo Armband (Thalmic Labs), elemento de estudo deste projeto cuja descrição do seu funcionamento se encontra nos capítulos seguintes; Leap Motion (Leap Motion Inc.), aparelho que utiliza infravermelho e câmeras para captar movimentos precisos e simultâneos dos dedos dentro de centésimos de milímetros; e o Kinect Sensor (Microsoft), muito utilizado juntamente com o videogame Xbox, da mesma fabricante, que utiliza câmeras, sensores de profundidade e de movimento. Os autores compararam o desempenho das plataformas durante 250 aplicações diferentes, das mais diversas áreas, como entretenimento, educação, música, navegação. Os autores citam que o Myo Armband reconhece um número limitado de gestos, mas com maior acurácia durantes suas aplicações.

Figura 2.3 - Leap Motion (esquerda): Aparelho que transforma o espaço acima de sua superfície em uma interface 3D para as mãos, com um ângulo de captação de 150°. Já o Kinect (esquerda) contém uma câmera VGA de cores RGB (vermelho, verde e azul), um sensor de profundidade e um microfone multi-camadas, para diferenciar sons que se encontram próximo do dispositivo de outros do ambiente. *Fonte: Amazon*

Vários outros estudos acerca da aplicabilidade e performance do Myo Armband, como ferramenta principal de aquisição de sinais para controle de próteses e braços robóticos, compartilham resultados promissores. Méndez *et al.* (2017) elaboraram um estudo com o intuito de comparar a limitada largura de banda de aquisição de EMG do Myo (200 Hz), com um sistema de aquisição convencional, de espectro maior, visando verificar sua aptidão para sistemas de reconhecimento de gestos. Realizou-se um estudo cruzado com oito participantes sem amputações, realizando nove gestos com as mãos. Seis características foram extraídas e classificadas utilizando a técnica de Análise Discriminante Linear (LDA). Os autores mostraram uma média de erro de classificação de 5,82 ± 3,63% para o método convencional e 9,86 ± 8,05% para o Myo, sem diferença significativa (P = 0,056), indicando que o Myo, apesar da largura de banda restrita, pode ser adequado para tais aplicações.

Em seu trabalho, Benalcázar *et al.* (2017) propõem um modelo para reconhecimento de gestos em tempo real, tendo o Myo como ferramenta de aquisição dos sinais eletromiográficos. O modelo proposto pelo autor é baseado no algoritmo de K-vizinhos mais próximos (k-Nearest Neighbours, k-NN) e outros algoritmos dinâmicos de distorção de tempo. Ao avaliar, medir e comparar a precisão de seu sistema em reconhecer cinco classes de gestos, os autores obtiveram um desempenho melhor do que o sistema do Myo, 86% contra 83%, indicando que há espaço para otimização no que se refere ao processamento dos dados adquiridos pelos sensores.

O Myo tem sido utilizado também como uma das plataformas de controle do TedCube, um sistema de controle por comandos de voz e de gestos. Caballero *et al.* (2015), destacam que durante determinadas cirurgias e procedimentos, é comum que os profissionais da saúde sejam auxiliados por imagens e projeções, muitas vezes necessitando de assistentes para girar figuras, aumentar e diminuir o zoom de determinada gravura. Nessa aplicação, o Myo pode ajudar o médico a manipular a interface do sistema sem a necessidade de tocar os equipamentos, com movimentos curtos e precisos, reduzindo a interação com equipamentos, diminuindo o risco de infecção e acidentes de trabalho. Além disso, procedimentos poderiam se tornar mais rápidos, diminuindo também custos relacionados ao uso de salas de cirurgia.

Outra interessante aplicação médica do Myo se apresenta na batalha contra a Doença de Parkinson, condição que afeta dez milhões de pessoas no mundo todo, de acordo com a Parkinson's Foundation. Exames como a Tomografia Computadorizada por Emissão de Pósitrons (PET-CT), Ressonância Magnética (MRI) e DatScan, são os principais procedimentos utilizados para o diagnóstico da doença, mas costumam ser caros e realizados apenas em clínicas especializadas. Nesse sentido, Song *et al.* (2014) utilizaram uma outra abordagem. Uma redução nos tremores, sintoma principal da doença, após a administração do fármaco Levodopa, é um forte indicativo do diagnóstico de Parkinson. Os autores criaram a aplicação Tremedic, que utiliza o Myo Armband como ferramenta para detectar variações na magnitude dos tremores e determinar se o paciente responde bem ao fármaco, confirmando o diagnóstico. Além disso, o Tremedic oferece monitoramento contínuo dos sintomas do paciente no seu dia-a-dia, armazenando e possivelmente exportando esses dados para auxiliar médicos e fisioterapeutas durante o tratamento.

O reconhecimento de gestos tem várias aplicações em campos médicos e de engenharia. O problema do gesto com a mão reconhecimento consiste em identificar, a qualquer momento, um dado gesto realizado pela mão. Neste trabalho, propomos um novo modelo para reconhecimento de gestos em tempo real. A entrada de este modelo é a eletromiografia de superfície medida pelo sensor comercial do Myo Armband colocada no antebraço. A saída é o rótulo do gesto executado pelo usuário em a qualquer momento. O modelo proposto é baseado no k-vizinho mais próximo e algoritmos dinâmicos de distorção de tempo. Este modelo pode aprenda a reconhecer qualquer gesto da mão. Avaliar o desempenho do nosso modelo, medimos e comparamos precisão em reconhecer 5 classes de gestos para a precisão de o

sistema proprietário do Myo Armband. Como resultado disso avaliação, determinamos que nosso modelo tem melhor desempenho (86% de precisão) do que o sistema Myo (83%).

2. 2 EMG

O sinal eletromiográfico (EMG) é um sinal adquirido pela resposta elétrica gerada pelos músculos durante a contração/relaxamento, atividades que representam o processo de controle neuromuscular vindo do cérebro. Para ativar determinado grupo muscular, o cérebro manda sinais de excitação pelo sistema nervoso central (SNC). Os músculos são inervados em grupos denominados como “Unidades Motoras” (FATTAH *et al.*, 2012) como ilustra a figura abaixo.

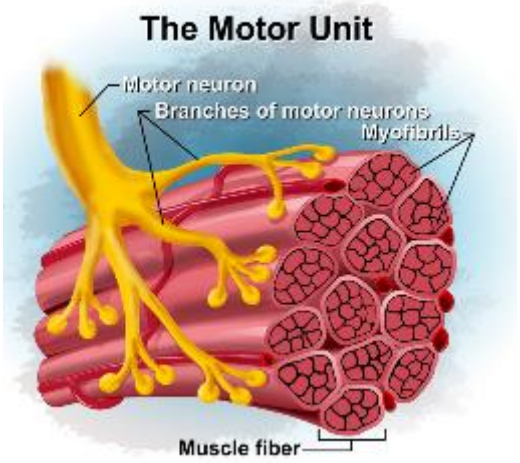

Figura 2.4 - Unidade Motora. *Fonte: http://chadwaterbury.com*

Uma unidade motora é composta de um neurônio motor alfa e todas as fibras musculares por ele inervadas. Uma fibra muscular contrai quando um potencial de ação do nervo motor que a supre alcança um limite de despolarização. Essa despolarização gera um campo magnético e seu potencial é medido como tensão. A despolarização, que se espalha pela membrana do músculo, é um potencial de ação do músculo. O potencial de ação da unidade motora é a soma espacial e temporal dos potenciais de ação musculares individuais para todas as fibras de uma única unidade motora. Sendo assim, um sinal eletromiográfico é a soma algébrica dos potenciais de ação de unidades motoras presentes na área de leitura do eletrodo que se usa. Essa área quase sempre irá incluir mais de uma unidade motora, dado que fibras de diferentes unidades motoras estão interligadas por todo o músculo. Qualquer parte de um músculo pode conter fibras que derivam de 20-50 unidades motoras. Uma única unidade motora pode ter entre

três e duas mil fibras musculares. Músculos que controlam movimentos mais finos tem menos fibras musculares. Há uma espécie de organização hierárquica durante a contração muscular, onde unidades motoras com menos fibras musculares são ativadas primeiro, seguidas pelas unidades motoras que possuem fibras maiores. (RASH *et al.,* 2002).

2. 2. 1 TIPOS DE EMG:

A aquisição de sinais mioelétricos é realizada a partir da acoplagem de eletrodos junto ao corpo do paciente. Pode-se dizer que existem dois grandes grupos de eletrodos: Intramusculares e superficiais.

- **Eletrodos Intramusculares:**

Os eletrodos intramusculares (Figura 3.1) podem ser agulhas ou fios rígidos. Esse tipo de aquisição garante uma maior precisão na seleção do músculo em estudo, diminuindo-se assim a interferência de outros músculos, fibras, unidades motoras. Além disso, pode-se alcançar grupos musculares mais específicos e obter sinais de maior amplitude. Esse tipo de eletrodo é comum em aplicações clínicas. Sua desvantagem é ser um componente invasivo e, dependendo do tipo de exame, pode causar um desconforto considerável ao paciente (RASH *et al.,* 2002).

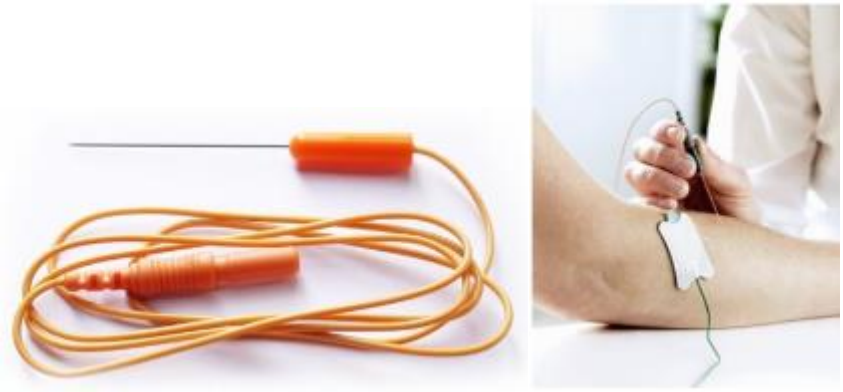

Figura 3.1 - Eletrodos Intramusculares. *Fonte: Kandel*

.

- **Eletrodos Superficiais:**

Os eletrodos superficiais (Figura 3.2) são componentes capazes de promover a aquisição de sinais mioelétricos de forma não-invasiva. Esse tipo de eletrodo é capaz de formar um equilíbrio químico entre a superfície de detecção e a pele por condução eletrolítica, fazendo que com a corrente flua para dentro do eletrodo. São simples de usar, não requerem uma restrita supervisão médica e nem certificação. Tem sido utilizados em estudos de controle motor, gravações neuromusculares, avaliações médicas voltadas para o esporte (MASSO *et al.,* 2010) e com crianças que tem problemas com agulhas. Além disso, tem sido cada vez mais usado para detectar atividades musculares com o intuito de desenvolver aplicações voltadas para o auxílio de pessoas com problemas motores e doenças que comprometam o controle muscular (JAMAL *et al.*, 2012).

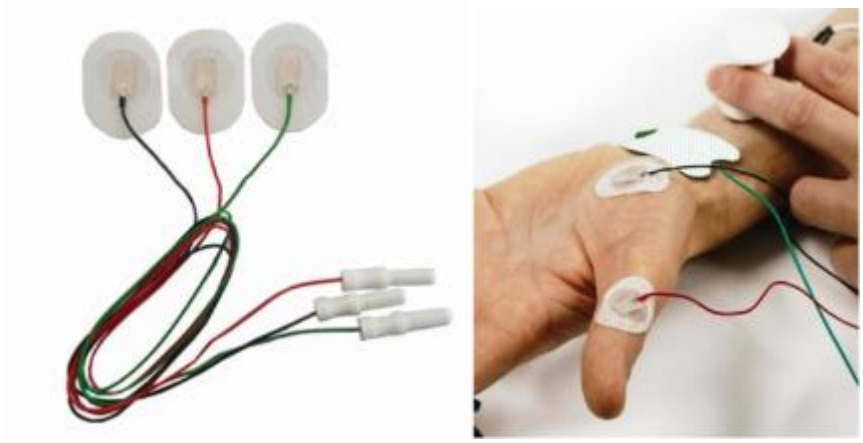

Figura 3.2 - Eletrodos Superficiais. *Fonte: Kandel*

Independentemente do método de medição, os potenciais do tecido superficial humano envolvem tensões de amplitude bastante pequenas, que podem variar de 1μV á 100mV. Os sinais podem ser perturbados por fontes de alta impedância, ruídos de diversas origens (linhas de transmissão, campos eletromagnéticos, radiofrequência, etc.), entre outros tipos de interferência (pele, eletrodos, movimento, etc.), como discutido em Jamaluddin *et al.* 2014. Os parâmetros dos eletrodos, por exemplo, contribuem significativamente para a SNR (razão sinal-ruído) do sistema em questão. Em Paraskevopoulou *et al.* (2015), os autores exemplificam como eletrodos influenciam também no offset DC, durante sua interface com a pele, sendo necessário rever a dinâmica do acoplamento AC adotado para combater uma possível saturação do sinal. Em seu amplo estudo acerca de biopotenciais e medições eletrofisiológicas, Nitish *et*

al. (1999) ilustram que eletrodos com configurações tripolares ou multipolares podem ajudar a resolver não só problemas com SNR, mas também combater a interferência derivada das linhas de transmissão, buscando uma razão de rejeição de modo-comum (CMRR) razoável o bastante para ter-se confiabilidade e precisão nos valores de leitura dos sensores.

