UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CÂMPUS CURITIBA
ENGENHARIA ELÉTRICA

NICOLAS JAN MULDER
VITOR AMANCIO JERONYMO

DESENVOLVIMENTO DE PROTÓTIPO PARA IDENTIFICAÇÃO DE MOVIMENTOS DE BOXE

TRABALHO DE CONCLUSÃO DE CURSO

CURITIBA
2017

NICOLAS JAN MULDER
VITOR AMANCIO JERONYMO

DESENVOLVIMENTO DE PROTÓTIPO PARA IDENTIFICAÇÃO DE MOVIMENTOS DE BOXE

Trabalho de Conclusão de Curso de Graduação, do curso de Engenharia Elétrica do Departamento Acadêmico de Eletrotécnica (DAELT) da Universidade Tecnológica Federal do Paraná (UTFPR) como requisito para obtenção do título de Engenheiro Eletricista.

Orientador: Prof. Dr. Elder Oroski

CURITIBA
2017

Nicolas Jan Mulder
Vitor Amancio Jeronymo

Desenvolvimento de um protótipo para a identificação de movimentos de boxe.

Este Trabalho de Conclusão de Curso de Graduação foi julgado e aprovado como requisito parcial para a obtenção do Título de Engenheiro Eletricista, do curso de Engenharia Elétrica do Departamento Acadêmico de Eletrotécnica (DAELT) da Universidade Tecnológica Federal do Paraná (UTFPR).

Curitiba, 29 de Junho de 2017.

_____
Prof. Emerson Rigoni, Dr.
Coordenador de Curso
Engenharia Elétrica

_____
Profa. Annemarlen Gehrke Castagna, Dra.
Responsável pelos Trabalhos de Conclusão de Curso
de Engenharia Elétrica do DAELT

ORIENTAÇÃO

_____
Elder Oroski, Dr
Universidade Tecnológica Federal do Paraná
Orientador

BANCA EXAMINADORA

_____
Elder Oroski, Dr.
Universidade Tecnológica Federal do Paraná

_____
Cintia de Lourdes Nahhas Rodacki, Dra.
Universidade Tecnológica Federal do Paraná

_____
Rafael Fontes Souto, Dr.
Universidade Tecnológica Federal do Paraná

A folha de aprovação assinada encontra-se na Coordenação do Curso de Engenharia Elétrica

RESUMO

JERONYMO, Vitor, MULDER, Nicolas. DESENVOLVIMENTO DE PROTÓTIPO PARA IDENTIFICAÇÃO DE MOVIMENTOS DE BOXE. 88 f. Trabalho de Conclusão de Curso – Engenharia Elétrica, Universidade Tecnológica Federal do Paraná. Curitiba, 2017.

Devido à falta de dispositivos capazes de auxiliar o treinamento do atleta no local de prática do esporte, decidiu-se desenvolver um protótipo capaz de contabilizar e identificar os principais movimentos do boxe. A pesquisa apresenta uma abordagem teórica e prática de sensores inerciais, conhecimentos de redes neurais, base FIR e conhecimento básico sobre o boxe. O projeto tem o objetivo de utilizar sensores inerciais em luvas de treinamento para identificar os tipos de golpes de boxe desferidos por um lutador. Discute-se os obstáculos e as necessidades do projeto, como os requerimentos dos sensores, quais golpes de boxe identificar e a estrutura da rede neural. Como o dispositivo escolhido, provido de sensores inerciais, permite conectividade *Bluetooth*, criou-se um aplicativo em *Android* para realizar a coleta de golpes "ideais" como também processar a simulação da rede neural já treinada. A coleta de amostras de golpes, sendo estes, jab, direto, *uppercut* e cruzado, foi realizada por dois atletas profissionais de boxe com 14 anos de experiência, dando veracidade aos golpes desenvolvidos. Ao final, o trabalho traz resultados das sessões de testes realizadas, com índice de reconhecimento médio de golpes de $84,8\%$ e faz uma análise sobre os problemas de identificação encontrados.

Palavras-chave: Boxe, Golpes, Redes Neurais, Sensores Inerciais, *Android*, *Bluetooth*, Base *FIR*, Levemberg-Marquart

ABSTRACT

JERONYMO, Vitor, MULDER, Nicolas. DEVELOPMENT OF A PROTOTYPE FOR THE IDENTIFICATION OF BOXING MOVEMENTS. 88 f. – Engenharia Elétrica, Universidade Tecnológica Federal do Paraná. Curitiba, 2017.

Due the lack of devices that are capable of assist athletes on field during the practice of sports, it was decided to develop a prototype capable of count and identify the main boxing movements. The research presents a theoretical and empirical approach of inertial sensors, neural network knowledge, FIR base and basic boxing knowledge. The main goal of the project is to use inertial sensors inside boxing gloves to identify the types of punches that are being thrown by the athlete. During the project its obstacles and needs were discussed, like the sensor requirements, which punches to identify and the structure of the neural network. Since the chosen device, with inertial sensors, has Bluetooth connectivity, it was developed an Android application to perform the data acquisition of ideal movements and perform the simulation of the trained neural network . The data acquisition of the movements, those being the jab, cross, uppercut and hook, was performed using two professional boxing athletes with 14 years of experience, providing credibility to the data. At the end, the project brings results of the test sessions performed, presenting an 84,8% average rate of success on the identification of the movements.

Keywords: Boxing, Moves, Neural Networks, Inertial Sensors, Android, Bluetooth, FIR Base, Levemberg-Marquart

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

API	*Application Programming Interface*
FIR	*Finite Impulse Response*
MEMs	*Micro-Electro-Mechanical Systems*
MIMO	*Multiple Inputs Multiple Outputs*
MSE	*Mean Squared Error*
PMC	*Perceptron* Multi Camadas
RNA	Rede Neural Artificial
SDK	textitSoftware Development Kit

SUMÁRIO

1 INTRODUÇÃO

A realização de esportes é da natureza humana, movida em grande parte pelo espirito competitivo, estimulando o aprimoramento de características físicas e da técnica envolvida em sua prática (GALLEGOS, 2002). De acordo com este autor, esta tendência pela competitividade torna a busca pelo aumento de desempenho um dos principais objetivos dos atletas profissionais. Davey (2004) ressalta que uma fração de segundo na melhoria de uma atividade esportiva pode decidir quem sairá vitorioso. O bom desempenho de um atleta está diretamente relacionado, além de seu condicionamento físico, à técnica empregada na prática do esporte.

Qualquer que seja a modalidade esportiva, o objetivo central é sempre o mesmo: superar os limites do homem. Embora a otimização do rendimento esportivo seja fruto da combinação de fatores tão diversos como os genéticos e os sócio-afetivos, é inegável que a obtenção do máximo rendimento depende em grande parte da elaboração de estratégias de treinamento capazes de potencializar as capacidades e habilidades envolvidas no desempenho da modalidade (AMADIO, SERRÃO, 2011).

Tendo como foco o aprimoramento da técnica e a análise da *performance* dos atletas profissionais, é comum efetuar testes em ambiente de laboratório onde características como velocidade, força, posicionamento e execução da técnica são avaliadas. No entanto, acredita-se que, quando esta avaliação é feita fora do local usual de prática, os resultados adquiridos podem não condizer com a realidade.

“Testes em laboratório necessariamente impõem limites na *performance* do atleta já que o ambiente é suficientemente diferente do ambiente de treinamento ” (DAVEY, 2004).

Para obter resultados mais fieis, esta mensuração deve ser feita no ambiente de prática, tornando necessário o uso de tecnologias portáteis. Com o avanço no desenvolvimento e barateamento de microcontroladores, sensores inerciais e transmissores sem fio, o uso de uma plataforma de medição reduzida torna-se viável e pode vir a oferecer resultados mais adequados à mensuração da real *performance* dos atletas. “Sistemas microeletrônicos baseados em sensores inerciais como acelerômetros e giroscópios oferecem redução significativa no custo

e tamanho quando comparados aos tradicionais sensores mecânicos e ópticos" (GAFFNEY, WALSH, 2011). Outro ponto importante no desenvolvimento de plataformas de mensuração portáteis é o seu formato, peso, potência e outros aspectos que implicam na praticabilidade de seu uso. Por exemplo, equipamentos para o uso de natação devem ser impermeáveis e hidrodinâmicos, para corrida, leves e resistentes a transpiração e, no caso dos esportes baseados nas artes marciais, que será o foco deste trabalho, devem ter grande resistência à impactos e tamanho reduzido (DAVEY, 2004).

1.0.1 Delimitação do Tema

Atualmente existem poucas plataformas de medição voltadas para artes marciais e boxe, fazendo com que este tipo de análise dependa muitas vezes da experiência do treinador, estando sujeito a falhas de observação e julgamento (NAVAS, 2012). A partir deste pensamento, nota-se a necessidade do desenvolvimento de novas plataformas tecnológicas para auxiliar no treinamento desses esportes. Visando este auxílio, o estudo presente neste trabalho será focado no desenvolvimento de um protótipo de luvas de boxe dotadas de sensores e transmissores capazes de coletar informações sobre o movimento do lutador, assim como processar estes dados de forma a se obter uma representação da técnica utilizada. Os dados coletados pelos sensores a serem utilizados no protótipo serão enviados através do protocolo *Bluetooth* para um computador onde estes serão processados e analisados. Concordando com Mitchell, os sensores serão de natureza inercial, como acelerômetros e giroscópios, devido ao seu tamanho reduzido, baixo custo e precisão.

> Com o desenvolvimento recente de unidades de movimento inercial precisas sem fio e relativamente baratas, aliado aos algoritmos para determinar com mais precisão a orientação do sensor, tornou-se viável implantar redes de sensores vestíveis em sessões de treinamento (AHMADI, MITCHELL, 2014).

A análise inicial dos dados coletados será focada na identificação dos movimentos fundamentais do boxe através de um conjunto de Redes Neurais. O algoritmo, desenvolvido neste trabalho, será responsável por quantificar o número de golpes executados pelo boxeador assim como identificar estes movimentos. Uma vez que o algoritmo esteja calibrado e identifique com certa precisão os movimentos praticados pelo atleta, será iniciado o processo de verificação de sua escalabilidade para um sistema de análise de técnicas mais complexas.

1.1 PROBLEMAS E PREMISSAS

O Boxe se popularizou de forma profissional a partir de 1920 e seu objetivo principal é deferir o maior número de golpes no oponente e defender-se dos golpes recebidos até que o *round* se encerre ou o oponente seja nocauteado (DA COSTA, 2006).

Para alcançar esses objetivos, o boxeador deve bater com os punhos no adversário em seu plano frontal, acima da cintura. Durante o combate, é obrigatório o uso de luvas e o cumprimento das regras (SILVA NETO, 2013).

O atleta durante a luta também tem a possibilidade, dentro daquilo que é permitido pelas regras, de utilizar técnicas e estratégias que combinem os movimentos básicos do esporte como socos, defesas, movimento dos pés e posturas para derrotar seu adversário (SILVA NETO, 2013).

No plano teórico, um dos desafios deste trabalho é realizar o estudo de noções básicas do esporte. Conhecer seus fundamentos é necessário para tornar o desenvolvimento do projeto viável, pois a partir destes dados é possível definir parâmetros de escolha para os sensores e grandezas que serão associadas aos algoritmos de identificação de golpes.

Complementando as afirmações de Neto (2013), o autor Navas (2012) ainda ressalta que o boxe necessita da execução precisa de seus movimentos para maximizar o efeito dos golpes e que critérios como a velocidade, força e reflexo são componentes fundamentais para que isto ocorra. Tal afirmação demonstra a necessidade de que as medidas coletadas estejam bem refinadas, o que pode ser um desafio com sensores inerciais (AHMADI, MITCHELL, 2014). Acelerômetros podem apresentar ruídos induzidos durante o seu uso e giroscópios muitas vezes apresentam erros de *Bias offset* (BOUFFARD, 2016). No entanto, com o uso de ferramentas de filtragem e melhora de dados, como o filtro Kalman estendido (EKF), espera-se obter resultados suficientemente precisos.

Tendo em vista que a execução de golpes envolve a ação da musculatura de várias partes do corpo como mãos, braços, costas, abdômen, quadril e pernas (SILVA, 2014), para a mensuração de todos esses movimentos faz-se necessário a instalação dos sensores nesses locais.

Infelizmente, esta abordagem seria pouco viável devido ao grande número de sensores que seriam fixados no corpo do atleta, não apenas encarecendo o projeto, mas também prejudicando a *performance* do lutador. Levando isto em consideração, foi decidido que o lugar mais adequado para o posicionamento dos sensores seria nas luvas do boxeador (KAVANAGH,

2008).

Para definir a localização exata dos sensores é preciso ter noção de partes da luva que podem sofrer impacto, já que existe uma relativa fragilidade dos componentes eletrônicos utilizados. Uma das posições mais básicas de defesa no boxe é a postura ortodoxa (Figura 1), que requer que o pé esquerdo esteja mais avançado do que o direito, mão direita em contato com a bochecha direita e mão esquerda em contato ou próxima à bochecha esquerda (ISportBoxing 2016). Então, serão instalados os sensores na parte na luva que está voltada para o corpo nesta posição, diretamente oposta ao recebimento dos golpes (KAVANAGH, 2008).

Figura 1: Esboço da postura ortodoxa.

Fonte: Adaptado de
<http://golpedireto.blogspot.com.br/2013/06/boxe-postura-ortodoxa.html>

Após a aquisição dos componentes eletrônicos, o próximo desafio será programar a interação entre os equipamentos e a calibração dos sensores. Assim que os movimentos básicos de boxe forem corretamente identificados, será possível implementar mais funções, como sequências de técnicas, confirmação de golpe atingido, técnicas mais avançadas, entre outras ideias que poderão surgir no decorrer do trabalho.