De forma bastante simplificada, o circuito de aquisição de sinais eletromiográficos costumam compartilhar alguns princípios básicos: alta amplificação do sinal, impedância de entrada, e rejeição da interferência elétrica. Configurações que contam com o uso de amplificadores de bio-potenciais como o INA128p, seguidos de filtros passa-faixas para garantir a banda de frequência desejada juntamente com o combate à interferências indesejadas, e finalizados por conversores analógico-digitais, que podem funcionar também como interfaces gráficas, como o NI USB-6009 ou NI DAQ, são extremamente comuns nas mais variadas aplicações que envolvem a aquisição de sinais eletromiográficos de superfície (SEMGs). Alguns exemplos de projetos que seguem essas características de configuração podem ser vistos em trabalhos como os de Fattah *et al.* (2012), que buscam identificar a Doença do Neurônio Motor comparando características do sinal de EMG no domínio da frequência e do tempo; Biswal *et al.* (2014) propõem o desenvolvimento de um dispositivo universal para leitura de biopotenciais de diferentes tipos (eletrocardiograma, eletromiograma e eletrooculograma) para facilitar avaliações médicas em regiões mais afastadas dos grandes centros; Em Moslem *et al.* (2011), os autores utilizam técnicas de aprendizado de máquinas que mixam dados de matrizes de eletrodos para melhorar o desempenho da eletromiografia de útero, a electro histerografia (EHG), que auxiliam no controle da gestação.

2. 3 UNIDADES DE MEDIDAS INERCIAIS

As Unidades de Medidas Inerciais (IMUs) são dispositivos capazes de medir grandezas como aceleração linear, taxa angular, campo magnético, força, utilizando sensores específicos, comumente por meio da integração e combinação de acelerômetros, giroscópios e magnetômetros. De acordo com Demirbas *et al.*, 2012, "*um acelerômetro é capaz de medir a aceleração linear em torno do seu eixo de sensibilidade, possibilitando então o cálculo da velocidade e posição do corpo em questão; enquanto isso, o giroscópio mede a taxa angular sobre o seu eixo de sensibilidade, dado que pode ser utilizado para manter a orientação no espaço".* Esses sensores, quando integrados com magnetômetros, que são capazes de medir a

intensidade, direção e sentido do campo magnético em sua proximidade, possibilitam a determinação da orientação relativa ao eixo vertical, uma vez que esse, sem perturbações, refere-se ao campo magnético da terra (BRUNNER *et al., 2015).* O Myo Armband reúne os três tipos de sensores acima citados em seu circuito para maximizar a eficiência no reconhecimento de gestos e posições.

Com o avanço da eletrônica/microeletrônica e com melhores algoritmos de calibração e previsão de erros, esses tipos de sensores se tornaram extremamente pequenos, com baixo gasto energético e com baixo custo de prototipagem. Com isso, tornou-se possível a o desenvolvimento de dispositivos de tamanho reduzido que levam esses sensores de alta precisão em sua configuração, como é o caso de celulares móveis por exemplo.

IMUs tem sido utilizadas juntamente com sistemas de navegação como o GPS, como ilustra King *et al., 1998*. No desenvolvimento de exoesqueletos, as IMUs podem ser empregadas em conjuntura com outros diversos sensores, se estabelecendo como uma alternativa ao EMG durante a captação de determinados sinais (JUNG *et al., 2015).*

Sensores inerciais e magnéticos apresentam determinadas vantagens para aplicações como a implementada neste trabalho. Eles não apresentam uma latência inerente associada ao seu funcionamento, responsabilizando quaisquer tipos de atraso aos circuitos de comunicação e processamento. Além disso, em comparação a outros sensores dos tipos eletromagnéticos, acústico e óticos, IMUs não necessitam de uma fonte de emissão de sinais para rastrear os objetos, o que é uma vantagem considerável quando trabalha-se em ambientes não totalmente controlados. Por outro lado, autores também relatam algumas desvantagens relacionadas as IMUs. Fourati *et al., 2014,* comentam que uma vez que os acelerômetros, por exemplo, medem a soma da aceleração linear com a aceleração gravitacional, não costuma haver medições muito distorcidas em situações de quase-estabilidade, mas durante uma medição de maior dinâmica, não é trivial a dissociação das duas quantidades físicas, o que pode tornar complicada a medida precisa do sensor. Moreira *et al.,* 2016, discorrem sobre outro tipo de erro comum a esse tipo de sensor, a deriva (drift). Uma vez que as medições atuais levam em conta leituras passadas, qualquer espécie de erro, por menor que seja, será propagado e acumulado em cada ciclo de cálculo, levando a uma imprecisão na posição estimada. De acordo com Veltink *et al.,* 2003, magnetômetros já não são tão susceptíveis aos erros de deriva, mas são influenciados por materiais ferromagnéticos que possam estar no ambiente ao redor, distorcendo a medida da orientação.

3. MYO ARMBAND

O Myo Armband (Thalmic Labs 2016) é um dispositivo destinado a ser utilizado no antebraço, sendo ele capaz de detectar gestos do punho e da mão, além de movimentos do braço, utilizando 8 eletrodos eletromiográficos de aço inoxidável, combinados com uma unidade de medida inercial (IMU). A IMU contém um acelerômetro de três eixos, um giroscópio de três eixos e um magnetômetro de três eixos. Além disso, o Myo contém um processador ARM Cortex M4 e envia os dados de leitura dos sensores utilizando um módulo bluetooth de baixa energia (BLE).

Figura 3.1: Myo Armband. *Fonte: Thalmic Labs*

3. 1 FUNCIONAMENTO

De acordo com a Thalmic Labs, o Myo Armband fornece dois tipos de dados aos seus usuários: dados espaciais e dados gestuais. Esses dados são visualizados no aplicativo Myo Connect, para computador disponível no site da fabricante (www.myo.com). Os dados espaciais são referentes à orientação e movimento do braço do usuário, obtidos pela IMU. Incluem-se dados de orientação que indicam em qual direção o Myo Armband está apontada; dados de aceleração brutos que representa a aceleração com que o Myo Armband está se movendo em qualquer momento, na forma de vetor tridimensional; e dados de velocidade angular fornecidos pelo giroscópio, também no formato de um vetor.

Os dados brutos do acelerômetro medem a aceleração linear do Myo Armband, levando em consideração a Constante Gravitacional Universal, *G*, como mostrado (3.1).

$$\boldsymbol{G} = 6{,}674184\ x\ 10^{-11}\ \ m^3 kg^{-1} s^{-2} \tag{3.1}$$

Considera-se também a Aceleração Gravitacional, *g*, conforme (3.2)

$$\boldsymbol{g} = 9{,}81\ \ m/s^2 \tag{3.2}$$

As equações acima se relacionam por meio de (3.3), onde *M* é a massa da terra e *r,* o raio. (THALMIC LABS, 2016). A consequência dessa medida é que, quando o usuário está em estado estacionário, um valor de 1 deve ser observado na direção vertical, devido à gravidade da Terra.

$$\boldsymbol{g} = \frac{G\ M}{r^2} \tag{3.3}$$

Os dados do giroscópio medem a aceleração angular do Myo Armband, com uma limitação de aproximadamente *16 rad / s*, de acordo com a fabricante. Weili *et al.,* 2014, argumentam que, embora cada componente dos dados por si só não seja de grande valia na maioria dos cenários, seu efeito combinado pelo cálculo da raiz quadrada da soma dos quadrados permite obter a magnitude da aceleração linear ou angular do braço, que, como afirma o autor, "*são indicadores muito eficazes da intensidade do braço movimento, que por sua vez contém informações emocionais ou rítmicas da performance* ". Os dados de orientação são apresentados em duas formas diferentes: na forma de quatérnions e os ângulos de Euler, ou ângulos de rolagem, arfagem e guinada, comumente conhecidos também por *"roll, pitch e yaw"*.

Os dados de orientação são calculados usando os dados brutos do acelerômetro e giroscópio da IMU. No entanto, a fabricante menciona que, a fim de obter dados de posição, a integração dupla dos dados de entrada seria necessária. Tal método é sabidamente responsável por inserir uma quantidade significativa de erro na medição, e, portanto, os desenvolvedores do Myo Armband escolheram não oferecer tais dados como uma saída padrão, alegando que "*o Myo Armband é mais adequado para obter as orientações relativas dos braços, em vez da posição absoluta*".

De acordo com a Thalmic Labs, o reconhecimento de gestos do Myo fornece informação suficiente para o controle de diversas aplicações disponibilizadas na loja da fabricante, o que é verdade. A braçadeira fornece os dados gestuais sob a forma de uma das várias poses predefinidas, que representam uma configuração da mão do usuário. Os gestos pré-determinados capazes de serem detectados pelo dispositivo são:

1. Duplo toque entre dedo médio e polegar (*Double Tap);*
2. Flexão do punho ou da mão (*Wave Left);*
3. Hiperextensão do punho ou da mão (*Wave Right);*
4. Extensão dos dedos (*Spread Fingers);*
5. Punho cerrado (*Make Fist).*

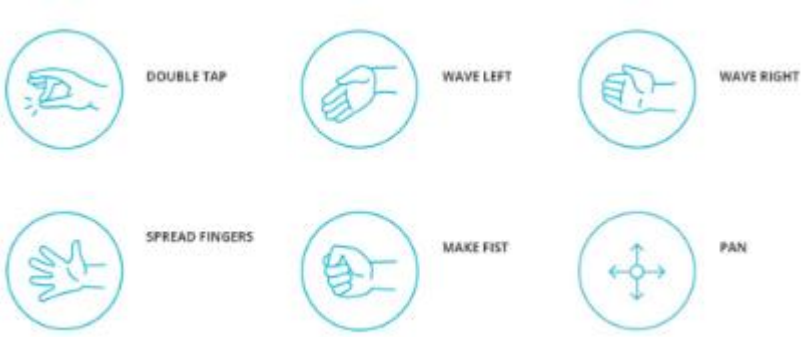

Figura 3.2 – Ilustração das poses predefinidas do Myo Armband. *Fonte: Thalmic Labs*

3. 2 LIMITAÇÕES

Os gestos reconhecidos pelo Myo Armband são resultados da combinação da leitura de dados EMG dos 8 sensores montados no dispositivo juntamente com as informações fornecidas pela IMU, treinados por redes neurais que não são fornecidas pela fabricante. Os dados puros (*raw data)* dos sensores EMG foram disponibilizados na loja da fabricante alguns meses após o lançamento do produto, e certamente são úteis para elaborar estudos mais aprofundados e possivelmente reconhecer novos gestos além dos predefinidos. Entretanto, utilizar os dados puros dos sensores EMG requer uma apurada técnica de filtro, processamento e treinamento via algoritmos de aprendizado máquinas, como os citados no capítulo 2, o que pode não ser uma tarefa trivial (BERNHARDT, 2015).

Os dados da IMU têm uma frequência de amostragem de 50 Hz e a EMG de 200 Hz. No entanto Jensenius *et al.,* 2015 demonstraram que, ao avaliar os dados do sensor fornecidos pelo Myo Armband, o fluxo de dados tinha uma taxa de quadros mais baixa do que os 50 Hz especificados. De acordo com a fabricante, esta questão pode ser devido ao uso do Myo em ambientes próximos a fontes comuns de interferência, como cabos de alimentação, fornos de micro-ondas, lâmpadas fluorescentes, câmeras de vídeo sem fio e telefones sem fio. Esses elementos podem causar a perda de pacotes nos dados transmitidos através de Bluetooth. Weili *et al.,* 2014, afirmam que *"Um gesto calculado pode não indicar lealmente o gesto real da mão"*. Eles argumentam que o gesto da mão é calculado a partir dos dados EMG medidos na pele do antebraço, sendo um efeito colateral do movimento de um outro grupo muscular, existindo a possibilidade de que um gesto calculado não indique lealmente o gesto realizado pela mão. Além disso, completam que quando forças externas são aplicadas aos músculos, ou há alguma outra interferência, como roupas apertadas, a precisão da medição pode ser consideravelmente degradada, comprometendo a confiabilidade do sistema.

3. 3 APLICAÇÕES

Apesar das ressalvas acerca de sua funcionalidade, o Myo Armband se mostra um dispositivo bastante interessante para as mais variadas aplicações:

- **Apresentações e controle de aplicações web**: O aplicativo Myo Connect, fornecido pela fabricante, já apresenta um modo de apresentação, que pode ser integrado com feramentas comuns como o Microsoft Power Point e Keynote, tendo uma experiência livre da necessidade de estar perto do computador ou mouse para passar slides. Além disso, é possível controlar outros aplicativos que apresentam integração com o Myo, como trocar de músicas durante o uso do programa de streaming Spotify.
- **Entretenimento:** O kit de desenvolvimento de software (SDK) fornecido pela Thalmic Labs, fabricante do Myo, permite, de forma relativamente simples, uma ampla integração com as mais diversas plataformas de entretenimento disponíveis atualmente, como dispositivos móveis (celulares e tablets), consoles, dispositivos de realidade virtual. Vários segmentos de drones disponibilizam seu código online e podem ser integrados ao Myo de forma a substituir joysticks.

Fazendo uso dos dados espaciais e de EMG captados pelo Myo, é possível entregar ao usuário uma experiência diferenciada do sensoriamento por toque ou uso de botões, comumente visto em jogos e em outras inúmeras aplicações. Um exemplo que está dentro desse novo espectro de possibilidades que dispositivos como o Myo fornecem pode ser visto na imagem (3.3). O renomado DJ Armin Van Buuren, em parceria com a Thalmic Labs, utilizou um Myo Armband em cada braço durante uma apresentação em Ibiza para controlar um grande anel de luzes.

Figura 3.3 – DJ Armin Van Buuren utilizando dois Myo Armbands para controle de parte da iluminação durante show em Ibiza, Espanha. *Fonte: Thalmic Labs*

- **Engenharia Biomédica:** Pela sua configuração não invasiva, sem fios e com uma boa instrumentação embutida, o Myo aparece como uma relevante alternativa a procedimentos e sistemas de eletromiografia mais robustos, que costumam trazer um valor agregado muitas vezes limitante a projetos de código aberto. Com o devido processamento e treinamento, é possível ter resultados consistentes em aplicações clínicas, como é o caso de Johnny Matheny (figura 3.4) que perdeu parte do braço esquerdo em sua luta contra o câncer, mas graças a um projeto fruto da parceria entre a Agência de Projetos de Pesquisa Avançada de Defesa Norte-Americano (DARPA) e o Laboratório de Física Aplica da Johns Hopkins University, recuperou boa parte dos movimentos do braço por via de

uma prótese que utilizava dois Myo Armbands como sensores principais. Johnny passou por uma cirurgia para rearranjar os nervos do braço e conectar a prótese diretamente a estrutura óssea de seu braço. Além do bom nível de atuação, o projeto visa fechar a malha de controle, acoplando sensores que irão fornecer dados do ambiente ao usuário, como a textura de objetos e também a temperatura. Esse projeto em específico foi a motivação inicial para o modelo de implementação que será apresentado a seguir.

Figura 3.4 – Johnny Matheny (direita) utilizando dois Myo Armbands para controlar uma prótese acoplada diretamente na estrutura óssea de seu braço. O projeto é uma parceria entre o Governo Norte-Americano (DARPA) e a Universidade Johns Hopkins (EUA). *Fonte: Wareable.com*

4. IMPLEMENTAÇÃO

4.1 PROPOSTA

O objetivo do trabalho, como comentado anteriormente, é desenvolver uma interface simples, amigável e eficiente, utilizando o Myo Armband como sensor para controlar um braço robótico. Para tal, utiliza-se elementos não só capazes de promover a integração entre sensoriamento e atuação, mas fazê-lo de tal forma com que o leitor possa replicar a configuração aqui apresentada em outros projetos, sem comprometer a controlabilidade do sistema e com um baixo custo de implementação do mesmo. Além disso, reforça-se que esse é um projeto inicial, cujos elementos foram escolhidos para habilitar diversos aperfeiçoamentos em projetos futuros.

O diagrama abaixo ilustra o fluxo de trabalho do sistema, apresentando os componentes principais. O Myo Armband é acoplado no antebraço do usuário, captando dados eletromiográficos e espaciais. Esses dados são enviados via bluetooth para um computador que está executando uma aplicação na ferramenta Unity 3D. A Unity apresenta os dados lidos pelo sensor utilizando sua Interface Gráfica do Usuário (GUI), comumente utilizada para desenvolver jogos e aplicações que requerem alta qualidade de imagens. Esses dados são enviados ao microcontrolador Arduino, que é capaz de atuar de acordo com a programação feita pelo usuário em função dos gestos e movimentos performados com o sensor acoplado em seu corpo. Por fim, comandos são enviados do Arduino para controlar os servo-motores de um robô com 6 graus de liberdade que simula um braço humano. Os detalhes de cada etapa do processo são detalhadamente explicitados nas seções a seguir.

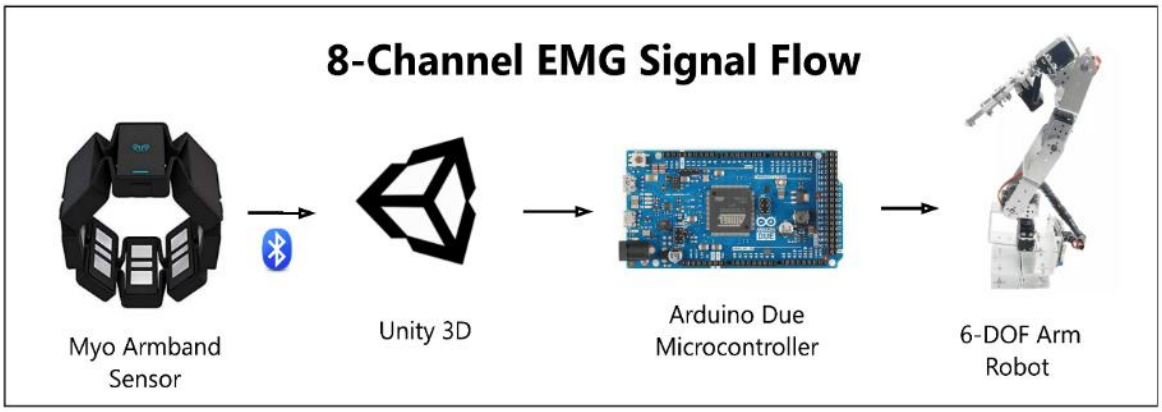

Diagrama 1. Fluxo do sinal de EMG do sistema de controle do braço robótico.

4.2 REQUERIMENTOS

Lista-se abaixo os componentes e procedimentos necessários para completa integração dos elementos utilizados. É possível, entretanto, utilizar variações nos modelos e ou métodos escolhidos sem comprometer o funcionamento do sistema no geral.

4.2.1 SOFTWARE

São utilizados três softwares principais para a implementação do projeto. O primeiro é o Myo Connect, fornecido de forma gratuita no site da Thalmic Labs. Por ele é possível verificar o status de conexão do módulo bluetooth, navegar entre as aplicações sincronizadas ao dispositivo, verificar o nível de bateria e desligar o sensor. A execução do aplicativo durante todo o tempo é imprescindível para o sistema, uma vez que o sucesso da conexão bluetooth é constantemente verificada.

O segundo software é a Unity 3D. Trata-se de um motor de jogo, ou ferramenta utilizada para a criação das mais diversas animações. É uma plataforma que não só integra soluções da comunidade "*open source"* mas é a escolha de desenvolvedores responsáveis pelos produtos grandes clientes finais, como Apple, Sony e Microsoft. A escolha dessa ferramenta se deve ao fato de que a Thalmic Labs desenvolveu um pacote do Myo Armband para a Unity 3D, de forma que é possível fazer uma integração bastante visual e criar diversos tipos de aplicações. Para isso, é necessário instalar o kit de desenvolvimento de software do Myo, (Myo SDK) no site da Thalmic e instalar também a Unity 3D, que está disponível para Windows, Mac e Linux. Dentre algumas linguagens compatíveis com a Unity 3D, utiliza-se nesse projeto o C#, aproveitando a a mesma linguagem em que foi escrito o pacote do Myo Armband, evitando-se conversões e possíveis perdas de funcionalidade.

4.2.2 HARDWARE

Para a etapa de atuação junto ao braço robótico, empregou-se a escolha do microcontrolador Arduino DUE. Trata-se de uma plataforma de prototipagem eletrônica bastante difundida na comunidade de software livre, com bastante fóruns de dúvidas que facilita o desenvolvimento de projetos em quaisquer níveis de complexidade. As placas de Arduino variam de preço e apresentam pequenas diferenças em funcionalidades, mas no geral são de fácil acesso, com baixo custo e programação simples. É necessário fazer o download do ambiente de desenvolvimento integrado (Arduino IDE) para fazer upload dos códigos à placa.

Um outro elemento importante nesse projeto é o driver para controle de até 16 servomotores via PWM (Modulação da Largura de Pulsos), da Adafruit, o PCA9685, ilustrado abaixo.

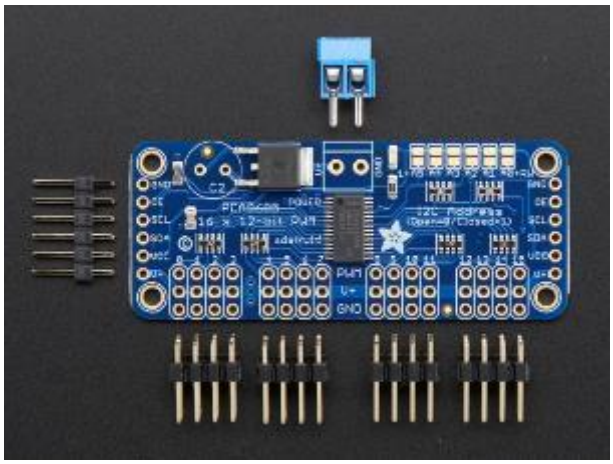

Figura 4.1 – Driver de 16 canais PWM da Adafruit, PCA9685. *Fonte: Adafruit*

Servos são controlados por meio de pulsos de largura variável (PWM). Os parâmetros são o pulso mínimo, pulso máximo e a taxa de repetição. Uma vez que o braço robótico tem 6 graus de liberdade, ou 6 servos, seriam necessárias muitas saídas lógicas do microcontrolador para o perfeito controle do robô. Por meio do protocolo de comunicação I2C (Inter-Integrated Circuit), tem-se uma espécie de barramento, trabalhando no modelo mestre-escravo, onde todos os dispositivos estão conectados ao "fio de controle", como ilustra a figura 4.2. Com isso, utilizando as portas do SDA (Serial Data), que envia informações sobre os pulsos e SCL (Serial Clock) que cuida de temporização do sistema, consegue-se reduzir bastante a quantidade de fios necessários. A Adafruit disponibiliza também uma biblioteca específica para controle de servos via Arduino, facilitando ainda mais a programação dos pulsos.

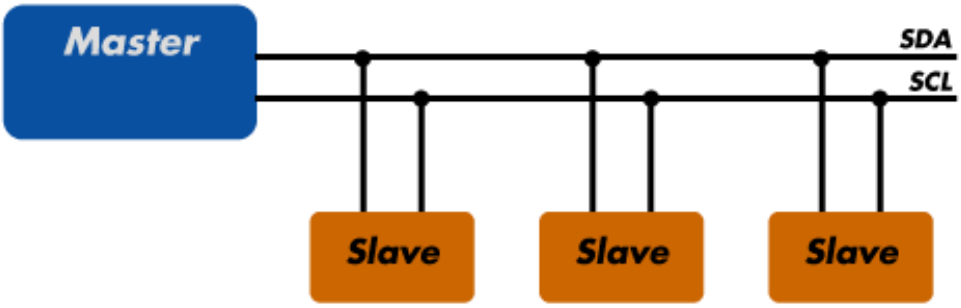

Figura 4.2 – Dispositivo conectado ao barramento atuando como Mestre (Arduino) e dispositivos atuando como escravo (servos). *Fonte: http:Arduinobr.com.br*

Por fim, o elemento final do sistema, um braço robótico com 6 servomotores (figura 4.3). O modelo aqui utilizado foi fornecido pela MEKHOS, mas trata-se de uma configuração bastante difundida no mundo. Os kits costumam vir desmontados e requerem cuidado em sua montagem, pela quantidade de porcas e parafusos, e pela possibilidade de danificação dos motores em casos de posicionamentos que obstruam seu correto funcionamento, forçando demasiadamente suas engrenagens.

Figura 4.3 – Braço robótico com 6 graus de liberdade. *Fonte: Alibaba*

4.3 INTERFACE: MYO - UNITY

O aplicativo Myo Connect, fornecido pela fabricante durante a instalação do Myo, garante a comunicação bluetooth e fornece as aplicações “de fábrica” fornecidas pelos desenvolvedores do sensor. Com ele, é possível controlar slides, diminuir e aumentar o volume do sistema, interagir de forma limitada com determinados aplicativos. Para acessar as linhas de código que dão acesso aos dados dos sensores é necessário utilizar o kit de desenvolvimento do Myo, como dito na seção de requerimentos. Uma vez instalado o kit, importa-se o pacote MyoUnity, para a Unity 3D, como ilustrado abaixo na figura abaixo. Com isso, torna-se disponível para visualização e edição os scripts que realizam a captação, processamento e interface do Myo Armband.