1.2 OBJETIVOS

1.2.1 Objetivo Geral

Desenvolver um protótipo baseado em um conjunto de sensores inerciais inseridos em um par de luvas de boxe, para identificar e quantificar os movimentos do boxe, auxiliando a equipe técnica na condução do treinamento do seu atleta.

1.2.2 Objetivos Específicos

- Realizar um estudo básico sobre o boxe;
- Definir os sensores eletrônicos a serem utilizados;
- Programar na plataforma definida para refinar a medição dos movimentos e identificar os golpes;
- Estudar filtragem estocástica para aplicar um algoritmo de tratamento de dados ruidosos;
- Estudo de redes neurais;
- Desenvolver uma interface para o monitoramento dos dados.

1.3 JUSTIFICATIVA

O desenvolvimento de inovações tecnológicas nos esportes possibilita uma melhoria no desempenho dos atletas e na orientação feita por treinadores através de análises de desempenho e análise biomecânica dos movimentos (OKAZAKI, 2012). Similarmente, o protótipo possibilitará a aquisição de dados de movimentos e técnicas empregadas por lutadores de boxe, a fim de disponibilizar esses dados para análise e melhoria de seus desempenhos.

Para que o projeto seja viável, existe a necessidade de desenvolver o protótipo com equipamentos de baixo custo, obtendo resultados satisfatórios e prestando auxílio tecnológico ao treinamento de novos lutadores, como Navas denota.

> Existem muitas ferramentas para ajudarem lutadores a aprender mecânicas de socos e chutes do esporte, ataques e simulações defensivas e ofensivas para melhorar a resistência, mas raramente vemos aparelhos tecnológicos usados para prestar assistência aos lutadores durante o processo de aprendizagem(NAVAS, 2012)

A quantificação e identificação de golpes durante o treino podem servir como um mecanismo de apoio para o treinador. A existência de dados serve para evitar erros de julgamento e observação. Além disso, é possível manter um histórico dos treinos para consultas posteriores.

Outra vantagem neste tipo de abordagem do projeto é que as mensurações feitas serão em ambientes mais próximos aos de luta para garantir que não ocorra o problema citado por Davey: “Tradicionalmente, a medição de desempenho dos atletas de elite é comumente feito

em um ambiente laboratorial, entretanto, impõe limites na *performance* do atleta, porque o ambiente é diferente ao de treinamento." (James, Davey, Rice, 2004).

Muitas vezes os treinadores dispõem-se somente de suas experiências profissionais para intuitivamente julgar se o atleta está realizando seus movimentos corretamente (VERKHOSHANSKI, 1990 apud BORIN; GOMES; LEITE, 2007). De frente a estas considerações, o trabalho ao ser concluído serve como contribuição tecnológica às artes marciais, oferecendo suporte aos treinadores sobre a *performance* do atleta.

1.4 PROCEDIMENTOS METODOLÓGICOS

Para a construção do protótipo, foi feita uma pesquisa aplicada visando auxiliar o treinamento do boxe de forma quantitativa, identificando os golpes desferidos e suas quantidades.

Para atender os requisitos, foi executado um levantamento bibliográfico do esporte através de material publicado nacionalmente, internacionalmente e de sites das associações do esporte. Este levantamento foi feito a fim de entender a dinâmica do boxe, seus principais conceitos e variáveis de interesse para o projeto.

Definida a movimentação e grandezas de interesse, iniciou-se um levantamento sobre os sensores a serem utilizados no protótipo, características de seu funcionamento, pontos positivos e dificuldades no seu uso. Com estes aspectos avaliados, foi decidido qual conjunto de sensores seria mais adequado tecnicamente e financeiramente.

Para a identificação dos golpes, foram estudados os conceitos de redes neurais, sua aplicabilidade, princípios de funcionamento, algoritmos e formas de treinamento para assegurar que seu uso seria adequado para o projeto proposto.

Decidiu-se elaborar um teste de conceito para validar os levantamentos bibliográficos efetuados. Assim, definiu-se um golpe modelo, uma plataforma de execução de captura dos movimentos, plataforma de execução de código, de forma que os resultados pudessem ser avaliados graficamente.

Com o sucesso do teste de conceito, foi escolhido o dispositivo de coleta dos dados e, através das interfaces de programação disponíveis, definido o desenvolvimento de um aplicativo *Android* para a análise dos treinos. Com a aplicação desenvolvida, foram coletados dados dos diferentes tipos de golpes praticados no boxe e feito o treinamento das redes neurais que são utilizadas dentro do aplicativo. Com o aplicativo contendo as redes, foram efetuados testes para

verificar o nível de acertos do protótipo na identificação dos golpes.

1.5 ESTRUTURA DO TRABALHO

Este Trabalho de conclusão de curso (TCC) é composto por nove capítulos. O capítulo um definie o escopo do projeto, seus objetivos e motivações. O segundo capítulo apresenta a história do boxe, fundamentos e principais golpes. Em sequência, o terceiro e quarto capítulos apresentam, respectivamente, a teoria básica sobre sensores *Micro-Electro-Mechanical Systems (MEMs)*, suas aplicações e funcionamento e a teoria sobre redes neurais e sua aplicação no projeto.

O quinto capítulo apresenta um teste realizado para validar o projeto, utilizando sensores presentes em um aparelho celular e uma rede neural programada para a aplicação desejada. Neste capítulo definiu-se também o dispositivo utilizado para as medições. O sexto capítulo apresenta os próximos passos no desenvolvimento do aplicativo, a definição da plataforma de criação e suas funcionalidades. Com o aplicativo desenvolvido no sétimo capítulo, foi explicado o processo de treinamento da rede neural utilizada pela aplicação para identificar os movimentos de boxe. No oitavo capítulo foram feitos testes com o protótipo a fim de verificar sua precisão e limites de identificação.

O nono capítulo conta com a conclusão dos resultados obtidos até o presente momento e são definidos os projetos futuros para o protótipo.

2 O BOXE

Este item oferece uma base contextual sobre a história do boxe, seu desenvolvimento, posturas e movimentos básicos, a fim de auxiliar no processo de decisão dos golpes a serem identificados e entender as necessidades do usuário.

2.1 A HISTÓRIA DO BOXE

Os primeiros registros da prática do boxe datam em torno do ano 3000 A.C no antigo Egito e desde então existem diversas evidências de sua prática pela humanidade (ARUS, 2012). Entre os eventos históricos mais significativos do esporte na antiguidade estão a inclusão do boxe nos jogos olímpicos da antiguidade na Grécia, no final do século 7 A.C., e mais tarde a sua popularização na antiga Roma até a queda do império, fazendo com que o boxe caísse no esquecimento (ARUS, 2012).

O ressurgimento do esporte ocorreu no século XVII na Inglaterra, onde em 1681 a primeira luta oficial ocorreu. Nos anos seguintes, o esporte teve sua popularidade aumentada até que no ano de 1698 lutas periódicas começaram a ocorrer no *Royal Theater* em Londres, o que tornou a cidade o centro da prática do boxe (DALEY, 2014). Nas lutas até então, o boxe praticado não seguia as regras como conhecemos hoje, os lutadores não utilizavam luvas, não havia divisão de acordo com o peso, a luta terminava apenas quando um dos adversários estava impossibilitado de lutar e era permitido bater no oponente mesmo enquanto ele estivesse no chão, o que resultava em apenas um campeão por torneio (DALEY, 2014).

Com o aumento do número de torneios, o primeiro código de regras foi escrito em 1743 sem propor grandes mudanças no modo de combate praticado naquela época. Somente em 1838 um novo conjunto de regras chamado de *The London Prize Ring Rules*, foi criado. O foco do esporte então passou a ser o combate apenas com as mãos, proibindo golpes abaixo da cintura, não era permitido bater no oponente caído no chão e a luta continuava até que um dos lutadores fosse nocauteado ou não tivesse condições de entrar no ringue e se manter em sua posição. Paralelamente, foi inventada a primeira luva de boxe propriamente dita, cuja função

era proteger as mãos e rosto dos lutadores. No entanto seu uso não era obrigatório (SPOON, 2014).

Este novo código de regras tornou o boxe um esporte mais humanizado, porém as lutas ainda chocavam as classes mais ricas, o que trouxe em 1867 o *Queensberry Code of Rules* (DALEY, 2014). O novo regulamento trazia modificações em 5 áreas principais: o oponente deveria usar luvas estofadas, o *round* deveria durar 3 minutos, qualquer forma de luta que não utilizasse as mãos era proibida, qualquer boxeador que caísse no chão do ringue deveria ficar de pé em até 10 segundos ou seria declarado como perdedor e os lutadores passaram a ser divididos de acordo com o seu peso (MENDONZA, 2010).

O código de *Queensberry* foi a base do boxe conhecido atualmente, fazendo com que o esporte se desenvolvesse e rapidamente ganhasse popularidade ao redor do mundo, resultando em torneios e na criação de associações como a *Association Internationale de Boxe Amateur* (AIBA) (SPOON, 2014).

Em 1904, o boxe foi incluído nos jogos olímpicos que ocorreu em St. Louis nos Estados Unidos, tendo como o único país participante o próprio país. Nos anos seguintes, o número de países participantes aumentou e ao longo do tempo as regras foram adequadas até o regulamento olímpico atual (DALEY, 2014).

2.2 REGULAMENTO DO BOXE OLÍMPICO

As regras do boxe são diferentes para o boxe amador e profissional, e mesmo entre as diversas organizações em cada um dos dois.

No boxe profissional ganha o lutador que nocautear o adversário ou atingir o maior número de pontos em 12 *rounds*. Já no boxe amador ganha aquele que acertar o maior número de golpes perfeitos no adversário ou conseguir nocautear o oponente, no entanto, com um número de *rounds* menor.

Em geral o boxe amador possui 3 *rounds* com 3 minutos cada, como pode ser visto nas olimpíadas. Durante os *rounds* cada um dos lutadores é pontuado pelos jurados, podendo receber até 10 pontos se vencer a luta. Além do nocaute, são avaliados os golpes, dominância, competitividade, técnica e tática.

Visando auxiliar o treino e a avaliação de aspectos como técnica, número de golpes, que infere na dominância durante a luta, e sequências de golpe, este projeto foi desenvolvido.

2.3 FUNDAMENTOS DO BOXE

A técnica do boxeador é um dos elementos mais decisivos no boxe. Um ataque eficiente geralmente depende da habilidade de inferir um golpe forte e preciso na cabeça e corpo do oponente (ARUS, 2012).

Existem quatro habilidades motoras bem conhecidas para a prática do boxe, as quais incluem: velocidade, resistência, potência e flexibilidade (ARUS, 2012). Estas qualidades motoras podem ser subdivididas em:

- Velocidade da execução, deslocamento, tempo de reação;
- Habilidade;
- Força;
- Flexibilidade, referida as juntas (tendões e ligamentos).

Boxeadores sempre tentam manter seu centro de gravidade em sua postura, e o posicionamento das mãos geralmente é alto, possibilitando mudanças de direção (ARUS, 2012). Este posicionamento das mãos, aliado com a proximidade do oponente, faz com que as técnicas utilizem movimentos circulares em sua maioria (ARUS, 2012).

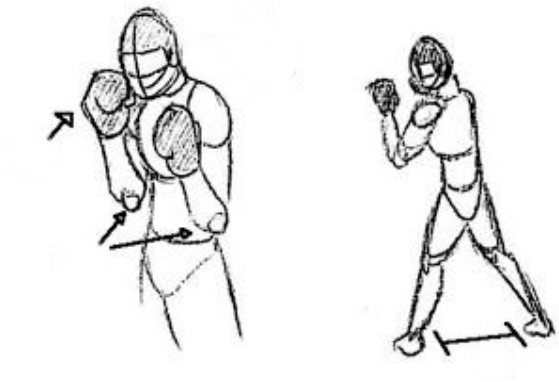
2.3.1 Posturas

Postura é o termo utilizado para definir a posição em que o lutador permanece no ringue e é a partir dela que a luta se inicia.

Existem duas posturas básicas de guarda que geralmente são adotadas pelos competidores: a postura ortodoxa e a postura chamada de "*Southpaw*". Na primeira, o boxeador permanece com seu pé esquerdo à frente do pé direito, o corpo ligeiramente virado para o lado, a mão esquerda a frente e a direita atrás junto ao queixo. Na segunda o posicionamento é inverso colocando o pé direito à frente do esquerdo e a mão direita à frente da esquerda. Ambas as posições oferecem equilíbrio entre defesa e ataque, diferenciando se o lutador vai inferir os golpes com a mão mais forte e defender com a mais fraca e vice-versa (MENDONZA, 2010).

Por ser mais utilizada, o projeto adotará como posição inicial a postura ortodoxa.

Figura 2: Postura Ortodoxa.



Fonte: Adaptado de
<http://golpedireto.blogspot.com.br/2013/06/boxe-postura-ortodoxa.html>

2.3.2 Golpes

A identificação dos golpes é a principal motivação do projeto e para que isso seja possível é necessário entender os fundamentos dos principais golpes no boxe.

Os golpes do boxe podem ser desferidos apenas do quadril para cima e para que sejam aceitos, os golpes devem ser aplicados na parte frontal do adversário, como no rosto e abdômen (SPOON, 2014).

No boxe, uma vez que um ataque é deferido, o receptor tem duas opções, defender ou contra atacar. Em outras palavras, existe uma constante troca de socos de ambos os lados (ARUS, 2012).

Existem quatro movimentos que são considerados como movimentos chave no boxe. Estes movimentos são chamados de *Jab*, Direto, Cruzado e *Uppercut* (ARUS, 2012).

Estes golpes aplicados sozinhos possuem um grande poder de ataque, no entanto, quando combinados, podem garantir o nocaute do adversário trazendo a vitória para o boxeador. Identificar estas sequências é mais uma aplicação possível do projeto em execução.