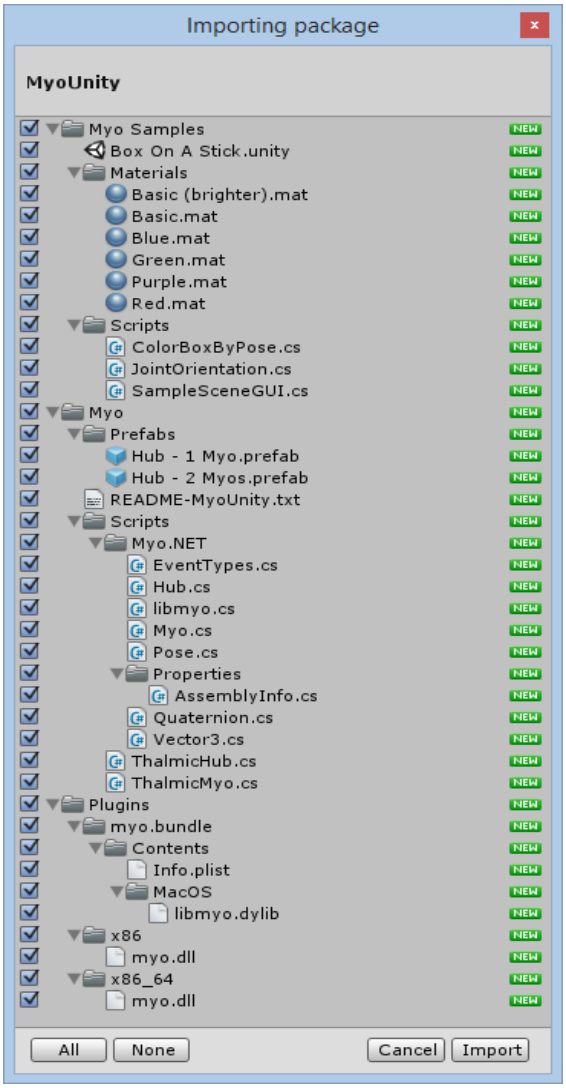

Figura 4.4 – Pacote de dados do Myo importados para Unity. *Fonte: Developerblog.myo*

É interessante que o leitor navegue pelos scripts para ter uma visão geral do processo que acontece dentro do Myo Armband, mas não é necessário a edição de todos eles. Nas seções abaixo, busca-se detalhar os elementos mais importantes do pacote.

4.3.1 QUATÉRNIONS

Existem várias maneiras de se definir a orientação de objetos computacionalmente. No caso do Myo Armband, essa orientação é armazenada em variáveis nomeadas quatérnions. Para discursar sobre esta classe de números, utiliza-se o estudo de FRANQUEIRA (1993), acerca dos quatérnions e sua aplicação em robótica. O autor cita que comumente, em um espaço euclidiano tridimensional, 6 parâmetros são necessários para especificar a localização e a orientação de uma extremidade ativa de um robô. Geralmente, a localização de um ponto, expressa em um sistema de coordenadas cartesianas, é dada por um vetor de posição (3 graus)

e a sua orientação é especificada pelos ângulos de Euler (outros 3 graus). Estes últimos são comumente usados, embora não sejam os mais eficientes para aplicações em robótica. Isto se deve à necessidade de se manipular uma matriz de rotação, o que exige um maior tempo de processamento e também a problemas de singularidade, como é o caso do "*Gimbal Lock",* quanto utiliza-se os ângulos de Euler. Dada as aplicações cada vez mais demandantes que tem-se desenvolvido, uma representação alternativa, como é o caso dos quatérnions, podem ser a solução.

Essa classe foi proposta por William Rowan Hamilton, em 1843, enquanto procurava estender o plano complexo ao espaço de três dimensões.

Seguindo o desenvolvimento exposto em FRANQUEIRA (2013), um quatérnio pode ser definido como:

$$\boldsymbol{Q} = q_r + q_x^{\bar{\imath}} + q_y^{\bar{\jmath}} + q_z^{\bar{k}} = q_r + \bar{v} \tag{4.3}$$

Onde q_r, q_x, q_y, q_z são escalares, $\bar{v}$ é um vetor, e $\bar{\imath}, \bar{\jmath}, \bar{k}$ são três vetores unitários de um sistema de coordenadas ortogonal (x, y, z).

O conjugado Q^*é definido como:

$$\boldsymbol{Q}^* = q_r - q_x^{\bar{\imath}} - q_y^{\bar{\jmath}} - q_z^{\bar{k}} = q_r - \bar{v} \tag{4.4}$$

Um quatérnio, que tenha apenas parte vetorial, pode representar uma posição no espaço tridimensional, sendo reescrito como:

$$\bar{\boldsymbol{r}} = q_x^{\bar{\imath}} + q_y^{\bar{\jmath}} + q_z^{\bar{k}} \tag{4.5}$$

O quatérnio unitário tem norma igual a:

$$\|\boldsymbol{Q}\| = q_x^2 + q_y^2 + q_z^2 = 1 \tag{4.6}$$

E também pode ser reescrito na forma:

$$\boldsymbol{Q} = \cos\frac{\theta}{2} + \bar{d}\sin\frac{\theta}{2} \tag{4.7}$$

Onde θ é o ângulo de rotação, ou quatérnio de rotação, e $\bar{d}$ é a direção do eixo de rotação. Se a rotação for em sentido contrário, a parte escalar permanece a mesma, mas a parte vetorial terá o sinal trocado, logo:

$$\boldsymbol{Q} = -(\cos\frac{\theta}{2} + \bar{d}\sin\frac{\theta}{2}) \tag{4.8}$$

A equação genérica para o recíproco de um quatérnio é:

$$\boldsymbol{Q^{-1}} = \frac{Q^*}{\|Q\|} \tag{4.9}$$

Como todos os efeitos de magnitude ficam distribuídos pela normalização, qualquer multiplicação de um quatérnio por um escalar, dará a mesma rotação. Este comportamento também é observado no caso de coordenadas homogêneas, onde o produto leva ao mesmo ponto não-homogêneo. Se um quatérnio de posição realizar uma rotação definida por outro, ele se transformará em um novo vetor de posição.

$$\overline{\boldsymbol{r_1}} = q_{x1}^{\bar{\imath}} + q_{y1}^{\bar{\jmath}} + q_{z1}^{\bar{k}} \tag{4.10}$$

E a relação entre ambos será:

$$\overline{\boldsymbol{r_1}} = Q\bar{r}Q^* \tag{4.11}$$

Por fim, o autor mostra que um quatérnio que só tenha a parte vetorial pode ser usado para representar uma translação, e quando aplicado sobre um vetor de posição transforma-o em outro:

$$\overline{\boldsymbol{r_2}} = \overline{r_1} + Q_r = (q_{x1} + q_x)\bar{\imath} + (q_{y1} + q_y)\bar{\jmath} + (q_{z1} + q_z)\bar{k} \tag{4.12}$$

Sendo assim, quando um vetor realiza uma rotação e uma translação, elas podem ser combinadas por:

$$\overline{\boldsymbol{r_2}} = Q_r + Q\bar{r}Q^* \tag{4.13}$$

De forma com que a representação da rotação sobre um eixo de coordenadas por quatérnio tem forma mais simples que a representação por matriz de uma transformação homogênea.

É importante o entendimento desses conceitos para facilitar a interpretação das variáveis presentes em alguns dos scripts, como o *JointOrientation.cs.* Os quatérnions não fornecem valores explícitos que facilmente se relacionam com um ou outro movimento do braço, pela

ótica do observador. Apesar de não ser o caso neste projeto, para enviar dados de translação e rotação com precisão a algum tipo de atuador, certamente o usuário terá que utilizar quatérnions para garantir que não ocorram erros associados às limitações de implementação dos ângulos de Euler, por mais que esses sejam satisfatórios em vários outros contextos.

4.3.2 AMBIENTE UNITY 3D

A Unity apresenta uma série de classes próprias do seu ambiente que facilitam bastante o desenvolvimento de aplicações. Projetos são constituídos por cenas (*"scences")* que podem ser interpretadas como diferentes níveis de um jogo. Cada objeto presente em um cena, são denominados "*Game Objects"*. Estes podem ser desde um cubo, ou uma esfera que será manipulada por algum personagem, ou, como é o caso deste projeto, representar um objeto externo, o Myo Armband. Neste projeto, existem três objetos cruciais. O primeiro deles é o *Joint*, com um objeto dedicado chamado *Stick*, que representa o que seria o braço do usuário. O script associado a esse objeto é o *JointOrientation.cs.* Nele estão presentes os valores dos quatérnions discutidos acima. Esse script é responsável por fazer os cálculos relacionados a conversão dos ângulos de Euler e aplica-los ao objeto que representa o braço, com alta precisão. Neste projeto os dados desse script não são diretamente utilizados.

O segundo objeto é denominado *Hub – 1 Myo*, com o script *ThalmicHub.cs* associado ele. Este é responsável pela conexão entre o Myo Connection e o ambiente da Unity. Não é necessária nenhuma alteração nesse script para que a conexão ocorra. Em caso de falha, durante a execução do cena é mostrada uma mensagem informando que é necessário conectar o Myo Armband ou realizar o movimento de sincronia.

O terceiro objeto é denominado *Myo.* Este é o elemento de maior importância para o projeto, uma vez que ele detém as variáveis em que estão armazenadas informações importantes como:

1. Status da conexão bluetooth;
2. Status de bloqueio/desbloqueio do dispositivo;
3. Braço sendo utilizado e eixo de referência;
4. Gesto sendo executado no momento;
5. Valores de EMG;
6. Valores da IMU;

7. Combinação dos sensores em valores normalizados (*Transform*);

Para acessar este script, basta um duplo-clique e ele será aberto no programa ou compilador dedicado do sistema. No caso deste projeto, as edições foram feitas utilizando o Microsoft Visual Studio 2017, com linguagem C#.

A figura abaixo ilustra os elementos utilizados na cena e os dados presentes no *Myo* durante a execução do sistema.

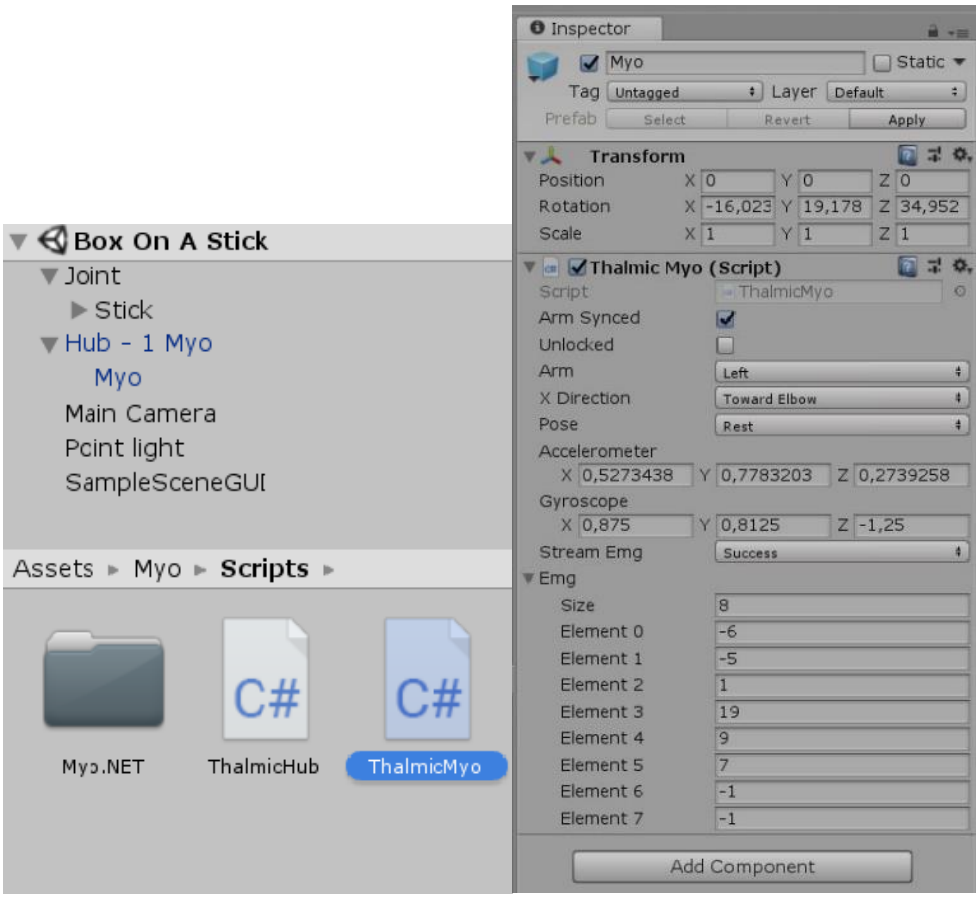

Figura 4.5 – Interface Unity. À esquerda, *GameObjects* utilizados no projeto e o script acessado, *ThalmicMyo.cs* .À direita, elementos presentes no script durante a execução do sistema.

4. 4 INTERFACE: UNITY - ARDUINO

A Thalmic Labs, fabricante do Myo Armband, se antecipou a aplicações que certamente usariam o sensor e já disponibilizou um pacote de integração, com os devidos scripts de conexão, como vimos acima. O mesmo não acontece de forma tão simples com o Arduino.

Durante a busca na literatura, não foram encontrados exemplos práticos dessa conexão. Existem aplicações, em sua grande maioria pagas, na loja da Thalmic Labs, que garantem a integração do Myo Armband diretamente com o Arduino. Como um dos objetivos deste trabalho é justamente desenvolver uma interface que possibilite o uso da Unity para futuras implementações gráficas juntamente com dispositivos capazes de promover algum tipo de atuação, detalhe-se aqui os procedimentos necessários para garantir essa interface.