- *Jab*

  O *Jab* é um soco executado com o punho mais a frente do lutador, tomando como base a postura ortodoxa. Tipicamente, é um dos mais rápidos e longos socos e também um dos mais utilizados. No entanto, é um dos menos efetivos devido a sua baixa força. Em geral, o golpe é utilizado para medir a distância necessária para deferir um golpe mais forte no

oponente.

O soco pode ser executado de várias formas, dando um passo para frente, um passo para trás e com movimentos circulares da perna durante sua execução. Todas essas variações devem ser levadas em conta durante o treinamento da rede neural, pois impactam na velocidade angular dos movimentos.

Durante a execução, o cotovelo do braço mais a frente do lutador é elevado até a altura do peito para que seja mais eficaz. Tal movimento gera não apenas uma variação linear, mas também rotacional devido a flexão do ombro do lutador.

- Direto

  Geralmente utilizado após a execução de um *Jab*, este soco é considerado um ataque clássico do boxe.

  Este golpe é executado em uma linha reta com o antebraço, que estava inicialmente posicionado em frente ao queixo. Durante a execução do golpe, o lutador puxa o outro braço em frente ao seu queixo para proteção, proporcionando também a rotação do quadril e aumentando o poder do soco.

  Do ponto de vista da biomecânica, o cruzado, quando executado adequadamente, é mais forte que o *Jab*, pois nesta técnica o boxeador tem a habilidade de utilizar a rotação seu quadril e ombro mais eficientemente. Na maior parte das execuções, o pé traseiro do lutador está elevado e virado para fora, garantindo um maior movimento de rotação.

- Cruzado

  Considerado o golpe mais forte entre os quatro principais, se deferido na face do oponente, pode resultar em nocaute. O poder do golpe está associado com o braço que é utilizado. Quando deferido com o braço posterior, possui um poder de ataque menor devido a limitações na rotação do ombro e quadril.

  Golpes com o braço mais recuados são extremamente poderosos. Nesta técnica a rotação do ombro e quadril é facilmente executada adicionando poder ao soco.

  Devido às rápidas rotações, a velocidade angular deste golpe é bastante alta. Para calcular a velocidade angular do golpe, é possível adotar a luva do boxeador como a referência de rotação e o ombro como um eixo de rotação. O braço a executar o golpe inicia com uma dobra de aproximadamente 65 graus do antebraço, o qual pode esticar um pouco mais no final do golpe. Durante a execução, a parte superior do braço rotaciona horizontalmente em relação ao ombro e o antebraço rotaciona em relação a parte superior do braço. O

ângulo descrito entre o soco e o eixo do ombro é de aproximadamente 45 graus.

- *Uppercut*

  Assim como o cruzado, o *uppercut* também é um golpe semicircular deferido em sua maioria com o braço anterior na postura ortodoxa. Sua execução consiste na ascensão vertical de um soco.

  Para que o golpe seja efetivo, o lutador deve inclinar a parte central de seu corpo em direção ao braço executor dobrando seus joelhos de forma que, ao acertar o oponente, que os mesmos sejam estendidos conferindo mais força ao golpe.

  O *uppercut* possui técnicas similares ao gancho, já que ambos são baseados em movimentos semi circulares. Os dois diferem no sentido da variação angular, *uppercut* varia na vertical e ganho na horizontal, fazendo com que o *uppercut* enfrente a força da gravidade em sua subida.

Cada um dos golpes estudados possui características únicas capazes de os diferenciar uns dos outros. Como a base de todos os movimentos são alterações angulares, utilizar sensores inerciais para obter os perfis de velocidade angular é uma alternativa viável para classificar os movimentos.

É importante também considerar as principais variantes de cada um dos golpes mencionados acima para que a identificação destes seja correta.

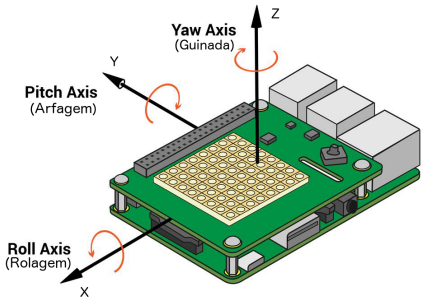
3 SENSORES

Para auxiliar na identificação dos golpes, o projeto prevê o uso de sensores mecânicos fabricados na escala de circuitos integrados. Estes sensores, conhecidos como MEMs (*Micro-Electro-Mechanical Systems*), são elementos mecânicos e eletromecânicos, miniaturizados fabricados através de técnicas de micro fabricação (CHRISTENSON, 2002), e apresentam baixo custo e escala. Sendo assim, estão entre os mais desenvolvidos durante os últimos anos (BARBOUR, 2001).

Para o projeto foram escolhidos três sensores inerciais: o acelerômetro, o giroscópio e o magnetômetro. Estes sensores serão responsáveis pela coleta de dados dos movimentos do lutador e serão utilizados na fusão sensorial para que seja possível a obtenção das componentes de orientação no espaço.

A fusão sensorial é a combinação de dados sensoriais ou dados derivados de fontes diferentes, de modo que a informação resultante tenha menos incerteza do que seria possível quando essas fontes fossem usadas individualmente (ELEMENREICH, 2002). Ela pode ser feita por algoritmos e métodos, como por exemplo, função de auto covariância, espectro de Fourier ou filtro de Kalman (MESTER, 2000).

Figura 3: Ângulos de rotação.



Fonte: Adaptado de
<https://www.raspberrypi.org/learning/astro-pi-guide/sensors/images/orientation.png>

Utilizando o filtro de Kalman serão obtidos os ângulos de Euler que expressam a orientação da luva por três ângulos de rotação, indicados na Figura 3: rolagem, arfagem e guinada em torno dos eixos x, y e z, respectivamente (KUIPERS, 1999)

Para compreender as grandezas fornecidas por cada um dos sensores escolhidos, seu funcionamento será explicado.

3.1 GIROSCÓPIOS

Giroscópios são sensores capazes de perceber mudanças na rotação e orientação sem um referencial fixo, assim, por consequência, podem ser utilizados para medir a velocidade angular.

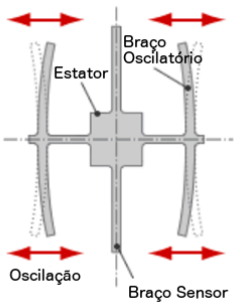
Existem diversos tipos de giroscópios de tamanhos e *performances* diferentes, no entanto, para o projeto, o giroscópio de interesse é o de vibração. A escolha foi baseada no seu extenso uso em micro eletrônicos.

> Nos últimos anos giroscópios vibracionais encontraram sua aplicabilidade em sistemas compactos de detecção de tremores em câmeras, percepção de movimentos em vídeo games, sistemas de controle eletrônicos de estabilidade entre outras aplicações (EPSON, 2013)

Giroscópios vibrantes medem a velocidade angular através da força de Coriolis aplicada em um elemento vibrante (GEEN, 2002). Esta força está presente quando um objeto encontra-se em movimento em relação a um referencial não inercial (CORBEN, 1960).

O sensor é constituído por uma parte fixa chamada de estator, onde braços sensores são fixados, e dois braços oscilatórios que irão vibrar com a rotação do componente como pode ser observado na Figura 4 (GEEN, 2002).

Figura 4: Estrutura de um giroscópio vibracional.

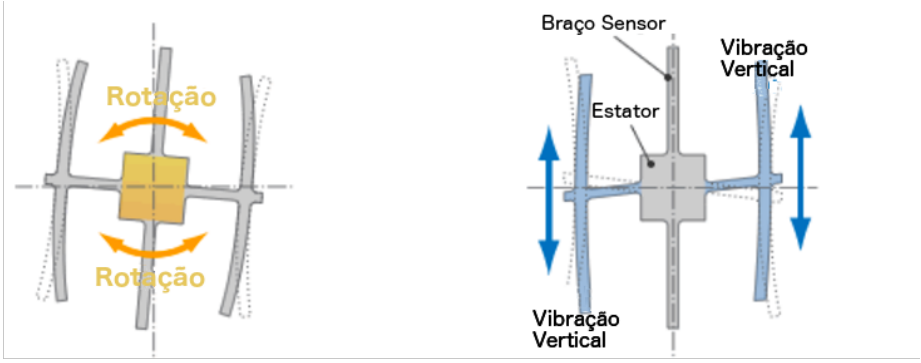


Fonte: Adaptado de <http://www5.epsondevice.com/en/information/technicalinfo/gyro/>

O braço oscilatório vibra em uma determinada direção, no entanto, uma vez que ocorre uma rotação, a força de Coriolis atua neste braço, produzindo vibrações no sentido vertical (GEEN, 2002).

Devido vibrações verticais, o braço fixo do sensor sofre pequenos dobramentos, gerando uma diferença de potencial que pode ser associada à velocidade angular sofrida pelo sensor (EPSON, 2013), como na Figura 5.

Figura 5: Estrutura de um giroscópio vibracional em movimento.



Fonte: Adaptado de <http://www5.epsondevice.com/en/information/technicalinfo/gyro/>

Pela integração da velocidade angular é possível achar a orientação, contudo, ao integrar a velocidade angular provinda do giroscópio, também integra-se o seu erro gerado devido ao efeito de escorregamento (TANENHAUS, 2012). Este efeito trata-se da alteração da medida do sensor durante o tempo, mesmo quando este permanece na mesma posição (TANENHAUS, 2012).

Devido à integração consecutiva dos erros vindos do escorregamento e ruídos do giroscópio, a orientação vinda do sensor possui um erro que cresce ao longo do tempo (TANENHAUS, 2012), não fornecendo dados adequados, quando sozinho para o projeto.

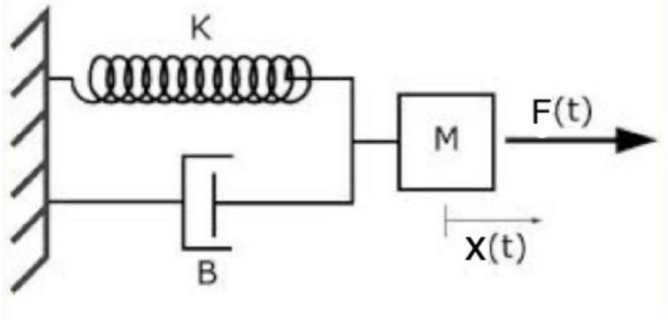
Para contornar o erro da orientação, ocasionado pela integração do escorregamento, é possível utilizar em conjunto um acelerômetro e magnetômetro, para que através do filtro de Kalman, o erro presente na orientação do giroscópio seja reduzido (LI, 2013).

3.2 ACELERÔMETRO

O acelerômetro é um sensor que converte a aceleração em um sinal elétrico de saída, podendo medir acelerações estáticas, como a gravidade, e dinâmicas como vibrações e movimentos (GRAHAM, 2000).

Seu funcionamento é baseado na segunda lei de Newton, através de medidas feitas em relação a uma massa de prova interna conectada por uma suspensão, formando um sistema massa-mola como na Figura 6.

Figura 6: Sistema massa-mola representando o princípio do acelerômetro.



Fonte: Adaptado de PUC-RJ(2014)

Quando uma força é aplicada ao sistema, a massa de prova sofre um deslocamento, provocando o estiramento ou compressão da mola. Esta força exercida pela mola é proporcional à força vinda da aceleração do corpo de prova (LEMKIN, 1997).

Para que seja possível utilizar acelerômetros em pequenos circuitos, a grande maioria destes equipamentos são construídos em microchips de silício com todos os seus componentes microscopicamente e quimicamente associados (CHRISTENSON, 2002).

Em geral estes sensores possuem placas capacitivas internas, algumas fixas e outras presas em molas microscópicas, podendo se mover com a ação de forças de aceleração. Enquanto uma placa se move em relação à outra, a capacitância entre elas muda, e é através dessas mudanças que a aceleração pode ser determinada (BEMHARD. 1997).

Com a popularização dos acelerômetros, a detecção de padrões de movimento através da aceleração se tornou um tópico de interesse de vários pesquisadores (BUJARI, 2012). Reconhecer movimentos cotidianos como atravessar uma rua, pegar um objeto e registrar quedas, já é uma realidade com o uso destes sensores (BAO, 2004)

É importante notar que apesar das aplicações feitas exclusivamente com o acelerômetro apresentarem bons resultados (BAO, 2004), quando efetua-se a avaliação de movimentos mais complexos, o acelerômetro pode fornecer dados imprecisos (BUJARI, 2012).

Tendo como base tais aplicações, este sensor foi escolhido para mensurar a magnitude da aceleração, que será utilizada como uma das entradas para a rede neural de identificação dos golpes, e também auxiliar na correção da orientação obtida pelo giroscópio.

3.3 MAGNETÔMETRO

O magnetômetro é um instrumento que mede o campo magnético e sua direção em um ponto do espaço. Este instrumento é amplamente utilizado para medir o campo magnético da Terra (EDELSTEIN, 2007).

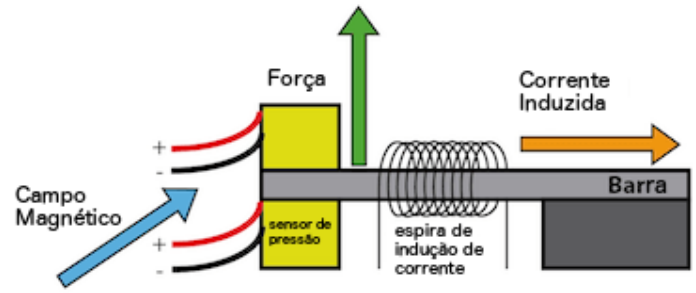
O campo magnético da Terra varia muito lentamente ao longo do tempo e espaço, podendo ser desconsiderado. Sendo assim, mudanças no campo magnético expressas no sistema de coordenadas da plataforma são associadas a mudanças na orientação do sensor (EL-DIASTY, 2014).

Magnetômetros MEMs são feitos de pequenas partes mecânicas com um eixo com sensores de pressão em ambos os lados e em uma espira com passagem de corrente em volta, como pode ser observado na Figura 7.

Quanto mais forte o campo magnético, ao qual o sensor está submetido, maior será o dobramento da barra percebido pelos sensores de pressão. Estes valores de pressão serão convertidos para valores de campo magnético (YONGYAO, 2012).

Este fenômeno ocorre devido à força de Lorentz que se trata da força aplicada em uma partícula pelo campo magnético quando a mesma possui circulação de corrente (THOMPSON, 2011).

Figura 7: Modelo de um magnetômetro MEMs.



Fonte: Adaptado de Thompson (2011).

Existem dois tipos básicos de magnetômetros: o que mede os componentes de um vetor de um campo magnético relativo à sua orientação no espaço e o que somente mede a magnitude do vetor de campo magnético total. Para a aplicação deste projeto, o magnetômetro vetorial será utilizado, pois é necessário obter as componentes do campo magnético para poder dimensionar a orientação corretamente (LI, 2013).

As medidas do acelerômetro e magnetômetro podem ser integradas para gerar um

compasso eletrônico que consegue gerar as componentes de orientação a partir da gravidade e campo geomagnético, sem produzir erros de derrapagem mesmo em longos períodos de tempo (LI, 2013).

4 REDES NEURAIS

A Rede Neural Artificial (RNA) é um algoritmo computacional que se assemelha às estruturas neurais biológicas e tem o objetivo de modelar a maneira que o cérebro processa uma determinada tarefa. As RNAs têm a capacidade de aprender a partir de entradas e saídas conhecidas para poder generalizar saídas razoáveis a partir de entradas desconhecidas, diferentes das estabelecidas no treinamento. Geralmente as RNAs são criadas usando componentes eletrônicos ou simuladas utilizando computadores com capacidade de processamento necessária (BISHOP, 1995).

A computação por RNA vem surgindo com aplicações reais em diversas áreas, sendo elas: robótica, controle de processos, aproximação de funções matemáticas, química quântica, previsão de tempo, trajetória de objetos, tomada de decisão, reconhecimento de padrões, reconhecimento de sequências, diagnósticos médicos, aplicações financeiras, previsão de números sorteados da loteria, entre muitas outras (BISHOP, 1995).

4.1 HISTÓRIA

Historicamente, as primeiras pesquisas sobre RNAs e o primeiro modelo de neurônio artificial surgiram em 1943, pelo neurofisiologista Warren McCulloch e o matemático William Pitts no artigo "*A logical calculus of the ideas immanent in nervous activity*" que uniu os estudos de neurofisiologia e lógica matemática (HAYKIN, 2004).

Seis anos depois, em 1949, houveram avanços consideráveis no conhecimento de Redes Neurais com a publicação do livro "*The organization behaviour*" pelo psicólogo Donald Hebb. Ele defendia que os ramos neurais ficam mais fortes cada vez que são usados, um conceito fundamental da maneira em que o ser humano aprende (HEBB, 1949). O livro tem importância porque foi uma fonte de inspiração para o desenvolvimento de modelos computacionais de sistemas de adaptação e de aprendizado (HAYKIN, 2004).

Baseado no modelo de neurônio de McCulloch, em 1958, Rosenblatt criou a forma

de rede neural mais simples, utilizada para a classificação de padrões linearmente separáveis, chamada de *perceptron*. Em 1960, Rosenblatt propôs o Teorema de Convergência *perceptron*, que prova a convergência do modelo de aprendizado supervisionado criado para a sua rede neural (KRIESEL, 2007).

Ainda em 1960, Bernard Widrow e Marcian E. Hoff criaram a Adaline (*adaptive linear element*) que é uma rede neural com sistema de aprendizado adaptável, rápido e preciso. A Adaline foi a primeira rede neural a ser comercialmente utilizada, podendo ser encontrada em quase todos os telefones analógicos para o filtro de eco. A diferença do *perceptron* para a Adaline é o procedimento de treinamento. Posteriormente, Widrow propôs uma das primeiras redes neurais em camadas treináveis com múltiplos elementos adaptativos, a estrutura Madaline, que é composta de múltiplas Adalines (KRIESEL, 2007).

A pesquisa da rede neural estagnou após a publicação da pesquisa de aprendizagem de máquina por Marvin Minsky e Seymour Papert em 1969, que descobriram dois problemas principais com as máquinas computacionais que processam redes neurais. O primeiro era que os *perceptrons* básicos eram incapazes de processar o circuito lógico OU exclusivo (XOR). O segundo foi que os computadores não tinham poder de processamento suficiente para lidar com a complexidade das redes neurais com múltiplas camadas. Então, a pesquisa em redes neurais diminuiu até que os computadores obtivessem maior poder de processamento (MINSKY, 1969).

Em 1975, com contribuições de diversos autores, foi criado o algoritmo *backpropagation* (retropropagação) que foi capaz de solucionar os problemas mencionados anteriormente. Assim, pôde-se dar continuidade às pesquisas com redes neurais (HAYKIN, 2004).

4.2 FUNCIONAMENTO DA REDE NEURAL

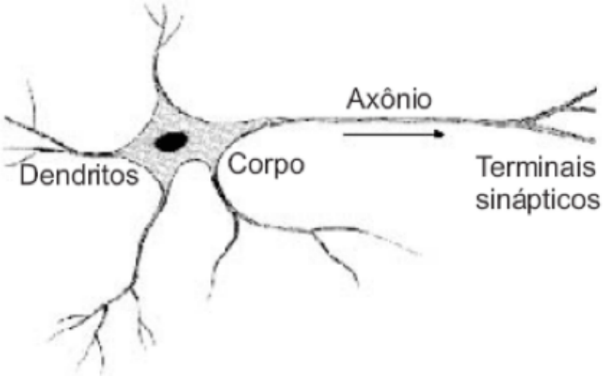
Para entender melhor o funcionamento da RNA utilizada no projeto, é necessário compreender alguns conceitos básicos como a estrutura dos neurônios biológicos e artificiais, as funções de transferência e sigmoidal e o funcionamento do *perceptron*.

4.2.1 Neurônio Biológico

Os neurônios biológicos consistem de dendritos, que recebem conexões de milhares de neurônios, e axônios, que transmitem a informação para outros neurônios, como mostra a Figura 8. Se houver estímulo de impulsos nervosos suficientes nos dendritos para ativar o potencial de

ativação do neurônio, haverá emissão de uma resposta em forma de pulso elétrico nos terminais dos seus axônios, propagando a informação para os próximos neurônios por meio de sinapses (GURNEY, 1997).

Figura 8: Modelo do neurônio biológico.

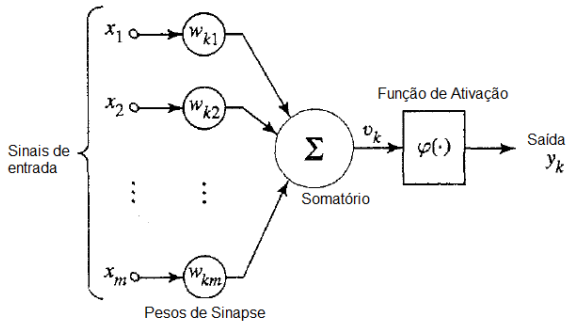


Fonte: Adaptado de
<http://www.scielo.br/scielo.php?script=sci-arttextpid=S0100-19652006000100003>

4.2.2 Neurônio Artificial

Similarmente ao neurônio biológico, o neurônio artificial possui o mesmo objetivo, que é receber, processar e propagar sinais. Seu modelo pode ser representado pela Figura 9. Os sinais de entrada x devem ser multiplicados por pesos sinápticos w_k, que são obtidos através do treinamento do neurônio. Estes pesos representam o nível de influência que as entradas possuem sobre a saída do neurônio. Após isto, a entrada x_i, $0 \leq i \leq m$, modificada, irá passar por uma função somatório como mostra a equação (1). Em seguida, o resultado do somatório se submete à uma função de ativação φ que produz a saída y_k, conforme a equação (2) (HAYKIN, 2004).

Figura 9: Modelo do neurônio Artificial.



Fonte: Adaptado de Haykin (2004).

$$v_k = \sum_{j=0}^{m} w_{kj} x_i \tag{1}$$

$$y_k = \varphi(v_k) \tag{2}$$

O neurônio da RNA, após ser treinado, é capaz de processar uma ou mais saídas a partir de suas entradas. O algoritmo de treinamento da RNA geralmente necessita de um banco de dados contendo entradas e saídas previamente conhecidas para conseguir estabelecer os pesos sinápticos dos neurônios que serão atribuídos à camada de entradas (GURNEY, 1997).

4.2.3 Funções de transferência (ou funções de ativação)

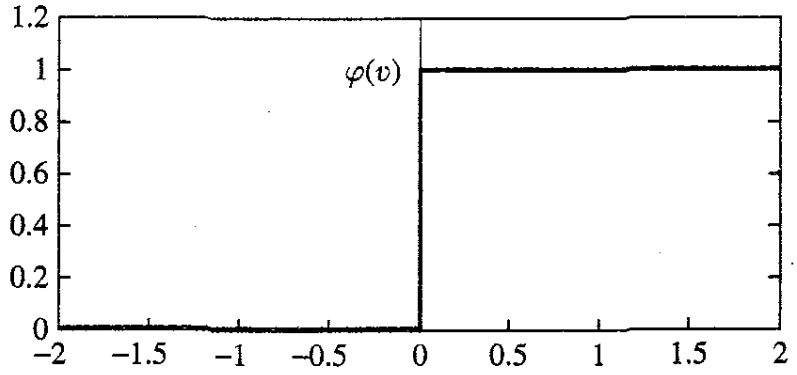
As funções de transferências analisam o somatório de um grupo de entradas para poder determinar uma saída correspondente (GURNEY, 1997).

Um exemplo de função de transferência é a função limiar (ou *threshold*). Essa função se assemelha à função degrau e pode ser representada pela Figura 10 (GURNEY, 1997). Para função de transferência $\varphi(v)$, se o valor de v for menor que 0, o resultado será 0. Se v for maior que 0, o resultado será 1 (HAYKIN, 2004).

$$\varphi(v) = 1 \; se \; v \geq 0 \tag{3}$$

$$\varphi(v) = 0 \; se \; v < 0 \tag{4}$$

Figura 10: Função Limiar.



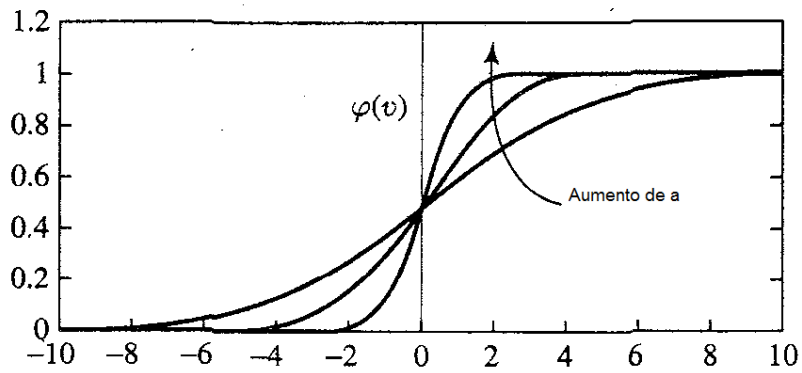
Fonte: Adaptado de Haykin (2004).

A função de transferência a ser utilizada neste projeto será a de logística sigmoidal. Esta função possibilita uma precisão melhor, pois assume valores contínuos entre 0 e 1,

possibilitando controlar a inclinação de sua assíntota com a variação do parâmetro *a* (HAYKIN, 2004). É dada pela equação (6) e pode ser exemplificada pela Figura 11.

$$\varphi(v) = \frac{1}{(1+e^{-av})} \tag{5}$$

Figura 11: Função logística sigmoidal.



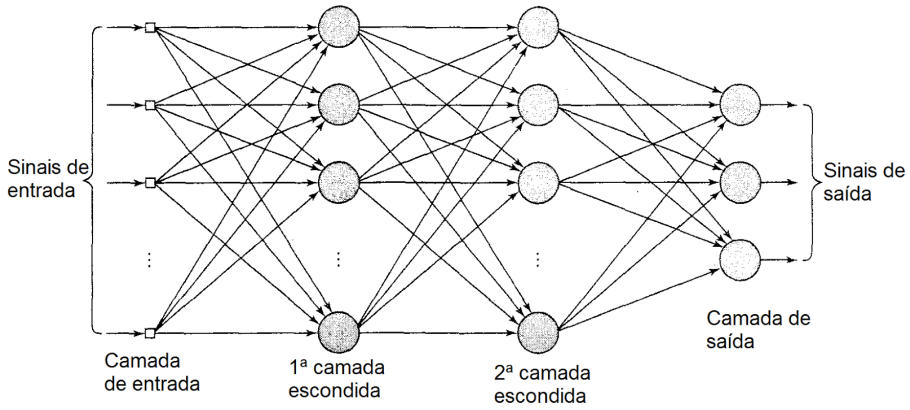
Fonte: Adaptado de Haykin (2004).

4.3 PERCEPTRON MULTI-CAMADAS (PMC)

Baseada na rede do *perceptron* de uma camada, desenvolvida por Rosenblatt, a rede do tipo PMC se caracteriza pela presença de múltiplas camadas de neurônios *Perceptron* interconectadas entre si. Diferentemente da *perceptron* de uma camada, ela é útil na resolução de problemas não lineares e reconhecimento de padrões e classes. Como a Figura 12 mostra, esta RNA consiste de uma camada de sinais de entrada, uma camada de neurônios de saída e entre elas podem existir inúmeras camadas intermediárias (ou camadas escondidas), que, através do *backpropagation*, reconhecem combinações de entradas mais significantes (TOURETZKY, 1989). Cada camada escondida pode conter inúmeros neurônios, sendo que as saídas dos neurônios de cada camada são as entradas dos neurônios da camada posterior. A camada de saída deve conter um mesmo número de neurônios para o mesmo número de sinais de saída.