4. 4. 1 CÓDIGO UNITY

Citam-se aqui os elementos necessários para a comunicação com o Arduino pelo lado de quem envia os dados, no caso a Unity. As linhas de código mostradas na seção referem-se apenas ao setup de comunicação. O código completo, devidamente sequenciado, é apresentado no Apêndice, última seção deste texto.

Uma excelente alternativa para comunicar-se em tempo real com o Arduino, de forma prática e rápida, independentemente da interface que se utiliza, é por meio de sua porta serial. Sendo assim, no script *ThalmicMyO.cs*, onde estão as variáveis utilizadas neste projeto, inclui-se a biblioteca *System.IO.Ports*, que contém classes para controlar portas seriais. Existem outras possibilidades, como a biblioteca de *System.Threading,* que permite uma programação multithread, ou assíncrona. Essa biblioteca também permite um controle temporal de ações mais sofisticado, elemento que é utilizado neste projeto.

```
1.  using System.IO.Ports; // arduino
2.  using System.Threading; // arduino
```

Declara-se então um objeto referenciado a porta serial que será utilizada, no caso a porta serial *"COM5".* Além disso, define-se a taxa de transferência em bits por segundo (*"Baud Rate")* que deve ser a mesma definida no código do Arduino, para que haja sincronia entre os sistemas. Aqui utiliza-se o valor de 9600 por ser um valor típico utilizado em comunicações seriais.

```
3.      public static SerialPort portaSerial = new SerialPort("COM5", 9600);
```

No bloco *Start(),* onde geralmente inciam-se os protocolos de comunicação, a função *Inicia_conexao()* é chamada. Esta, reproduzida logo abaixo, verifica se a porta serial está aberta. Se sim, apresenta uma mensagem de que a porta já está aberta ao console da Unity. Caso

contrário, abre a porta e informa que abriu. Caso a porta não possa ser aberta, o próprio Visual Studio fornece uma mensagem de erro referente ao problema.

```
4.  void Start() {
5.          if (isPaired) {
6.              streamEmg = _myo.SetStreamEmg (_myoStreamEmg);
7.          }
8.
9.          Inicia_conexao();
10.     }
11.     public void Inicia_conexao()
12.     {
13.
14.         if (portaSerial != null)
15.         {
16.             if (portaSerial.IsOpen)
17.             {
18.                 portaSerial.Close();
19.                 print("Porta já está aberta!");
20.             }
21.             else
22.             {
23.                 portaSerial.Open();
24.                 portaSerial.ReadTimeout = 10;
25.                 print("Porta foi aberta!");
26.
27.             }
28.
29.         }
30.
31.     }
```

Uma vez que há comunicação, acessamos as poses reconhecidas pelo Myo Armband e as enviamos, via porta serial, para que o Arduino possa controlar o robô. A Unity envia informações apenas no formato de *strings* pela porta serial. Sendo assim, de forma a exigir menos poder computacional, realizou-se uma pequena codificação, utilizando valores de 0 a 5 para representar os gestos e o “descanso”. Com isso, o Arduino compara apenas um caractere para entender qual o tipo de movimento sendo realizado no momento. O uso do *sleep* da biblioteca *Threading* é fundamental para dar tempo aos servos se moverem sem necessitar de uma mensagem de volta enviado pelo Arduino, o que muitas vezes compromete as rotinas de comunicação. O tempo de espera foi testado empiricamente durante os testes.

```
32.             pose = _myoPose;
33.             unlocked = _myoUnlocked;
34.
35.             if (pose.ToString() == "Rest")
36.             {
37.                 Debug.Log("0");
38.                 portaSerial.Write("0");
39.             }
40.
41.             if (pose.ToString() == "Fist")
```

```
            {
                Debug.Log("1");
                 portaSerial.Write("1");
            }

            if (pose.ToString() == "WaveOut")
            {
                Debug.Log("2");
                portaSerial.Write("2");
                System.Threading.Thread.Sleep(300);
            }

            if (pose.ToString() == "WaveIn")
            {
                Debug.Log("3");
                portaSerial.Write("3");
                System.Threading.Thread.Sleep(300);
            }

            if (pose.ToString() == "FingersSpread")
            {
                Debug.Log("4");
                portaSerial.Write("4");
            }

            if (pose.ToString() == "DoubleTap")
            {
                Debug.Log("5");
                portaSerial.Write("5");
                System.Threading.Thread.Sleep(800);
            }

```

Para finalizar o processo de comunicação, utiliza-se uma classe própria das bibliotecas padrões da Unity para que com o fim da aplicação, a porta serial seja também fechada.

```
    private void OnApplicationQuit()
    {

        portaSerial.Close();
        print("Porta Fechada!");

    }
}
```

4. 4. 2 CÓDIGO ARDUINO

No lado do atuador, o código é mais simples. No setup, abre-se a comunicação serial com o comando *Serial.begin* e o *"baud rate"* combinado.

```
1.  void setup() {
2.
3.
4.    Serial.begin (9600);
```

Após isso, a cada *loop* do Arduino, verifica-se se a conexão está ativa com a *Serial.available(), e* considera-se apenas um byte para comparação. No exemplo abaixo, caso não haja diferença entre a comparação do caractere recebido com o caractere "5", que representa o gesto de toque duplo dos dedos (*"DoubleTap")*, entramos no laço que troca de servo a ser controlado no momento, iterando a variável *SERVO_ATUAL*. O procedimento é o mesmo para quaisquer tipos de dados enviados pela Unity.

```
5.  void loop() {
6.    // put your main code here, to run repeatedly:
7.
8.    delay(4);
9.
10.    if(Serial.available()){
11.
12.     int dado = 1;
13.
14.     Serial.readBytesUntil(dado, pose, 1);
15.
16.     if(strcmp(pose, "5")==0){
17.
18.     SERVO_ATUAL = SERVO_ATUAL + 3;
19.
20.     if(SERVO_ATUAL > 15){
21.       SERVO_ATUAL = 0;
22.
23.     }
24.
25.     }
26.
```

4. 5 INTERFACE: ARDUINO – SERVOMOTORES

Para que o Arduino comande os servomotores que configuram o braço robótico de forma correta, sem comprometer eletricamente nenhum componente do sistema, é necessário realizar algumas conexões básicas. O diagrama abaixo ilustra a configuração do sistema.

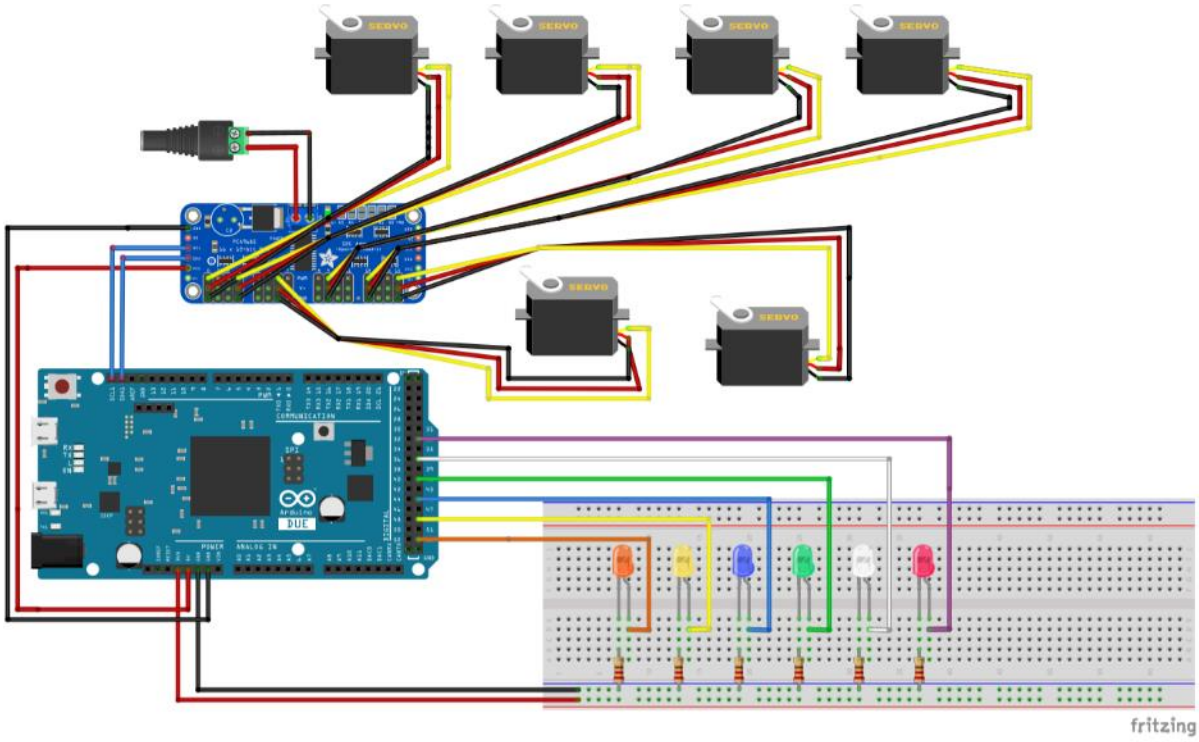

Diagrama 2 – Ligações elétricas entre Arduino, Driver PWM e Servomotores, com LEDs para auxiliar o usuário durante a execução do programa. *Diagrama feito com Fritzing.*

O Arduino DUE apresenta duas saídas analógicas para serem usadas como fonte, uma de 3.3V e outra de 5V. A de menor tensão alimenta o apenas a placa dodriver PCA9685, pois a fonte de alimentação que energizará os motores é dedicada, de 5V, ligada na rede, representada pela pequena tomada na parte superior do diagrama. Os LEDs são acionados pela saída de 5V do Arduino, com utilização de resistores que impedem danos caso haja algum tipo de surto de corrente. Essas conexões podem ser verificadas pelos fios vermelhos. Além disso, os fios pretos representam os pólos negativo ou terra (GND). Os fios coloridos representam as saídas digitais do Arduino que são responsáveis por acionar os LEDs de acordo com o código programado. Os fios azuis que saem das portas SCL e SDA são conectados às respectivas entradas presentes na placa do driver PWM, cujo funcionamento foi explicado na seção 4.2.2

4. 6 SERVOS

Antes de fazer os testes com o sistema completo, usando o Myo Armband e Unity como receptor e mediador dos gestos, é extremamente importante que se faça a calibração de cada um dos servos. Sugere-se que essa etapa seja realizada antes mesmo do encaixe dos motores na configuração final do robô, para definir os mínimos e máximos de cada um dos servos e evitar a perda de referência dos eixos durante o programa. Apesar de ser uma etapa bastante empírica, a Adafruit disponibiliza dois códigos para o Arduino, presentes também na seção de Apêndice, que auxiliam bastante nessa etapa do projeto.

5. RESULTADOS E DISCUSSÕES

Após completar todas as etapas citadas na seção de implementação, deu-se início aos testes. Como era de esperar, alguns problemas foram encontrados. Estes são detalhados abaixo para que a experiência do leitor seja otimizada. A figura abaixo ilustra o sistema:

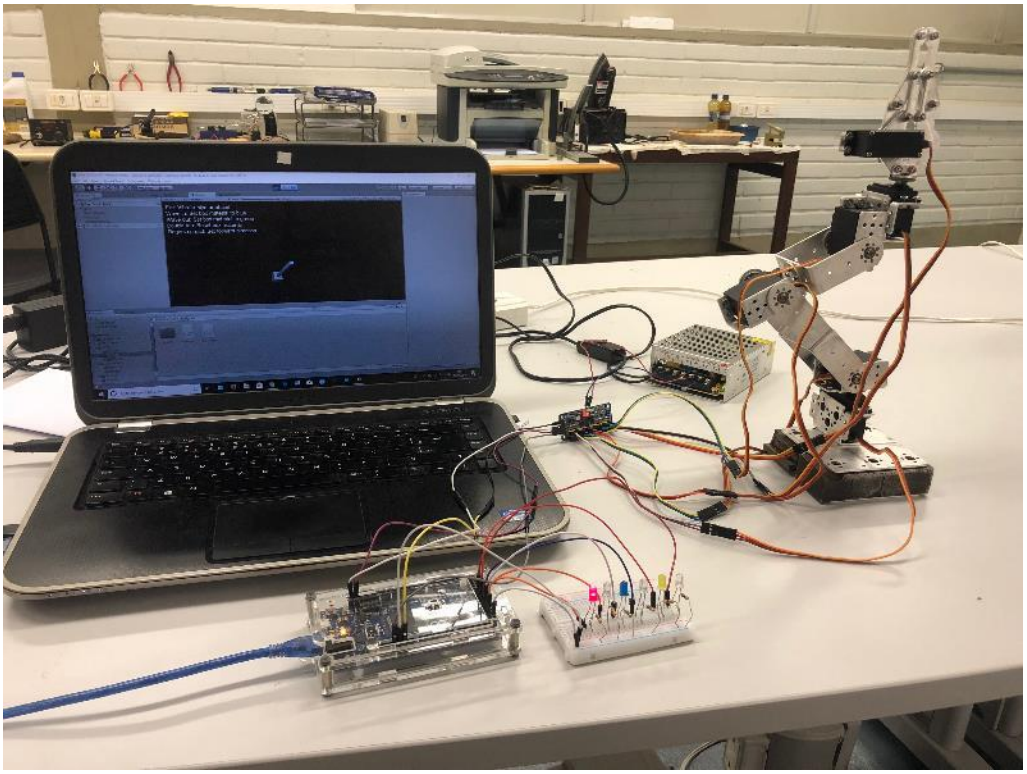

Figura 5.1 – Sistema completo.