Inicialmente, os valores dos pesos sinápticos são inicializados com valores randômicos para posteriormente serem ajustados pelo método de retropropagação ou *backpropagation*. Com sinais de entrada e saída conhecidos, este método consegue determinar o erro de saída da rede comparando os sinais de saída simulados com os reais. Assim, ocorre a retropropagação

Figura 12: Multi-Layer Perceptron.



Fonte: Adaptado de Haykin (2004).

do erro pela rede para reajustar os valores dos pesos sinápticos de todos os neurônios, a fim de que na próxima simulação o erro de saída da rede seja menor. Este processo é repetido inúmeras vezes até que se chegue em um valor de erro aceitável.

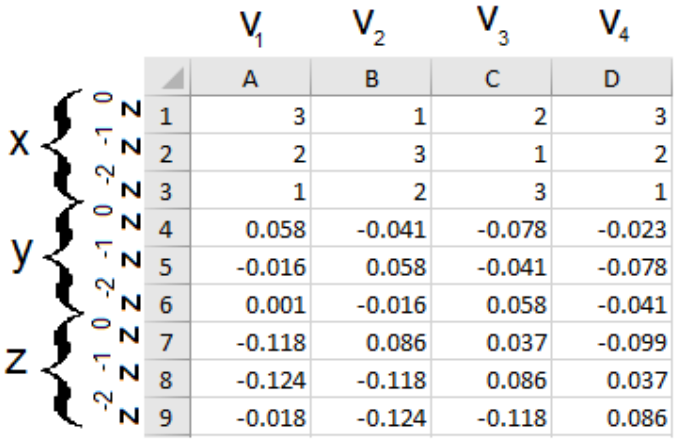
4.3.1 Modelo *Finite Impulse Response* (FIR)

Para ser possível o reconhecimento da variação dos dados dos sensores em relação ao tempo é importante que os sinais de entrada da RNA não sejam instantâneos, mas discretos em relação ao tempo. Então, utiliza-se a base *FIR* que se baseia no filtro *FIR*. O valor de saída do modelo *FIR* de uma ordem N retorna o somatório das entradas $x[n-i]$ mais recentes multiplicadas por um certo peso b_i. Na RNA, esse somatório é feito nos neurônios da primeira camada escondida.

$$y[n] = b_0x[n] + b_ix[n-1] + b_2x[n-2]... + b_Nx[n-N] = \sum_{i=0}^{N} b_ix[n-i] \tag{6}$$

Para alimentar a rede neural, é necessário que as amostras coletadas pelos sensores sejam atrasadas e armazenadas em um vetor dependendo da ordem do modelo *FIR*. Por exemplo, para um modelo *FIR* de ordem dois e amostras coletadas "1 2 3 1 2 3", a uma taxa de amostragem de 20Hz (período de 0,05 segundos), tem-se quatro vetores de amostras com elementos Z^{-2},Z^{-1},Z^0, sendo eles $X1 = [1\ 2\ 3], X2 = [2\ 3\ 1], X3 = [3\ 1\ 2]$ e $X4 = [1\ 2\ 3]$. Dentro dos neurônios da primeira camada escondida é realizado o somatório do modelo *FIR* $\sum_{i=0}^{2} b_i.x[3-i]$ para cada vetor de amostras. Cada vetor tem o histórico de variação de amostras que alimentarão uma RNA treinada que irá indicar se essa variação corresponde a

Figura 13: Vetores de amostras no modelo *FIR*.



Fonte: Autoria Própria.

uma identificação positiva ou não. Supondo que essas seis amostras são do eixo x de um acelerômetro, para incluir as amostras dos eixos y e z, repete-se o mesmo processo. Se uma RNA for alimentada com essas três entradas para cada eixo, sendo no total nove elementos de entrada, ela poderá reconhecer se aquela sequência de variação de aceleração nos três eixos corresponde à um padrão desejado em períodos de $0,15$ segundos $(0,05.3)$. Os vetores podem ser exemplificados na Figura 13.

4.3.2 Algoritmo de treinamento

Para definir os pesos da rede neural, é necessário utilizar ferramentas de aproximação de funções que sejam capazes de aproximar as saídas da rede neural das saídas reais com um valor de erro aceitável (ARIF, 2009). Dentre os vários algoritmos existentes no treinamento de redes neurais, o de Levenberg-Marquardt tem sido considerado como um dos mais efetivos (MOTA, 2004).

O algoritmo de Levenberg-Marquardt foi desenvolvido pelos estatísticos Kenneth Levenberg e Donald Marquardt para soluções numéricas de problemas de funções não lineares. Baseado no método dos mínimos quadrados, foi comprovado que sua implementação em redes PMCs apresenta um desempenho de 10 a 100 vezes mais rápido do que o algoritmo de *backpropagation* convencional (HAGAN, MENHAJ, 1994).

A partir de uma função de erros, como o erro médio quadrático, entre o valor de saída simulado e o valor de saída conhecido, o algoritmo de *backpropagation* consegue reajustar cada

peso da RNA equacionando a variação dos erros em função da variação dos pesos por derivadas parciais. Como o *backpropagation*, o método de Levenberg-Marquardt também necessita de entradas e saídas conhecidas para poder gerar um vetor E com os erros totais. A partir de pesos sinápticos aleatórios, ocorrem os seus redimensionamentos, que se dão pela equação a seguir:

$$\Delta W = (J^T(W)J(W) + \mu I)^{-1} J^T(W)E, \tag{7}$$

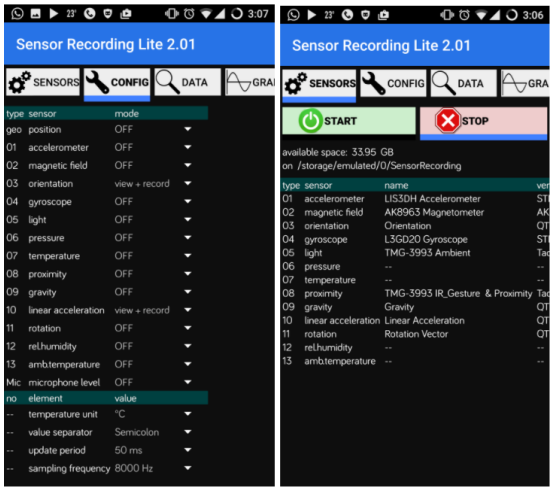
sendo ΔW o reajuste de pesos, $J(W)$ a matriz Jacobiana de derivadas do erro em relação ao peso, I é a matriz identidade μ, é a constante de aprendizagem e E é o vetor de erros. Obtendo-se novos valores dos pesos, simula-se a rede novamente para observar o comportamento dos erros. Se o erro aumentar, resetam-se os pesos para valores antes do reajuste feito e aumenta-se a constante μ para ser feito novamente a iteração. Se o erro diminuir, aceitam-se os novos pesos e para a próxima iteração, diminui-se a constante μ (YU,2011). É feita a iteração inúmeras vezes até se chegar a um valor de erro desejado.

5 TESTES DE CONCEITO

Com o objetivo de comprovar a viabilidade do projeto, foi proposto um plano de teste utilizando sensores inerciais de um aparelho celular. O telefone utilizado é da marca *OnePlusOne* modelo A0001 que possui o acelerômetro LIS3DH, giroscópio L3GD20 e magnetômetro AK8963. Para a aquisição de dados dos sensores do celular, foi usado o aplicativo *Sensor Recording Lite 2.01* por Michael L. Braun na plataforma *Android*.

O aplicativo permite definir uma taxa de amostragem dos dados e selecionar os sensores desejados para serem registrados, sendo eles: posição por GPS, acelerômetro, magnetômetro, orientação, giroscópio, luminosidade, pressão, temperatura, proximidade, aceleração da gravidade, aceleração linear (sem componente gravidade), rotação, humidade relativa, temperatura ambiente e microfone como mostra a Figura 14.

Figura 14: Aplicativo *Sensor Recording Lite 2.01*.



Fonte: Autoria Própria.

É de interesse do teste de conceito somente a aceleração linear nos três eixos, x, y e z, e as componentes de orientação *pitch* e *roll*. Estas componentes são utilizadas para definir os ângulos de rotação em torno de um eixo, *pitch* em torno do eixo y e *roll* em torno do eixo

x (LABAYRADE, 2003). É importante ressaltar que esses dados escolhidos que o aplicativo fornece já são tratados, ou seja, passam por algoritmos que diminuem o ruído do sensor e fazem a fusão sensorial para se obter as componentes de orientação.

Foi empiricamente definido a taxa de amostragem em 20Hz (período de 50 milissegundos). Com o aplicativo configurado adequadamente, foi possível iniciar a coleta dos dados.

O movimento de interesse foi um soco que consiste em segurar o celular com a mão, estender a mão para frente e em seguida retornar à posição anterior, tentando gerar variação de aceleração somente em um eixo e não causar variações na inclinação em nenhum eixo.

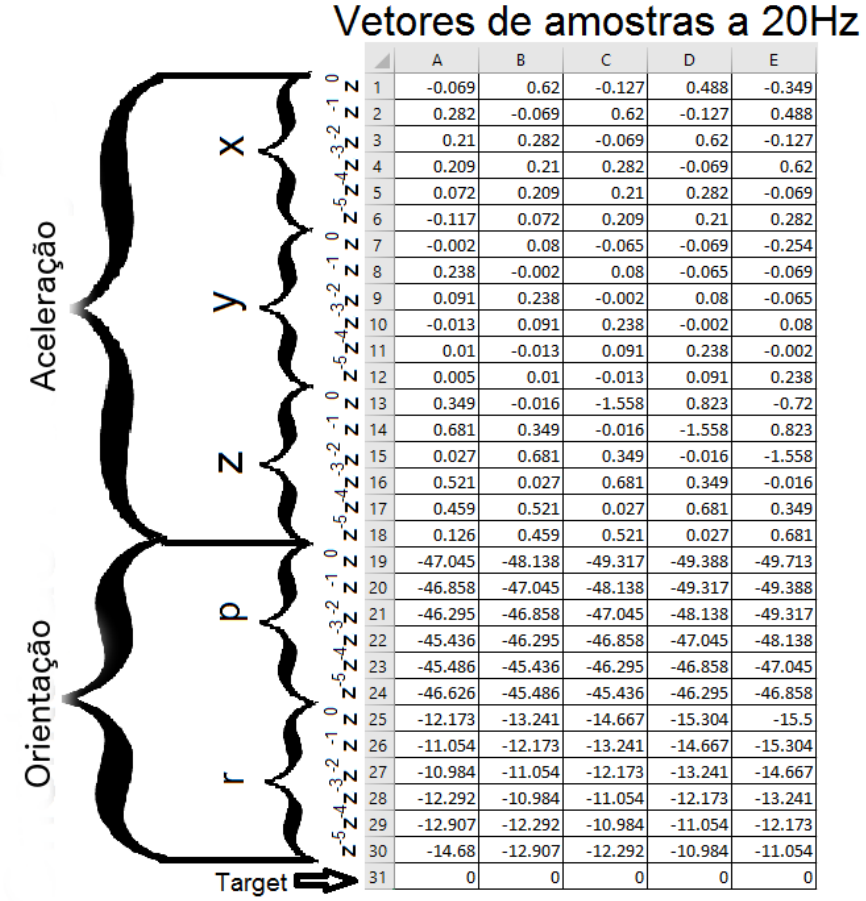
Para o treinamento da rede neural, são necessárias entradas e saídas correspondentes para que posteriormente possa ser realizado o treinamento dos pesos pelo método de Levenberg-Marquardt com função de erro dada pelo erro médio quadrático. Assim, foram registrados os dados dos sensores para se criar dois bancos de amostras que serão os elementos de entrada de treinamento. No primeiro banco, foram armazenados os dados dos movimentos que não correspondem à um soco, ou seja, vários tipos de movimentos aleatórios, bruscos e lentos. Já no segundo, os dados dos movimentos que correspondem ao movimento desejado, que seria o golpe "perfeito", neste caso a extensão da mão e seu retorno a posição inicial. O aplicativo foi utilizado em sua versão grátis, em que o registro de dados é limitado para seções de um minuto.

Ao final de cada sessão de coleta de dados o aplicativo gera um arquivo de *log* no formato *.csv* que, após ser transferido para o computador, pode ser analisado com o *software* MATLAB.

Como um golpe pode ter duração de períodos tão rápidos como 0,3 segundos (ARUS, 2012), é importante que o período do modelo *FIR* não ultrapasse esse tempo. Então, para uma frequência de 20Hz, o modelo deve possuir ordem cinco, a fim de que cada vetor de entrada da rede possa registrar um período de 0,3 segundos.

A fim de ser possível gerar o modelo *FIR* de uma ordem qualquer, foi desenvolvido um código na linguagem MATLAB capaz de gerar uma base de ordem *n* para as amostras de cada componente de cada sensor. O código também cria um vetor de saída (*target*) com valores que dependem do banco de dados usado, presente no Apêndice A. Para o primeiro banco de dados, o vetor *target* é preenchido com zeros e, para o segundo, com uns. Utilizando-se desse código, foi possível criar a matriz de treinamento da rede neural, e parte dela pode ser exemplificada na Figura 15.

Figura 15: Matriz de treinamento da RNA.



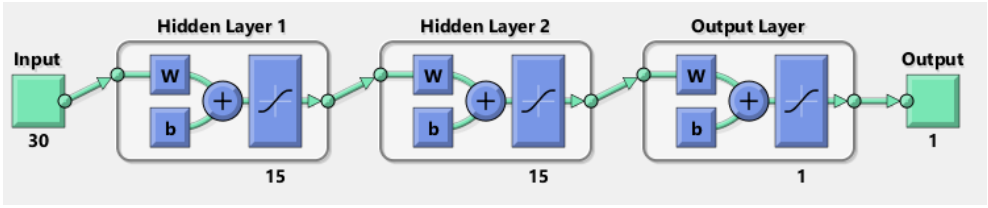
	A	B	C	D	E
1	-0.069	0.62	-0.127	0.488	-0.349
2	0.282	-0.069	0.62	-0.127	0.488
3	0.21	0.282	-0.069	0.62	-0.127
4	0.209	0.21	0.282	-0.069	0.62
5	0.072	0.209	0.21	0.282	-0.069
6	-0.117	0.072	0.209	0.21	0.282
7	-0.002	0.08	-0.065	-0.069	-0.254
8	0.238	-0.002	0.08	-0.065	-0.069
9	0.091	0.238	-0.002	0.08	-0.065
10	-0.013	0.091	0.238	-0.002	0.08
11	0.01	-0.013	0.091	0.238	-0.002
12	0.005	0.01	-0.013	0.091	0.238
13	0.349	-0.016	-1.558	0.823	-0.72
14	0.681	0.349	-0.016	-1.558	0.823
15	0.027	0.681	0.349	-0.016	-1.558
16	0.521	0.027	0.681	0.349	-0.016
17	0.459	0.521	0.027	0.681	0.349
18	0.126	0.459	0.521	0.027	0.681
19	-47.045	-48.138	-49.317	-49.388	-49.713
20	-46.858	-47.045	-48.138	-49.317	-49.388
21	-46.295	-46.858	-47.045	-48.138	-49.317
22	-45.436	-46.295	-46.858	-47.045	-48.138
23	-45.486	-45.436	-46.295	-46.858	-47.045
24	-46.626	-45.486	-45.436	-46.295	-46.858
25	-12.173	-13.241	-14.667	-15.304	-15.5
26	-11.054	-12.173	-13.241	-14.667	-15.304
27	-10.984	-11.054	-12.173	-13.241	-14.667
28	-12.292	-10.984	-11.054	-12.173	-13.241
29	-12.907	-12.292	-10.984	-11.054	-12.173
30	-14.68	-12.907	-12.292	-10.984	-11.054
31	0	0	0	0	0

Fonte: Autoria Própria.

Como a rede neural recebe 30 elementos de entrada, empiricamente adotou-se a configuração da rede por duas camadas escondidas, com 15 neurônios cada, e uma camada de saída que contém apenas um neurônio. A configuração da RNA simulada pode ser mostrada na Figura 16.

Após o treinamento da RNA, foram gravadas, com o auxílio do aplicativo presente no Apêndice B, duas seções de 1 minuto cada, em que nos primeiros 30 segundos foram feitos movimentos aleatórios, bruscos e lentos e, em seguida, em intervalos de 10 segundos foram feitos os golpes nos quais a rede foi treinada para identificar. Após simular os dados na rede

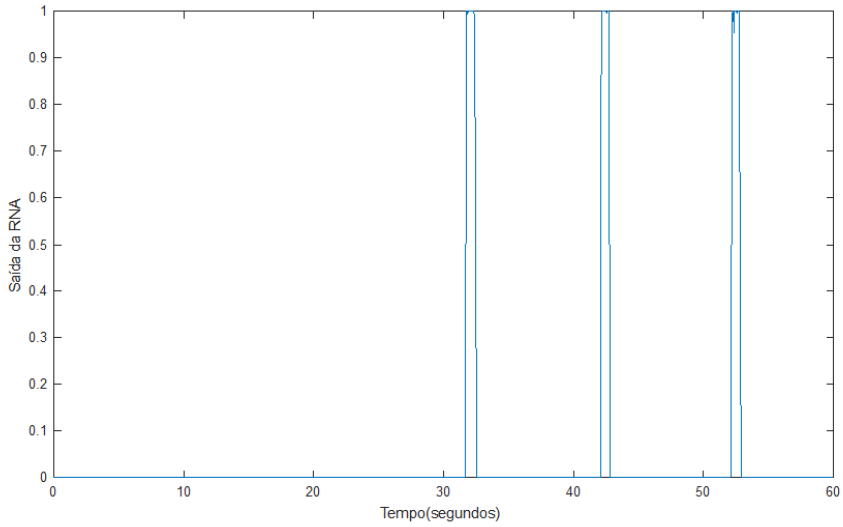
Figura 16: Configurações da RNA.



Fonte: Autoria Própria.

neural, foi possível produzir os gráficos das Figuras 17 e 18.

Figura 17: Simulação 1.



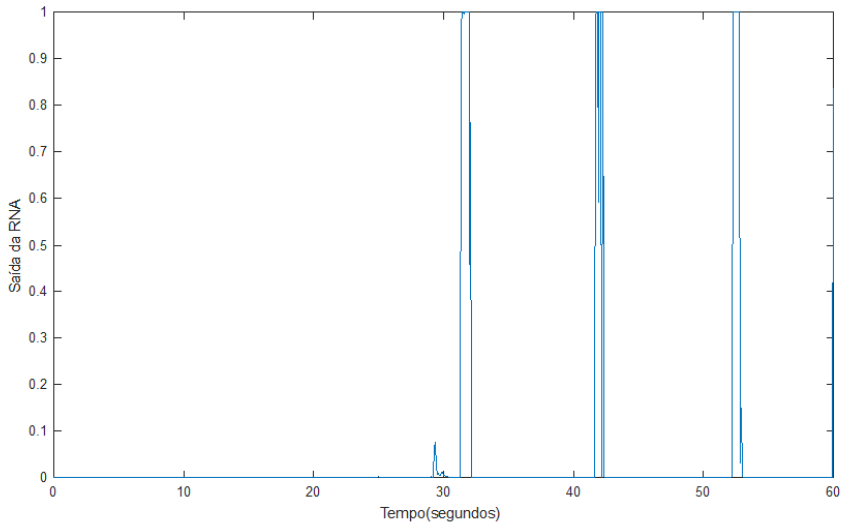
Fonte: Autoria Própria.

É possível verificar que nas duas simulações realizadas, entre 0 e 30 segundos, a rede não detectou o movimento de soco e nos instantes 32, 42 e 52s, o golpe foi identificado. Portanto, conclui-se que a rede neural conseguiu discernir corretamente os movimentos aleatórios do golpe proposto.

5.1 DISPOSITIVO DE MEDIÇÃO

Com os resultados obtidos no teste de conceito, foram definidas as características necessárias dos sensores para a escolha do dispositivo a ser implementado. Ele deve conter sensores do tipo acelerômetro, giroscópio e magnetômetro, é desejável a presença de um

Figura 18: Simulação 2.



Fonte: Autoria Própria.

algoritmo de fusão sensorial integrado e a taxa de amostragem deve ser de pelo menos 20Hz. Para efetuar o *download* dos dados coletados, optou-se pelo protocolo *Bluetooth 4.2* devido ao seu baixo consumo de energia. O *Bluetooth 4.2* é a versão mais avançada até hoje, consome menos energia e tem taxa da transferência de dados 2,7 vezes maior do que a versão anterior (BLUETOOTH, 2014). Fisicamente o dispositivo deve ser de pequenas dimensões para que seja colocado dentro das luvas de boxe.

Baseado nestes requisitos, o dispositivo *MetaMotion R* da empresa *MBIENTLAB* foi escolhido. Dentro do dispositivo encontram-se diferentes sensores e funções de processamento como:

- BMI160 que contém acelerômetro e giroscópio;
- BMM150 com magnetômetro;
- BMP280 contendo barômetro e termômetro que não serão utilizados;
- Fusão sensorial por filtro de Kalman;
- Calibraçao do *offset* com monitoramento de saidas.

Além de possuir os sensores desejados para o projeto, o *MetaMotion R* conta também com um algoritmo de fusão sensorial. Este combina as medidas nos 3 eixos do giroscópio,

do magnetômetro e do acelerômetro para fornecer um vetor de orientação robusta na forma de ângulos de Euler. Os dados obtidos através da fusão sensorial também são utilizados para melhorar as saídas individuais dos sensores (MBIENTLAB, 2017).

As informações coletadas podem ser registradas em sua memória interna de 2Mb, a uma taxa de 800Hz, ou serem transmitidas diretamente via *Bluetooth*, a uma taxa de 100Hz.

A MBIENTLAB disponibiliza o aplicativo *Metabase*, através do qual é possível conectar diferentes dispositivos e coletar dados dos sensores. No entanto, suas configurações para a captura de dados é bastante limitada, tornando seu uso inviável para a luva inteligente.

Para aplicações mais específicas a empresa disponibiliza uma *Application Programming Interface (API)* para a plataforma *Android*, possibilitando o desenvolvimento de aplicativos com configurações de captura de dados mais complexas.

Um *API* é um conjunto de instruções e rotas capazes de se comunicar com o *hardware* para o qual foi desenvolvido, seu uso possibilita interagir com os diversos aspectos do dispositivo em uma programação de nível mais alto (ORENSTEIN, 2000).

Assim, utilizando o *MetaWear Android API*, foi decidido desenvolver um aplicativo para a plataforma *Android* capaz de se comunicar com o *MetaMotion R*, configurar seu sensores, gravar seus dados e então tratar e processar as informações para a identificação dos golpes de boxe.

6 DESENVOLVIMENTO DO APLICATIVO

Para que a análise seja efetuada no ambiente de treinamento, é necessário o uso de um dispositivo portátil que promova a avaliação do treino no local de sua prática. Buscando uma solução uso e operação intuitiva, optou-se pelo desenvolvimento de um aplicativo para a plataforma *Android* devido a sua grande presença no mercado de *smartphones*.

De acordo com os números divulgados pela consultoria IDC, 95,5% dos aparelhos comercializados entre julho e setembro de 2016 executavam o sistema operacional do Google (IDC, 2016).

6.1 FUNCIONAMENTO DO APLICATIVO

O aplicativo é responsável por controlar as luvas inteligentes através da conexão *Bluetooth*, e ao final do treinamento, efetuar o *download* destes dados para o seu processamento.

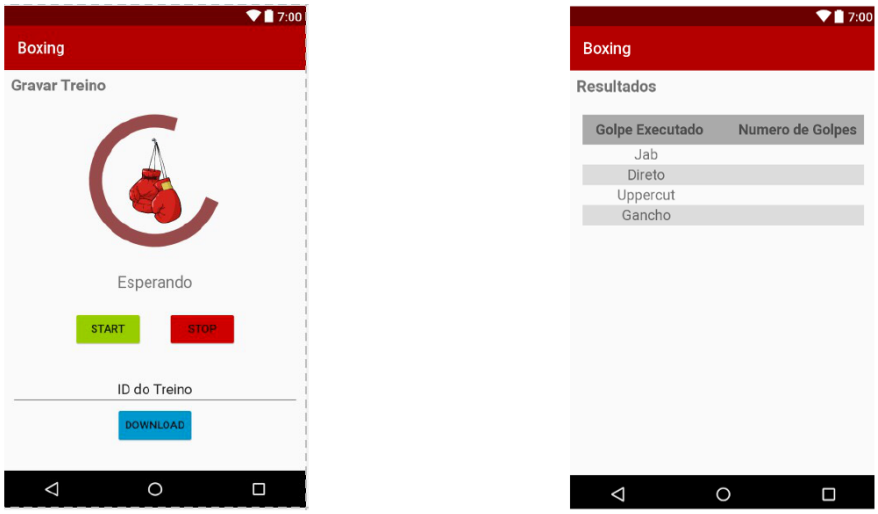
Escolhida a plataforma de desenvolvimento, foi definido o escopo do aplicativo especificando as funcionalidades desejadas e fluxos de operação.

De forma simplificada, o aplicativo deve se conectar com os dispositivos *MetaMotion R* presentes no interior das luvas. Uma vez que as conexões estejam estabelecidas a gravação dos dados é liberada e pode ser iniciada através do botão "*Start*", como mostra a Figura 19.

Ao fim da sessão de treinamento, o usuário deve pressionar o botão "*Stop*" para que a coleta termine e os dados sejam transferidos para o telefone celular.

Uma vez que todos os dados estejam armazenados no aplicativo ocorre o seu processamento e reconhecimento dos golpes, para que então ocorra a apresentação dos resultados, como pode ser observado na Figura 19.

Figura 19: Coleta de dados (Esquerda) e Resultados (Direita).



Fonte: Autoria Própria.

6.2 A PLATAFORMA *ANDROID*

A plataforma *Android* trata-se de um sistema operacional voltado para dispositivos móveis com seu núcleo baseado em *Linux*, e seu código atual é disponibilizado pela empresa Google (DEVELOPERS, 2017).

O ambiente de desenvolvimento de aplicativos para o sistema operacional *Android* é baseado na linguagem *Java* que é uma linguagem orientada a objeto (SUN, 2017). Quando uma linguagem é dita orientada a objeto significa que existem diferentes propriedades definidas previamente que são capazes de alterar as características de um elemento que possui uma estrutura de dados, comportamento e operações definidas (SANTOS, 2013). O elemento que sofre estas alterações de propriedades é chamado de objeto (RICARTE, 2001).

As funções capazes de alterar as propriedades de um objeto são chamadas de métodos e estão localizados no mesmo agrupamento de código dos objetos. Estes agrupamentos de instruções e parâmetros para criar objetos e definir métodos são chamados de classe (SANTOS, 2013).