Primeiramente, o Myo Armband. O dispositivo se mostrou confiável após um período de testes, algo em torno de três dias. É extremamente importante que o usuário treine com o sensor para de fato entender como os gestos são captados de forma diferenciada entre diferentes braços. Um exemplo pessoal: Durante o gesto de *"WaveIn"*, por exemplo, o posicionamento do polegar interfere bastante na captação do gesto, de forma com que caso ele não esteja direcionado para cima, muitas vezes o movimento não é reconhecido, independentemente da força exercida para direcionar os dedos "para dentro" da mão. O *"DoubleTap"* deve ser realizado com firmeza, e funciona apenas entre os dedos polegar e médio. Deve-se atentar também com a velocidade em que se realiza os gestos. Os outros gestos são detectados com considerável facilidade pelo dispositivo.

Outro ponto acerca do Myo é o tempo de bateria. Apesar de a Thalmic Labs afirmar que, quando completamente carregado, o dispositivo tem capacidade para durar até 24 horas ligado, durante o período de testes, percebeu-se que a bateria durava cerca de um terço que é sugerido pela fabricante. Além disso, não é possível utilizá-lo durante o carregamento. Outro ponto importante é evitar quaisquer movimentos ou obstáculos que desloquem o Myo Armband da posição inicial em que ele sincronizou.

A Unity 3D se mostrou uma excepcional ferramenta de visualização dos dados. Esta reproduz em tempo real movimentos mínimos realizados com o braço e não apresenta perdas de comunicação constantes. Entretanto, é importante testar o tempo a ser definido como ideal para envio das informações para o Arduino por meio da porta serial. Para o movimento da garra, por exemplo, não foi necessário aguardar nenhum instante para enviar outro gesto, uma vez que o servo responsável por este movimento o realizava com uma diferença entre pulsos mínimos e máximos bastante pequena. Não é o caso dos servos que controlam a base do robô, por exemplo.

Para desenvolver um código de fácil entendimento e edição, os servos foram nomeados, calibrados e tiveram seus valores de máximo, mínimo e também de posição inicial explícitos no começo do programa. Assim, o usuário ao ligar o sistema, pode visualmente conferir se concorda ou não com os valores definidos e alterá-los sem a necessidade de se aprofundar no programa. Além disso, a variável *"PASSO",* define a velocidade com que os servos vão girando, dando ao usuário a possibilidade de aumentar a velocidade do sistema. Entretanto, sugere-se muita cautela para não danificar os motores, por mais com que o código não permita que os motores cheguem aos seus valores de limite. O trecho abaixo ilustra o método utilizado.

```
27. #define SERVO_1  0
28. #define SERVO_1_INIC 300
29. #define SERVO_1_MIN  120
30. #define SERVO_1_MAX  590
31. #define PASSO 12
32.
```

O método empregado para controlar os servos, via Arduino, não foi o ideal. Definiu-se controlar cada servo de uma vez, alternando de forma sequencial, subindo da base do robô até sua garra, sem que fosse possível retornar a um motor em que acabou de controlar. Certamente esse método pode ser alterado para garantir maior eficiência no controle. Apesar disso, utilizar os LEDs para informar qual motor está atuando foi uma boa forma de dar *feedback* ao usuário. A figura 5.2 ilustra um teste feito onde o objetivo era pegar a borracha e colocar dentro da caixa.

Figura 5.2 – Montagem sequencial de fotos durante um dos testes realizados. Partindo do canto esquerdo superior, percebe-se que o LED vermelho está aceso, e a cada *DoubleTap* o LED seguinte se acende, o que significa que agora controla-se o servo acima, até chegar na garra e voltar para a base. Os servos do corpo do robô são controlados com os movimentos de *WaveIn* (primeira foto, primeira linha) e *WaveOut* (primeira foto, terceira linha). Na garra, são utilizados o *Fist* para agarrar o objeto (terceira foto, terceira linha) e *FingersSpread* para soltar a borracha na caixa (quinta foto, terceira linha). *O desenvolvimento foi realizado no Laboratório de TV Digital da Universidade de Brasília.*

6. CONSIDERAÇÕES FINAIS

6.1 CONCLUSÃO

O objetivo deste projeto era desenvolver um sistema que integrasse o Myo Armband a uma plataforma que fornecesse uma interface intuitiva e amigável, como é o caso da Unity 3D. Esperava-se também que essa interface possibilitasse a atuação em tempo real, junto ao microcontrolador Arduino, no controle de um braço robótico que respondesse somente aos dados gerados pelo sensor mioelétrico, sem nenhuma rotina pré-programada. Sendo assim, todos os objetivos iniciais foram alcançados, e o leitor já tem em mãos um bom ponto de partida para inovar em aplicações que utilizem os elementos ou configurações aqui apresentadas.

6.2 SUGESTÕES PARA TRABALHOS FUTUROS

Todas as etapas de implementação deste projeto apresentam grande potencial de melhoria. Os dados armazenados pelos quatérnions, por exemplo, não são utilizados neste projeto. Estes são capazes de garantir a rotação e translação espacial do atuador, assim como é apresentado no objeto *Stick* da Unity 3D, gerando uma experiência muito mais real e confortável ao usuário do que o método aqui utilizado. Além disso, a Unity permite todo o tipo de aplicação gráfica. É possível criar um objeto com formato de braço humano, integrar o dispositivo Leap Motion para simular as mãos, e treinar uma Rede Neural que garanta a integração desses elementos, gerando um projeto extremamente completo e raro na literatura.

Por fim, com a utilização de uma prótese inteligente mais elaborada, com mais graus de liberdade, maior destreza e mobilidade, disponibilizaria um amplo espectro de oportunidades que não podem ser exploradas com o braço robótico aqui utilizado. Apesar do custo, essas próteses podem sim, se integradas com softwares devidamente preparados, representar um ganho na qualidade de vida de pessoas que perderam seus membros, uma vez que caminhamos em direção ao fechamento da malha de controle desses dispositivos, ou seja, projetando dispositivos que não só são capazes de entender nossas intenções via sensores neurais, mas que são capazes de fornecer *biofeedback,* do ambiente. Sejam para aplicações de entretenimento ou clínicas, que este seja um projeto facilitador e inspirador.

REFERÊNCIAS BIBLIOGRÁFICAS

AUGUSTIN'E LAWLER, PETER. (2005). *"The Problem of Technology. Perspectives on Political Science"*. 34. 125-134. 10.3200/PPSC.34.3.125-134.

G. LOBACCARO, S. CARLUCCI, AND E. LÖFSTRÖM, *"A Review of Systems and Technologies for Smart Homes and Smart Grids"*. Energies, vol. 9, no. 5, p. 348, May 2016.

PARKER PA, SCOTT RN, *"Myoelectric control of prostheses"*. Critical Reviews in Biomedical Engineering. 1986; 13(4): 283-310.

PARKER PA, ENGLEHART K, HUDGINS B, *"Myoelectric signal processing for control of powered limb prostheses"*. Journal of Electromyograph and Kinesiology. 2006; 16 (6): 541-548.

LJ. S. WEST, G. G. M. CANNING, S. A. PERRYMAN, AND K. KING, *"Novel Technologies for the detection of Fusarium head blight disease and airborne inoculum."* Tropical Plant Pathology, vol. 42, no. 3, pp. 203–209, Feb. 2017.

FARINA D, JIANG N, REHBAUM H, HOLOBAR A, GRAIMANN B, DIETL H, ASZMANN OC, *"The extraction of neural information from the surface EMG for the control of upper-limb protheses: emerging avenues and challenges"*. IEEE Trans Neural Syst Rehabl Eng 2014; 22(4): 797-809

SOARES, INGRID. " *No Brasil, o diabetes provocou 12.478 amputações só em 2017".* Disponível em: <https://www.correiobraziliense.com.br/app/noticia/brasil/2018/08/04/interna-brasil,699209/no-brasil-o-diabetes-provocou-12-748-amputacoes-so-em-2017.shtml.> Acesso em: 28 Jan. 2019.

JAMALUDDIN, F. N., AHMAD, S. A., NOOR, S. B. M., & HASAN, W. Z. W. (2014). *Flexible bio-signals channels acquisition system for ECG and EMG application*. In 2014 IEEE Student Conference on Research and Development. IEEE. https://doi.org/10.1109/scored.2014.7072996

PARASKEVOPOULOU, S. E., EFTEKHAR, A., KULASEKERAM, N., & TOUMAZOU, C. (2015). *A low-noise instrumentation amplifier with DC suppression for*

recording ENG signals. In 2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC). IEEE. https://doi.org/10.1109/embc.2015.7318947

N. V THAKOR, *"Biopotentials and Electrophysiology Measurement",* in Measurements, Instrumentations and Sensors Handbook, CRC Press LLC, 1999.

UNSAL, D., AND DEMIRBAS, K. 2012. *"Estimation of Deterministic and Stochastic IMU Error Parameters."* Record - IEEE PLANS, Position Location and Navigation Symposium (2): 862–68.

BRUNNER, T., LAUFFENBURGER, J. P., CHANGEY, S., AND BASSET, M. 2015. *"Magnetometer Augmented IMU Simulator: In-Depth Elaboration."* Sensors (Switzerland) 15(3): 5293–5310.

ANOWARUL FATTAH, S. (2012). Identifying the Motor Neuron Disease in EMG Signal *Using Time and Frequency Domain Features with Comparison. Signal & Image Processing* : An International Journal, 3(2), 99–114. https://doi.org/10.5121/sipij.2012.3207

BISWAL, D., UVANESH, K., CHAMPATY, B., RAY, S. S., & PAL, K. (2014). *Development of an ambulatory universal bio potential recording device.* 2014 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT). doi:10.1109/iccicct.2014.6993130

MOSLEM, B., KHALIL, M., DIAB, M. O., CHKEIR, A., & MARQUE, C. (2011). *A multisensor data fusion approach for improving the classification accuracy of uterine EMG signals.* In 2011 18th IEEE International Conference on Electronics, Circuits, and Systems. IEEE. doi.org/10.1109/icecs.2011.6122222

ACCESS PROSTHETICS, 2017. *"15 Limb Loss Statitics That May Surprise You".* Disponível em: <https://accessprosthetics.com/15-limb-loss-statistics-may-surprise/.>. Acesso em: 28 Jan. 2019.

FOURATI, H., MANAMANNI, N., AFILAL, L., AND HANDRICH, Y. 2014. *"Complementary Observer for Body Segments Motion Capturing by Inertial and Magnetic Sensors."* IEEE/ASME Transactions on Mechatronics 19(1): 149–57.

ROETENBERG, D., LUINGE, H., AND VELTINK, P. 2003. *"Inertial and Magnetic Sensing of Human Movement near Ferromagnetic Materials."* In Proceedings - 2nd IEEE and

ACM International Symposium on Mixed and Augmented Reality, ISMAR 2003, IEEE Comput.Soc, 268 http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1240714 (August 31, 2016).

NETO, P., PEREIRA, D., PIRES, J. N., AND MOREIRA, A. P. 2013. *"Real-Time and Continuous Hand Gesture Spotting: An Approach Based on Artificial Neural Networks."* 2013 IEEE International Conference on Robotics and Automation: 178–83. http://arxiv.org/abs/1309.2084 (March 11, 2016).

JUNG, J. Y., HEO, W., YANG, H., AND PARK, H. 2015. *"A Neural Network-Based Gait Phase Classification Method Using Sensors Equipped on Lower Limb Exoskeleton Robots."* Sensors (Switzerland) 15(11): 27738–59. Disponível em: <http://www.mdpi.com/1424- 8220/15/11/27738/ (March 14, 2016).>

KING, A. D. 1998. *"Inertial Navigation - Forty Years of Evolution."* Gec Review 13(3): 140–49.

WEILI, S., 2014. *"Using MYO Armband in Performances."* http://shi-weili.com/using-myoarmband-in-performances/ (August 30, 2016).

BERNHARDT, P., 2015 "*Raw and Uncut Drops Today."* Disponível em: < https://developerblog.myo.com/raw-uncut-drops-today/>. Acesso em: 29 Jan. 2019.

NYOMEN, K., HAUGEN, M. R. AND JENSENIUS, A. R. 2015 *"MuMYO — Evaluating and Exploring the MYO Armband for Musical Interaction."* Proceedings of the International Conference on New Interfaces for Musical Expression: 215–18. https://nime2015.lsu.edu/proceedings/179/0179-paper.pdf (April 7, 2016).