> As classes de programação são receitas de um objeto, onde características e comportamentos são definidos, permitindo assim armazenar propriedades e métodos dentro dela. Para construir uma classe é preciso utilizar o pilar da abstração (GARCIA, 2015)

Em resumo, a linguagem *Java* possui classes onde existem instruções para criar diferentes tipos de objetos, com atributos de dados diferentes, esses elementos podem ter suas propriedades alteradas através de um método. A compreensão destes conceitos é importante para o desenvolvimento de um aplicativo *Android* e entendimento das próximas etapas deste projeto (DEVELOPERS, 2017).

6.3 DESENVOLVENDO EM *ANDROID*

Um aplicativo *Android* possui diversos blocos de construção e cada um desses componentes representa uma forma diferente do sistema interagir com a aplicação. Estes blocos individuais ajudam a definir o comportamento geral do aplicativo (DEVELOPER, 2017).

Um dos principais blocos de um aplicativo são as atividades. Elas representam a tela de *interface* com o usuário, e apesar de serem independentes umas das outras, oferecem uma experiência coesa no uso do aplicativo (DEVELOPER, 2017).

Esta independência entre as atividades faz com que cada uma possua seu próprio ciclo de vida. Este pode depender da associação com outras atividades e seu gerenciamento ocorre através do uso de métodos de retorno de chamada utilizados pelo sistema quando ocorre alguma transição no estado da aplicação. Na Tabela 1 tem-se um resumo dos métodos de retorno utilizados durante o ciclo de vida de uma aplicação.

Tabela 1: Tabela de métodos de retorno.

Método	Descrição
onCreate	Chamado quando a atividade é criada pela primeira vez. É onde se deve fazer toda a configuração estática normal - criar exibições, vincular dados e listas.
onRestart	Chamado depois que a atividade tiver sido interrompida, logo antes de ser reiniciada.
onStart	Chamado logo antes da atividade se tornar visível ao usuário.
onResume	Chamado logo antes da atividade iniciar o contato com o usuário.
onDestroy	Chamado quando a atividade não está mais visível ao usuário. Isso pode acontecer porque ela está sendo substituída por outra atividade.

Fonte: Adaptado de <http://https://developer.android.com/guide/components/activities.html>

Além das atividades, também existem os blocos de serviços responsáveis por operações em segundo plano, provedores de conteúdo para compartilhar os dados dos aplicativos e receptores de transmissão que são responsáveis por entregar mensagens vindas do sistema.

Ao abrir um aplicativo, antes que ocorra a execução destes blocos, o sistema efetua a leitura de um arquivo chamado *AndroidManifest.xml*. Dentro deste arquivo estão declarados todos os componentes, permissões, requisitos e recursos de hardware e software a serem utilizados pela aplicação.

> Todo aplicativo tem que ter um arquivo *AndroidManifest.xml* (precisamente com esse nome) no diretório raiz. O arquivo de manifesto apresenta informações essenciais sobre o aplicativo ao sistema *Android*, necessárias para o sistema antes que ele possa executar o código do aplicativo (DEVELOPERS, 2017).

Além da linguagem *java* existem outros recursos separados do código-fonte, em sua grande maioria, relacionados com os aspectos visuais do aplicativo (DEVELOPERS, 2017).

Um exemplo de recurso separado são os arquivos *.xml* onde são definidos os menus, estilo, animações e cores de acordo com a necessidade do desenvolvedor. Para acessar estes recursos, a ferramenta de desenvolvimento *Android* atribui uma identificação específica para cada um dos itens a serem utilizados no arquivo *.xml*, desde que estejam inseridos nos diretórios corretos. Estes recursos foram utilizados para criar a *interface* com o usuário.

6.4 RECURSOS PARA O DESENVOLVIMENTO

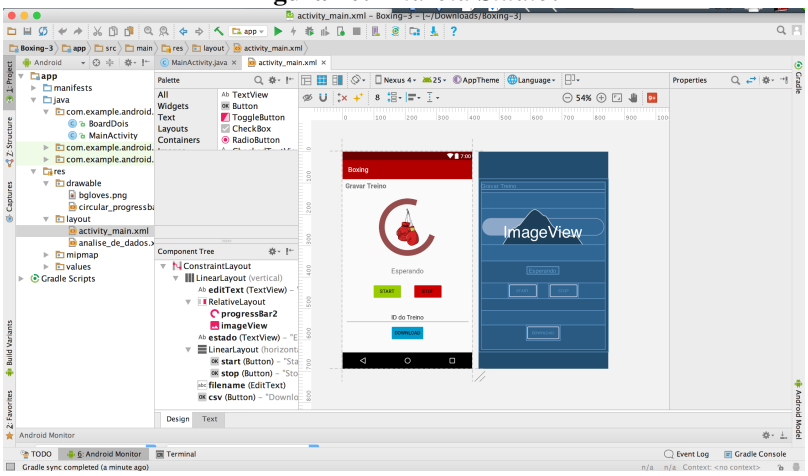
Como visto anteriormente, aplicações *Android* são constituídas de diversos blocos e elementos de código-fonte que operam em conjunto. Para que todos estes elementos possam estar integrados, é necessário gerenciar o projeto através de uma plataforma de desenvolvimento (*Software Development Kit - SDK*). Para o aplicativo proposto, foi utilizado o *SDK Android Studio 2.3* como mostra a Figura 20.

Este *SDK* foi escolhido por se tratar do ambiente oficial para o desenvolvimento de aplicativos disponibilizado pela empresa Google, o *software* possibilita agrupar todos os recursos da aplicação, emular o aplicativo e também efetuar o processo de testes e *debug* diretamente no aparelho celular do desenvolvedor.

O aparelho celular utilizado durante o processo foi o modelo A0001 da marca OnePlus One rodando a versão 6.0.1 do sistema *Android*, mesmo aparelho utilizado para o teste de prova de conceito.

Para que seja possível comunicar o aplicativo com dispositivos *MetaMotion R* é necessário implementar a *interface* de programação do aplicativo (*Application Programming Interface - API*) disponibilizada pelo fabricante dos dispositivos.

Figura 20: ***Android Studio*****.**



Fonte: Autoria Própria.

O *API* facilita o processo de programação pois define como a comunicação entre um dispositivo ou serviço e o aplicativo vai ocorrer, estabelece as regras de acesso dos recursos e fornece blocos de função com métodos bem definidos (BURTON, 2015).

6.5 CONFIGURAÇÕES DO AMBIENTE DE DESENVOLVIMENTO

Antes de iniciar a programação do aplicativo, é necessário configurar o *Android Studio* para se comunicar com os dispositivos *MetaMotion R*. Para que isso seja possível, é necessário compilar as dependências *MetaWear Android API 3.0*.

O processo de compilação de dependências ocorre através do sistema de automação de compilação *open source Gradle*. Este sistema é responsável pelo processo de construção do aplicativo juntando todos os recursos, dependências e bibliotecas em um único pacote que será executado pelo aparelho (BURTON, 2015).

As configurações do sistema *Gradle* estão localizadas no arquivo *build.gradle* e são aplicadas ao módulo do projeto, permitindo a inclusão de dependências no processo de compilação do aplicativo (ANDROIDPRO, 2017).

Com o arquivo *build.gradle* sincronizado, deve-se declarar os serviços a serem utilizados no módulo *AndroidManifest.xml*. Para comunicar com o dispositivo, é necessário declarar o serviço *BtleService* dentro do manifesto, então instanciar a classe dentro da atividade que precisa de acesso ao objeto referente aos dispositivos *MetaMotion R*.

Instanciar uma classe significa criar um objeto que represente a classe dentro do fragmento da atividade em que seus recursos serão utilizados (SANTOS, 2013). No caso do serviço *BtleService*, foi declarado dentro do método *OnCreate()* responsável pelas primeiras operações do aplicativo quando é iniciado.

6.6 CONFIGURANDO A CONEXÃO COM OS DISPOSITIVOS

Com o ambiente de desenvolvimento configurado, inicia-se a programação do aplicativo. O primeiro passo é estabelecer a conexão com os dispositivos através da classe *BluetoothDevice*, assim que a conexão é estabelecida. Para que seja possível enviar instruções de configuração, utiliza-se o método *getMetaWearBoard para* criar um objeto. Este objeto representa o dispositivo dentro do aplicativo. No caso deste projeto foram escolhidos os nomes *board* e *board2* para cada um dos dispositivos *MetaMotion R*.

Para configurar os sensores, é necessário estabelecer uma conexão assíncrona através dos objetos que representam os dispositivos. A grande vantagem do uso de conexões assíncronas é que elas permitem o processamento paralelo de diversas tarefas (DEVELOPERS, 2017). Com a conexão estabelecida, é possível executar outras operações para a captura dos dados.

No caso deste projeto, todas as conexões com os sensores também ocorrem de forma assíncrona, já que os giroscópios, acelerômetros e magnetômetros, utilizados para a fusão sensorial, estão constantemente obtendo informações em segundo plano (MBIENTLAB, 2017).

Definidos os tipos de conexão do dispositivo, e seus sensores, é possível prosseguir para a captura dos dados.

6.7 CONFIGURAÇÕES DA COLETA DE DADOS

Os ângulos de Euler (*pitch e roll*) e as componentes da aceleração linear são as entradas desejadas para a rede neural do projeto. Para obter tais medidas, é necessário utilizar o módulo de fusão sensorial, *SensorFusion*, da plataforma.

Consultando as especificações técnicas do produto, verificou-se que a taxa de transferência no modo de *streaming* é insuficiente para transmitir os ângulos de Euler e aceleração em tempo real. Além disso, o método *streaming* pode apresentar perda de pacotes quando a comunicação *Bluetooth* está fraca. Para contornar este problema, foi decidido utilizar o módulo de *Logging*, presente no *API*. Este módulo é responsável por armazenar os dados

coletados na memória interna do *MetaMotion R*; dessa forma, os dados só são transferidos para o aplicativo ao final da coleta.

A coleta de dados dos sensores é feita através do estabelecimento de uma rota. Esta tem uma origem e um destino bem definidos, no caso do dispositivo, os sensores são a origem, e o destino é o módulo de *Logging*. Dentro desta rota são especificados os parametros de aquisição de cada um dos sensores.

Para que as os valores medidos pelo sensor sejam baixados de forma correta, é necessário configurar alguns parâmetros como pode ser observado na Tabela 2.

Tabela 2: Configurações do módulo *Sensor Fusion*.

Parâmetro	Configuração	Descrição
Modo de Operação	NDof (*Number of Degrees of Freedom*)	Cálculo da orientação absoluta através do giroscópio, acelerômetro e magnetômetro.
Frequência do Acelerômetro	100Hz	Frequências necessárias para o funcionamento do NDof.
Frequência do Giroscópio	100Hz	
Frequência do Magnetômetro	25Hz	
Unidade da Aceleração	g	Aceleração corrigida.
Unidade dos Ângulos de Euler	graus	Ângulos de posição.

Fonte: Autoria Própria.

Uma vez configurados estes parâmetros no código do aplicativo, sempre que um dispositivo *MetaMotion R* for conectado ao aplicativo, ele receberá estas configurações.

Ao final da coleta, quando o usuário pressionar o botão "*Stop*", a função de *download* é iniciada, em um dispositivo por vez, tratando cada uma das informações baixadas de acordo com seu tipo de grandeza. Todos estes dados recebidos são armazenados em um vetor que deve passar por um processo de tratamento antes de ser enviado para rede neural.

6.8 TRATAMENTO DOS DADOS

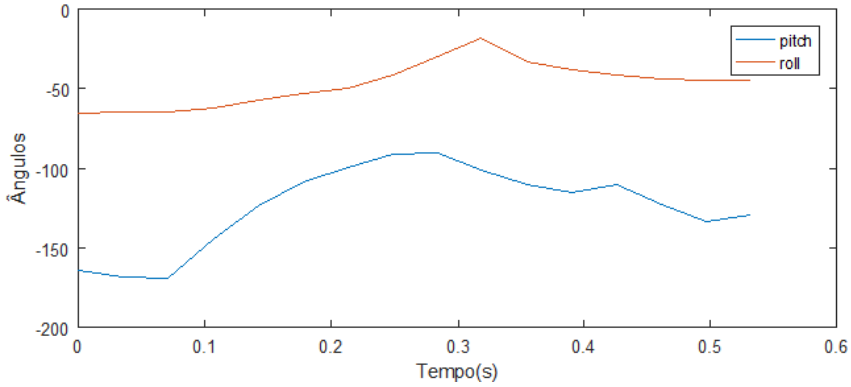
Os dados dos sensores de cada uma das luvas baixados pelo aplicativo devem ser adequados pela base *FIR* antes que possam constituir as matrizes de entradas de simulação das redes neurais. A função responsável por processar os dados de entrada para a base *FIR* foi denominada "baseFIR" e está disponível no Apêndice C.

Em seguida, as matrizes geradas pela função "baseFIR" serão o elemento de entrada da função de simulação da rede neural,"simulaRede", treinada para identificar os golpes.

7 TREINAMENTO DA REDE NEURAL

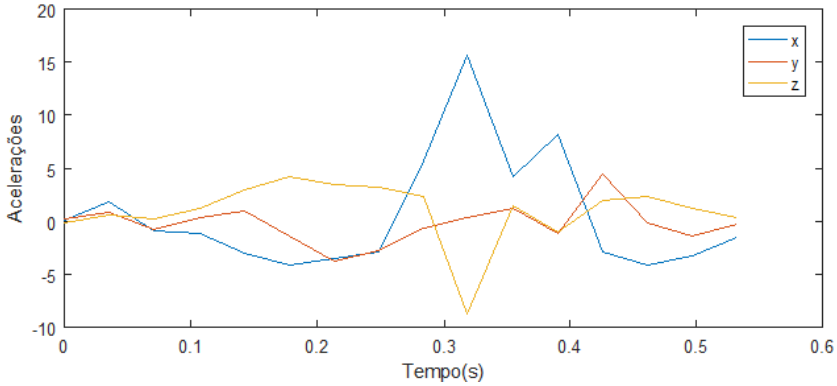
Para que o aplicativo possa identificar os golpes desferidos pelo lutador, é necessário treinar a sua rede neural. Este processo deve ser feito através da coleta de dados referentes aos golpes que se deseja identificar como pode ser visto nas Figuras 21 e 22.