APÊNDICES

1 – CÓDIGO UNITY: *ThalmicMyo.CS*

```
82. using UnityEngine;
83. using System.Collections;
84. using System.IO.Ports; // arduino
85. using System.Threading; // arduino
86. 
87. using Arm = Thalmic.Myo.Arm;
88. using XDirection = Thalmic.Myo.XDirection;
89. using VibrationType = Thalmic.Myo.VibrationType;
90. using Pose = Thalmic.Myo.Pose;
91. using LockingPolicy = Thalmic.Myo.LockingPolicy;
92. using UnlockType = Thalmic.Myo.UnlockType;
93. using StreamEmg = Thalmic.Myo.StreamEmg;
94. 
95. // Represents a Myo armband. Myo's orientation is made available through transform.l
    ocalRotation, and other properties
96. // like the current pose are provided explicitly below. All spatial data about Myo i
    s provided following Unity
97. // coordinate system conventions (the y axis is up, the z axis is forward, and the c
    oordinate system is left-handed).
98. public class ThalmicMyo : MonoBehaviour {
99. 
100.           public static SerialPort portaSerial = new SerialPort("COM5", 9600);
101. 
102.           // True if and only if Myo has detected that it is on an arm.
103.           public bool armSynced;
104. 
105.           // Returns true if and only if Myo is unlocked.
106.           public bool unlocked;
107. 
108.           // The current arm that Myo is being worn on. An arm of Unknown means tha
    t Myo is unable to detect the arm
109.           // (e.g. because it's not currently being worn).
110.           public Arm arm;
111. 
112.           // The current direction of Myo's +x axis relative to the user's arm. A x
    Direction of Unknown means that Myo is
113.           // unable to detect the direction (e.g. because it's not currently being
    worn).
114.           public XDirection xDirection;
115. 
116.           // The current pose detected by Myo. A pose of Unknown means that Myo is
    unable to detect the pose (e.g. because
117.           // it's not currently being worn).
118.           public Pose pose = Pose.Unknown;
119. 
```

```
120.            // Myo's current accelerometer reading, representing the acceleration due
     to force on the Myo armband in units of
121.            // g (roughly 9.8 m/s^2) and following Unity coordinate system convention
    s.
122.            public Vector3 accelerometer;
123.
124.            // Myo's current gyroscope reading, representing the angular velocity abo
    ut each of Myo's axes in degrees/second
125.            // following Unity coordinate system conventions.
126.            public Vector3 gyroscope;
127.
128.            public Thalmic.Myo.Result streamEmg;
129.
130.            public int[] emg;
131.
132.            // True if and only if this Myo armband has paired successfully, at which
     point it will provide data and a
133.            // connection with it will be maintained when possible.
134.            public bool isPaired {
135.                get { return _myo != null; }
136.            }
137.
138.            // Vibrate the Myo with the provided type of vibration, e.g. VibrationTyp
    e.Short or VibrationType.Medium.
139.            public void Vibrate (VibrationType type) {
140.                _myo.Vibrate (type);
141.            }
142.
143.            // Cause the Myo to unlock with the provided type of unlock. e.g. UnlockT
    ype.Timed or UnlockType.Hold.
144.            public void Unlock (UnlockType type) {
145.                _myo.Unlock (type);
146.            }
147.
148.            // Cause the Myo to re-lock immediately.
149.            public void Lock () {
150.                _myo.Lock ();
151.            }
152.
153.            /// Notify the Myo that a user action was recognized.
154.            public void NotifyUserAction () {
155.                _myo.NotifyUserAction ();
156.            }
157.
158.
159.            void Start() {
160.                if (isPaired) {
161.                    streamEmg = _myo.SetStreamEmg (_myoStreamEmg);
162.                }
163.
164.                Inicia_conexao();
165.            }
166.
167.            void Update() {
168.
169.                lock (_lock) {
170.                    armSynced = _myoArmSynced;
171.                    arm = _myoArm;
172.                    xDirection = _myoXDirection;
173.                    if (_myoQuaternion != null) {
174.                        transform.localRotation = new Quaternion(_myoQuaternion.Y, _m
    yoQuaternion.Z, -_myoQuaternion.X, -_myoQuaternion.W);
175.                    }
176.                    if (_myoAccelerometer != null) {
```

```
177.                         accelerometer = new Vector3(_myoAccelerometer.Y, _myoAccelero
    meter.Z, -_myoAccelerometer.X);
178.                     }
179.                     if (_myoGyroscope != null) {
180.                         gyroscope = new Vector3(_myoGyroscope.Y, _myoGyroscope.Z, -
    _myoGyroscope.X);
181.                     }
182.                     if (isPaired && streamEmg == Thalmic.Myo.Result.Success) {
183.                         emg = _myo.emgData;
184.                     }
185.
186.                     pose = _myoPose;
187.                     unlocked = _myoUnlocked;
188.
189.                     if (pose.ToString() == "Rest")
190.                     {
191.                         Debug.Log("0");
192.                         portaSerial.Write("0");
193.                     }
194.
195.                     if (pose.ToString() == "Fist")
196.                     {
197.                        Debug.Log("1");
198.                         portaSerial.Write("1");
199.                     }
200.
201.                     if (pose.ToString() == "WaveOut")
202.                     {
203.                         Debug.Log("2");
204.                         portaSerial.Write("2");
205.                         System.Threading.Thread.Sleep(300);
206.                     }
207.
208.                     if (pose.ToString() == "WaveIn")
209.                     {
210.                         Debug.Log("3");
211.                         portaSerial.Write("3");
212.                         System.Threading.Thread.Sleep(300);
213.                     }
214.
215.                     if (pose.ToString() == "FingersSpread")
216.                     {
217.                         Debug.Log("4");
218.                         portaSerial.Write("4");
219.                     }
220.
221.                     if (pose.ToString() == "DoubleTap")
222.                     {
223.                         Debug.Log("5");
224.                         portaSerial.Write("5");
225.                         System.Threading.Thread.Sleep(800);
226.                     }
227.
228.                 }
229.             }
230.
231.             public void Inicia_conexao()
232.             {
233.
234.                 if (portaSerial != null)
235.                 {
236.                     if (portaSerial.IsOpen)
237.                     {
238.                         portaSerial.Close();
```

```
239.                         print("Porta já está aberta!");
240.                     }
241.                     else
242.                     {
243.                         portaSerial.Open();
244.                         portaSerial.ReadTimeout = 10;
245.                         print("Porta foi aberta!");
246.
247.                     }
248.
249.                 }
250.
251.             }
252.
253.
254.             void myo_OnArmSync(object sender, Thalmic.Myo.ArmSyncedEventArgs e) {
255.                 lock (_lock) {
256.                     _myoArmSynced = true;
257.                     _myoArm = e.Arm;
258.                     _myoXDirection = e.XDirection;
259.                 }
260.             }
261.
262.             void myo_OnArmUnsync(object sender, Thalmic.Myo.MyoEventArgs e) {
263.                 lock (_lock) {
264.                     _myoArmSynced = false;
265.                     _myoArm = Arm.Unknown;
266.                     _myoXDirection = XDirection.Unknown;
267.                 }
268.             }
269.
270.             void myo_OnOrientationData(object sender, Thalmic.Myo.OrientationDataEven
    tArgs e) {
271.                 lock (_lock) {
272.                     _myoQuaternion = e.Orientation;
273.                 }
274.             }
275.
276.             void myo_OnAccelerometerData(object sender, Thalmic.Myo.AccelerometerData
    EventArgs e) {
277.                 lock (_lock) {
278.                     _myoAccelerometer = e.Accelerometer;
279.                 }
280.             }
281.
282.             void myo_OnGyroscopeData(object sender, Thalmic.Myo.GyroscopeDataEventArg
    s e) {
283.                 lock (_lock) {
284.                     _myoGyroscope = e.Gyroscope;
285.                 }
286.             }
287.
288.             void myo_OnEmgData(object sender, Thalmic.Myo.EmgDataEventArgs e) {
289.                 lock (_lock) {
290.                     _myoEmg = e.Emg;
291.                 }
292.             }
293.
294.             public void myo_OnPoseChange(object sender, Thalmic.Myo.PoseEventArgs e)
    {
295.                 lock (_lock) {
296.                     _myoPose = e.Pose;
297.                 }
298.             }
```

```
299.
300.            void myo_OnUnlock(object sender, Thalmic.Myo.MyoEventArgs e) {
301.                lock (_lock) {
302.                    _myoUnlocked = true;
303.                }
304.            }
305.
306.            void myo_OnLock(object sender, Thalmic.Myo.MyoEventArgs e) {
307.                lock (_lock) {
308.                    _myoUnlocked = false;
309.                }
310.            }
311.
312.            public Thalmic.Myo.Myo internalMyo {
313.                get { return _myo; }
314.                set {
315.                    if (_myo != null) {
316.                        _myo.ArmSynced -= myo_OnArmSync;
317.                        _myo.ArmUnsynced -= myo_OnArmUnsync;
318.                        _myo.OrientationData -= myo_OnOrientationData;
319.                        _myo.AccelerometerData -= myo_OnAccelerometerData;
320.                        _myo.GyroscopeData -= myo_OnGyroscopeData;
321.                        _myo.PoseChange -= myo_OnPoseChange;
322.                        _myo.Unlocked -= myo_OnUnlock;
323.                        _myo.Locked -= myo_OnLock;
324.                    }
325.                    _myo = value;
326.                    if (value != null) {
327.                        value.ArmSynced += myo_OnArmSync;
328.                        value.ArmUnsynced += myo_OnArmUnsync;
329.                        value.OrientationData += myo_OnOrientationData;
330.                        value.AccelerometerData += myo_OnAccelerometerData;
331.                        value.GyroscopeData += myo_OnGyroscopeData;
332.                        value.EmgData += myo_OnEmgData;
333.                        value.PoseChange += myo_OnPoseChange;
334.                        value.Unlocked += myo_OnUnlock;
335.                        value.Locked += myo_OnLock;
336.                    }
337.                }
338.            }
339.
340.            private Object _lock = new Object();
341.
342.            private bool _myoArmSynced = false;
343.            private Arm _myoArm = Arm.Unknown;
344.            private XDirection _myoXDirection = XDirection.Unknown;
345.            private Thalmic.Myo.Quaternion _myoQuaternion = null;
346.            private Thalmic.Myo.Vector3 _myoAccelerometer = null;
347.            private Thalmic.Myo.Vector3 _myoGyroscope = null;
348.            private int[] _myoEmg = new int[7];
349.            private Pose _myoPose = Pose.Unknown;
350.            private bool _myoUnlocked = false;
351.            private StreamEmg _myoStreamEmg = StreamEmg.Enabled;
352.
353.            private Thalmic.Myo.Myo _myo;
354.
355.            private void OnApplicationQuit()
356.            {
357.
358.                portaSerial.Close();
359.                print("Porta Fechada!");
360.
361.            }
362.        }
```

2 – ARDUINO

```
#include <Wire.h>
#include <Adafruit_PWMServoDriver.h>

// Valores de máximo e minimo todos com margem de 10 unidades

#define SERVO_1  0
#define SERVO_1_INIC 300
#define SERVO_1_MIN  120
#define SERVO_1_MAX  590

#define SERVO_2  3
#define SERVO_2_INIC 450
#define SERVO_2_MIN  110
#define SERVO_2_MAX  550

#define SERVO_3  6
#define SERVO_3_INIC 500
#define SERVO_3_MIN  120
#define SERVO_3_MAX  550

#define SERVO_4  9
#define SERVO_4_INIC 500
#define SERVO_4_MIN  140
#define SERVO_4_MAX  560

#define SERVO_5  12
#define SERVO_5_INIC 200
#define SERVO_5_MIN  160
#define SERVO_5_MAX  590

#define SERVO_6  15
#define SERVO_6_INIC 250
#define SERVO_6_RELAX 175
#define SERVO_6_MIN  95
#define SERVO_6_MAX  250

#define PASSO 12

Adafruit_PWMServoDriver pwm = Adafruit_PWMServoDriver();

char pose[1];

int SERVO_1_ANTIGO = SERVO_1_INIC;
int SERVO_1_ATUAL;
int SERVO_2_ANTIGO = SERVO_2_INIC;
int SERVO_2_ATUAL;
int SERVO_3_ANTIGO = SERVO_3_INIC;
int SERVO_3_ATUAL;
int SERVO_4_ANTIGO = SERVO_4_INIC;
int SERVO_4_ATUAL;
int SERVO_5_ANTIGO = SERVO_5_INIC;
int SERVO_5_ATUAL;
int SERVO_6_ANTIGO = SERVO_6_INIC;
int SERVO_6_ATUAL;
```

```
87.
88.
89. void setup() {
90.   // put your setup code here, to run once:
91.
92.    int SERVO_ATUAL = 0;
93.
94.   pinMode(31, OUTPUT);
95.   pinMode(35, OUTPUT);
96.   pinMode(39, OUTPUT);
97.   pinMode(41, OUTPUT);
98.   pinMode(45, OUTPUT);
99.   pinMode(49, OUTPUT);
100.
101.         Serial.begin (9600);
102.
103.
104.         pwm.begin();
105.         pwm.setPWMFreq(60);
106.
107.
108.         pwm.setPWM(SERVO_2, 0, SERVO_2_INIC);
109.         delay(2000);
110.         pwm.setPWM(SERVO_3, 0, SERVO_3_INIC);
111.         delay(2000);
112.         pwm.setPWM(SERVO_1, 0, SERVO_1_INIC);
113.         delay(2000);
114.         pwm.setPWM(SERVO_4, 0, SERVO_4_INIC);
115.         delay(2000);
116.         pwm.setPWM(SERVO_5, 0, SERVO_5_INIC);
117.         delay(2000);
118.         pwm.setPWM(SERVO_6, 0, SERVO_6_INIC);
119.         delay(2000);
120.
121.       }
122.
123.
124.       void loop() {
125.         // put your main code here, to run repeatedly:
126.
127.         delay(4);
128.
129.          if(Serial.available()){
130.
131.           int dado = 1;
132.
133.           Serial.readBytesUntil(dado, pose, 1);
134.
135.           if(strcmp(pose, "5")==0){
136.
137.           SERVO_ATUAL = SERVO_ATUAL + 3;
138.
139.           if(SERVO_ATUAL > 15){
140.             SERVO_ATUAL = 0;
141.
142.           }
143.
144.           }
145.
146.       // SERVO 1 (BASE)
147.
148.       if(SERVO_ATUAL == SERVO_1){
149.         digitalWrite(31, LOW);
150.         digitalWrite(49, HIGH);
```

```
151.
152.           if(strcmp(pose, "3")==0){   //WaveIn
153.
154.             SERVO_1_ATUAL = SERVO_1_ANTIGO + PASSO ;
155.
156.             if(SERVO_1_ATUAL >= SERVO_1_MAX){
157.               SERVO_1_ATUAL = SERVO_1_ANTIGO - PASSO ;
158.             }
159.
160.             pwm.setPWM(SERVO_ATUAL, 0, SERVO_1_ATUAL);
161.             SERVO_1_ANTIGO = SERVO_1_ANTIGO + PASSO;
162.
163.             }
164.
165.          if(strcmp(pose, "2")==0){ //WaveOut
166.
167.             SERVO_1_ATUAL = SERVO_1_ANTIGO - PASSO;
168.
169.            if(SERVO_1_ATUAL <= SERVO_1_MIN){
170.               SERVO_1_ATUAL = SERVO_1_ANTIGO + PASSO ;
171.             }
172.
173.            pwm.setPWM(SERVO_ATUAL, 0, SERVO_1_ATUAL);
174.            SERVO_1_ANTIGO = SERVO_1_ANTIGO - PASSO;
175.
176.
177.           }
178.       }
179.
180.       // SERVO 2
181.
182.       if(SERVO_ATUAL == 3){
183.         digitalWrite(49, LOW);
184.         digitalWrite(45, HIGH);
185.
186.        if(strcmp(pose, "3")==0){   //WaveIn
187.
188.             SERVO_2_ATUAL = SERVO_2_ANTIGO + PASSO ;
189.
190.             if(SERVO_2_ATUAL >= SERVO_2_MAX){
191.               SERVO_2_ATUAL = SERVO_2_ANTIGO - PASSO ;
192.             }
193.
194.             pwm.setPWM(SERVO_ATUAL, 0, SERVO_2_ATUAL);
195.             SERVO_2_ANTIGO = SERVO_2_ANTIGO + PASSO;
196.
197.             }
198.
199.
200.
201.          if(strcmp(pose, "2")==0){ //WaveOut
202.
203.             SERVO_2_ATUAL = SERVO_2_ANTIGO - PASSO;
204.
205.            if(SERVO_2_ATUAL <= SERVO_2_MIN){
206.               SERVO_2_ATUAL = SERVO_2_ANTIGO + PASSO ;
207.             }
208.
209.            pwm.setPWM(SERVO_ATUAL, 0, SERVO_2_ATUAL);
210.            SERVO_2_ANTIGO = SERVO_2_ANTIGO - PASSO;
211.
212.
213.           }
214.
```

```
215.       }
216.
217.       // SERVO 3
218.
219.       if(SERVO_ATUAL == 6){
220.         digitalWrite(45, LOW);
221.         digitalWrite(41, HIGH);
222.
223.       if(strcmp(pose, "3")==0){   //WaveIn
224.
225.             SERVO_3_ATUAL = SERVO_3_ANTIGO + PASSO ;
226.
227.             if(SERVO_3_ATUAL >= SERVO_3_MAX){
228.               SERVO_3_ATUAL = SERVO_3_ANTIGO - PASSO ;
229.             }
230.
231.             pwm.setPWM(SERVO_ATUAL, 0, SERVO_3_ATUAL);
232.             SERVO_3_ANTIGO = SERVO_3_ANTIGO + PASSO;
233.
234.             }
235.
236.          if(strcmp(pose, "2")==0){ //WaveOut
237.
238.             SERVO_3_ATUAL = SERVO_3_ANTIGO - PASSO;
239.
240.            if(SERVO_3_ATUAL <= SERVO_3_MIN){
241.               SERVO_3_ATUAL = SERVO_3_ANTIGO + PASSO ;
242.             }
243.
244.            pwm.setPWM(SERVO_ATUAL, 0, SERVO_3_ATUAL);
245.            SERVO_3_ANTIGO = SERVO_3_ANTIGO - PASSO;
246.
247.
248.           }
249.
250.       }
251.
252.       // SERVO 4
253.
254.       if(SERVO_ATUAL == 9){
255.         digitalWrite(41, LOW);
256.         digitalWrite(39, HIGH);
257.
258.          if(strcmp(pose, "3")==0){   //WaveIn
259.
260.             SERVO_4_ATUAL = SERVO_4_ANTIGO + PASSO ;
261.
262.             if(SERVO_4_ATUAL >= SERVO_4_MAX){
263.               SERVO_4_ATUAL = SERVO_4_ANTIGO - PASSO ;
264.             }
265.
266.             pwm.setPWM(SERVO_ATUAL, 0, SERVO_4_ATUAL);
267.             SERVO_4_ANTIGO = SERVO_4_ANTIGO + PASSO;
268.
269.             }
270.
271.
272.
273.          if(strcmp(pose, "2")==0){ //WaveOut
274.
275.             SERVO_4_ATUAL = SERVO_4_ANTIGO - PASSO;
276.
277.            if(SERVO_4_ATUAL <= SERVO_4_MIN){
278.               SERVO_4_ATUAL = SERVO_4_ANTIGO + PASSO ;
```

```
279.              }
280.
281.             pwm.setPWM(SERVO_ATUAL, 0, SERVO_4_ATUAL);
282.             SERVO_4_ANTIGO = SERVO_4_ANTIGO - PASSO;
283.
284.
285.           }
286.       }
287.
288.       // SERVO 5
289.
290.       if(SERVO_ATUAL == 12){
291.         digitalWrite(39, LOW);
292.         digitalWrite(35, HIGH);
293.
294.
295.           if(strcmp(pose, "3")==0){   //WaveIn
296.
297.             SERVO_5_ATUAL = SERVO_5_ANTIGO + PASSO ;
298.
299.             if(SERVO_5_ATUAL >= SERVO_5_MAX){
300.               SERVO_5_ATUAL = SERVO_5_ANTIGO - PASSO ;
301.             }
302.
303.             pwm.setPWM(SERVO_ATUAL, 0, SERVO_5_ATUAL);
304.             SERVO_5_ANTIGO = SERVO_5_ANTIGO + PASSO;
305.
306.             }
307.
308.          if(strcmp(pose, "2")==0){ //WaveOut
309.
310.             SERVO_5_ATUAL = SERVO_5_ANTIGO - PASSO;
311.
312.            if(SERVO_5_ATUAL <= SERVO_5_MIN){
313.               SERVO_5_ATUAL = SERVO_5_ANTIGO + PASSO ;
314.             }
315.
316.            pwm.setPWM(SERVO_ATUAL, 0, SERVO_5_ATUAL);
317.            SERVO_5_ANTIGO = SERVO_5_ANTIGO - PASSO;
318.
319.
320.           }
321.
322.       }
323.
324.
325.        // SERVO 6 (GARRA)
326.
327.        if(SERVO_ATUAL == 15){
328.
329.             digitalWrite(35, LOW);
330.             digitalWrite(31, HIGH);
331.
332.       //       if(strcmp(pose, "0")==0){
333.       //       pwm.setPWM(SERVO_ATUAL, 0, SERVO_6_RELAX);
334.       //     }
335.
336.             if(strcmp(pose, "1")==0){
337.            pwm.setPWM(SERVO_ATUAL, 0, SERVO_6_MAX);
338.
339.             }
340.
341.            if(strcmp(pose, "4")==0){
342.             pwm.setPWM(SERVO_ATUAL, 0, SERVO_6_MIN);
```

```
343.
344.             }
345.           }
346.         }
347.       }
```

3 – CALIBRAÇÃO (ADAFRUIT)

```
1.  /*************************************************
2.    This is an example for our Adafruit 16-channel PWM & Servo driver
3.    PWM test - this will drive 16 PWMs in a 'wave'
4.
5.    Pick one up today in the adafruit shop!
6.    ------> http://www.adafruit.com/products/815
7.
8.    These displays use I2C to communicate, 2 pins are required to
9.    interface. For Arduino UNOs, thats SCL -> Analog 5, SDA -> Analog 4
10.
11.   Adafruit invests time and resources providing this open source code,
12.   please support Adafruit and open-source hardware by purchasing
13.   products from Adafruit!
14.
15.   Written by Limor Fried/Ladyada for Adafruit Industries.
16.   BSD license, all text above must be included in any redistribution
17.  *************************************************/
18.
19. #include <Wire.h>
20. #include <Adafruit_PWMServoDriver.h>
21.
22. // called this way, it uses the default address 0x40
23. Adafruit_PWMServoDriver pwm = Adafruit_PWMServoDriver();
24. // you can also call it with a different address you want
25. //Adafruit_PWMServoDriver pwm = Adafruit_PWMServoDriver(0x41);
26.
27. void setup() {
28.   Serial.begin(9600);
29.   Serial.println("16 channel PWM test!");
30.
31.   // if you want to really speed stuff up, you can go into 'fast 400khz I2C' mode
32.   // some i2c devices dont like this so much so if you're sharing the bus, watch
33.   // out for this!
34.
35.   pwm.begin();
36.   pwm.setPWMFreq(1600);  // This is the maximum PWM frequency
37.
38.   // save I2C bitrate
39.   uint8_t twbrbackup = TWBR;
40.   // must be changed after calling Wire.begin() (inside pwm.begin())
41.   TWBR = 12; // upgrade to 400KHz!
42.
43. }
44.
45. void loop() {
46.   // Drive each PWM in a 'wave'
47.   for (uint16_t i=0; i<4096; i += 8) {
48.     for (uint8_t pwmnum=0; pwmnum < 16; pwmnum++) {
49.       pwm.setPWM(pwmnum, 0, (i + (4096/16)*pwmnum) % 4096 );
50.     }
51.   }
52. }
```

```
/*************************************************
  This is an example for our Adafruit 16-channel PWM & Servo driver
  Servo test - this will drive 16 servos, one after the other

  Pick one up today in the adafruit shop!
  ------> http://www.adafruit.com/products/815

  These displays use I2C to communicate, 2 pins are required to
  interface. For Arduino UNOs, thats SCL -> Analog 5, SDA -> Analog 4

  Adafruit invests time and resources providing this open source code,
  please support Adafruit and open-source hardware by purchasing
  products from Adafruit!

  Written by Limor Fried/Ladyada for Adafruit Industries.
  BSD license, all text above must be included in any redistribution
 *************************************************/

#include <Wire.h>
#include <Adafruit_PWMServoDriver.h>

// called this way, it uses the default address 0x40
Adafruit_PWMServoDriver pwm = Adafruit_PWMServoDriver();
// you can also call it with a different address you want
//Adafruit_PWMServoDriver pwm = Adafruit_PWMServoDriver(0x41);

// Depending on your servo make, the pulse width min and max may vary, you
// want these to be as small/large as possible without hitting the hard stop
// for max range. You'll have to tweak them as necessary to match the servos you
// have!
#define SERVOMIN  150 // this is the 'minimum' pulse length count (out of 4096)
#define SERVOMAX  600 // this is the 'maximum' pulse length count (out of 4096)

// our servo # counter
uint8_t servonum = 0;

void setup() {
  Serial.begin(9600);
  Serial.println("16 channel Servo test!");

  pwm.begin();

  pwm.setPWMFreq(60);  // Analog servos run at ~60 Hz updates
}

// you can use this function if you'd like to set the pulse length in seconds
// e.g. setServoPulse(0, 0.001) is a ~1 millisecond pulse width. its not precise!
      void setServoPulse(uint8_t n, double pulse) {
        double pulselength;

        pulselength = 1000000;   // 1,000,000 us per second
        pulselength /= 60;   // 60 Hz
        Serial.print(pulselength); Serial.println(" us per period");
        pulselength /= 4096;  // 12 bits of resolution
        Serial.print(pulselength); Serial.println(" us per bit");
        pulse *= 1000;
        pulse /= pulselength;
        Serial.println(pulse);
        pwm.setPWM(n, 0, pulse);
      }

      void loop() {
        // Drive each servo one at a time
        Serial.println(servonum);
```

```
117.         for (uint16_t pulselen = SERVOMIN; pulselen < SERVOMAX; pulselen++) {
118.           pwm.setPWM(servonum, 0, pulselen);
119.         }
120.         delay(500);
121.         for (uint16_t pulselen = SERVOMAX; pulselen > SERVOMIN; pulselen--) {
122.           pwm.setPWM(servonum, 0, pulselen);
123.         }
124.         delay(500);
125.
126.         servonum ++;
127.         if (servonum > 15) servonum = 0;
128.       }
```