Figura 21: Variação dos ângulos de *pitch* e *roll* na execução do *jab*.



Fonte: Autoria Própria.

Figura 22: Variação da aceleração nos três eixos durante a execução do *jab*.



Fonte: Autoria Própria.

Dentro do código do aplicativo foi inserida a biblioteca *.OpenCsv* para permitir a exportação dos dados coletados. Estes arquivos ficam dentro da memória do celular e podem ser transferidos para o computador.

A obtenção destes dados ocorreu através de sessões de treinamento controladas com dois lutadores profissionais do boxe. O primeiro, Hemerson Neris De Souza, 27 anos, 1,8m de altura, 90kg, com 14 anos de experiência no boxe, e o segundo, Claudiomar Pedra dos Santos, 36 anos, 1,72m de altura, 70kg, com 14 anos de experiência no esporte. Em cada um dos treinos, foram definidos os golpes a serem executados e o intervalo de tempo entre cada um deles. Este processo auxilia na separação dos dados para o treinamento da rede, pois os instantes em que ocorrem os movimentos já são conhecidos. Foram utilizados também recursos de vídeo para gravar as sessões de treino e auxiliar nas análises.

Durante os treinos, foram definidas diferentes sequências de golpes para coletar as variações dos socos devido a execução de um movimento anterior, como pode ser visto na Tabela 3. Além de movimentos referentes a golpes conhecidos, foram feitos movimentos que não representavam golpes para auxiliar no treinamento da rede.

Após a transferência dos dados para o computador, verificou-se que a quantidade de amostras de um sensor em relação ao outro variava, mesmo com uma mesma taxa de amostragem definida, impossibilitando assim a criação de uma única rede neural. Por essa razão, foram treinadas duas redes, uma para cada luva. Para os treinos, adotou-se o ponto de vista de uma pessoa destra. Isto significa que, para a mão direita, os golpes de maior ocorrência são: direto, cruzado e *uppercut* e, para a mão esquerda, os golpes: *jab*, cruzado e *uppercut*.

Assim, para cada um dos golpes foi atribuído um *target*. Para mão direita foram definidos os valores $[100]$, $[010]$ e $[001]$ para os golpes direto, cruzado e *uppercut* respectivamente. O mesmo processo foi adotado para os golpes executados pela mão esquerda. Os *targets* definidos foram $[100]$, $[010]$ e $[001]$ para os golpes *jab*, cruzado e *uppercut* respectivamente, como na Tabela 4.

Antes de utilizar o algoritmo de treinamento, deve-se determinar o número de entradas das redes neurais, para isto, é preciso primeiramente determinar qual a ordem do modelo *FIR* a ser utilizado. Para a frequência de $30Hz$ optou-se por um *FIR* de ordem 9 para satisfazer a condição de um período de $0,3s$ $(9.1/30)$. A adequação dos dados para o modelo *FIR* é feita utilizando o código do MATLAB presente no Apêndice C. Definida a ordem da base *FIR*, cada componente de aceleração e orientação terá 10 elementos de entrada. Somando os elementos dos eixos x, y e z da aceleração e os elementos de $pitch$ e $roll$ da orientação, obtém-se 50 elementos de entrada para a rede neural.

Tabela 3: Sequência de golpes coletados.

Sequência dos Golpes				Número de Coletas da Sequência	Número de Lutadores
M. Esquerda	M. Direita			20	2
JAB	DIRETO				
M. Esquerda	M. Direita			20	2
JAB	UPPERCUT				
M. Esquerda	M. Direita	M. Esquerda		20	2
JAB	UPPERCUT	CRUZADO			
M. Esquerda	M. Direita	M. Esquerda	M. Direita	20	2
JAB	DIRETO	UPPERCUT	CRUZADO		

Fonte: Autoria Própria.

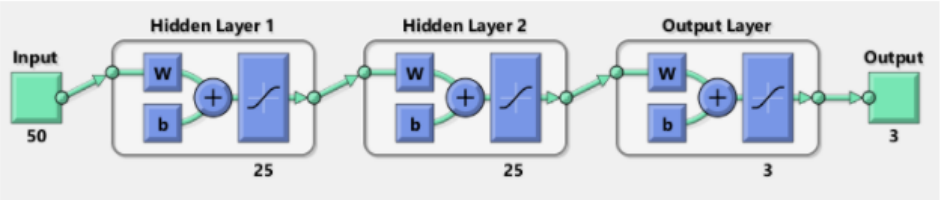
Tabela 4: Divisão dos golpes por mão.

M. Direita	Número de vezes	Valor do *Target*
DIRETO	60	1 0 0
UPPERCUT	20	0 1 0
CRUZADO	20	0 0 1
M. Esquerda	Número de vezes	Valor do *Target*
JAB	80	1 0 0
UPPERCUT	20	0 1 0
CRUZADO	20	0 0 1

Fonte: Autoria Própria.

Uma vez que cada um dos golpes esteja separado, adequado ao modelo *FIR* e associado a um *target*, este será utilizado como uma entrada para a matriz de treinamento da rede neural através do método de Levenberg Marquardt, com a mesma estratégia utilizada no teste de conceito.

Figura 23: Rede neural treinada.

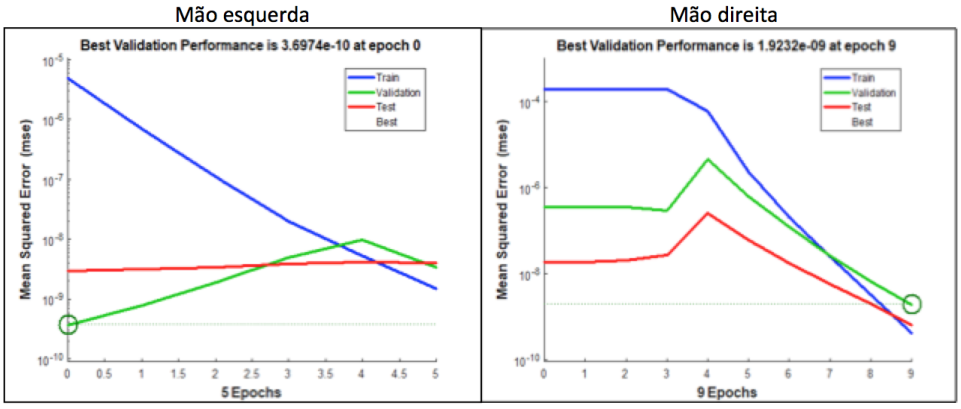


Fonte: Autoria Própria.

As redes neurais serão do tipo *MIMO* (*multiple inputs multiple outputs*) e foram estruturadas com 50 elementos de entrada, duas camadas escondidas com 25 neurônios cada e uma camada de saída com 3 neurônios, que corresponderão aos 3 golpes mencionados, na mesma ordem como na Figura 23.

Com os treinamentos efetuados, obtiveram-se os valores de época de 5 e 9 no treinamento da rede neural da mão esquerda e direita, respectivamente. A época representa o número de iterações necessárias para alcançar o *MSE* desejado, como na Figura 24.

Figura 24: Dados de treinamento da rede neural.



Fonte: Autoria Própria.

Finalizando o treinamento das redes neurais, é possível gerar uma função no MATLAB que simula cada rede, contendo as constantes dos pesos de todos os seus neurônios. Com a função gerada e os dados de sessões de treinamento livre, onde o lutador poderia executar

qualquer golpe sem qualquer intervalo de tempo entre estes, foi possível fazer uma pré avaliação da rede treinada. Utilizando estes dados, simulava-se a rede no MATLAB e verificava-se o índice de acertos na identificação dos golpes.

Com os resultados obtidos, foi possível verificar que a separação de golpes considerados adequados para o treinamento da rede era de extrema importância. A seleção dos golpes deve ocorrer de forma precisa, definindo o ponto de inicio do golpe e de final, selecionando somente golpes com a execução correta da técnica e suas variações. Todos estes fatores contribuem para a boa *performance* da rede neural.

Tendo como base estes parâmetros, para selecionar os golpes que seriam parte da rede neural do aplicativo, com auxilio do programa Excel e dos vídeos gravados, foram localizados picos de aceleração nos dados, para definir a localização dos golpes, e estes eram separados de acordo com o seu tipo para o novo treinamento da rede.

A partir disto é possível realizar a tradução desta função de MATLAB para uma função em *Java*, com o objetivo utilizar o próprio aplicativo para também simular as redes neurais para dados desconhecidos e contabilizar o número de golpes.

8 TESTES E RESULTADOS

Após a inclusão da rede neural treinada dentro do aplicativo, iniciou-se o processo de testes com a participação do boxeador Claudiomar, descrito anteriormente. O objetivo foi verificar a precisão na identificação dos golpes, os limites na identificação dos movimentos e a velocidade de processamento dos resultados.

Os testes foram divididos em três etapas, a primeira etapa contava apenas com movimentos simples, com intervalos de tempo bem definidos para a execução de cada um dos testes. A segunda foi feita com combinações de pelo menos 3 golpes com um intervalo de tempo também pré definido. A último etapa foi efetuada de forma livre, onde o lutador podia executar qualquer golpe, sem um intervalo de tempo entre estes. Com o auxílio do lutador, foram identificados os golpes no vídeo da sessão e então comparados com os resultados obtidos pelo aplicativo.

8.1 PRIMEIRA ETAPA DE TESTES

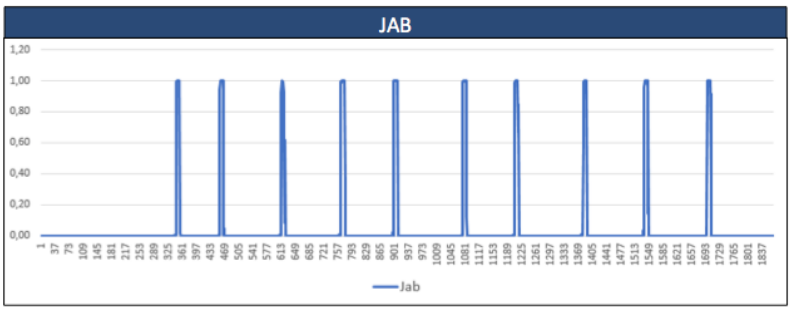
Para esta primeira etapa foram executadas sessões de treino de boxe contendo 10 execuções de cada um dos socos. Em cada uma delas, foi solicitado para que o lutador executasse um dos tipos de golpe, com um intervalo pré-definido entre eles, o tempo entre os golpes foi de 5 segundos.

Os resultados fornecidos pela rede neural estão disponíveis nas Figuras de número 25 até 28, onde a identificação do golpe corresponde a um pico no gráfico com uma determinada duração. Para que um golpe seja considerado como identificado, a função “ContaPicos” verifica se os valores de quatro amostras consecutivas da saída da rede neural possui uma média de 0.8, e para impedir que seja identificado o mesmo golpe mais de uma vez pela função, são ignoradas as 10 próximas amostras.

A espessura dos picos se refere ao número de amostras que condizem ao golpe. Quando os picos são muito curtos, eles se referem apenas a um instante que se pareceu com

o golpe, não sendo identificado pela função “ContaPicos” como um movimento válido.

Figura 25: Identificação do *Jab*.



Fonte: Autoria Própria.

Figura 26: Identificação do Direto.

Fonte: Autoria Própria.

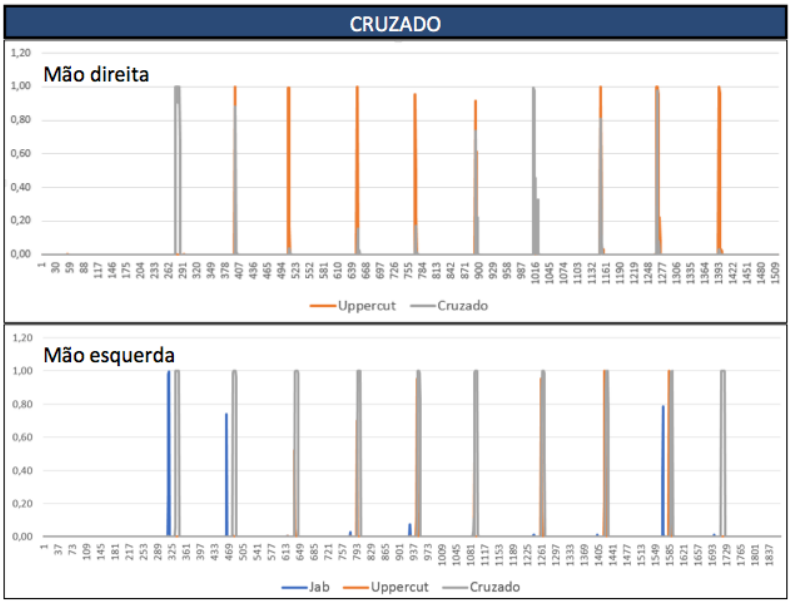
A identificação do cruzado presente na Figura 27 obteve o pior desempenho. A rede neural da mão direita confundiu o golpe com o *uppercut*, como pode ser observado pela linha laranja presente no gráfico.

Como pode ser observado, nesta primeira etapa foram identificados $83,3\%$ dos golpes, sendo que o *jab*, direto e *uppercut* foram os golpes de melhor identificação e o cruzado obteve o pior reconhecimento. Os dados obtidos nesta etapa estão consolidados na Tabela 5.

8.2 SEGUNDA ETAPA DE TESTES

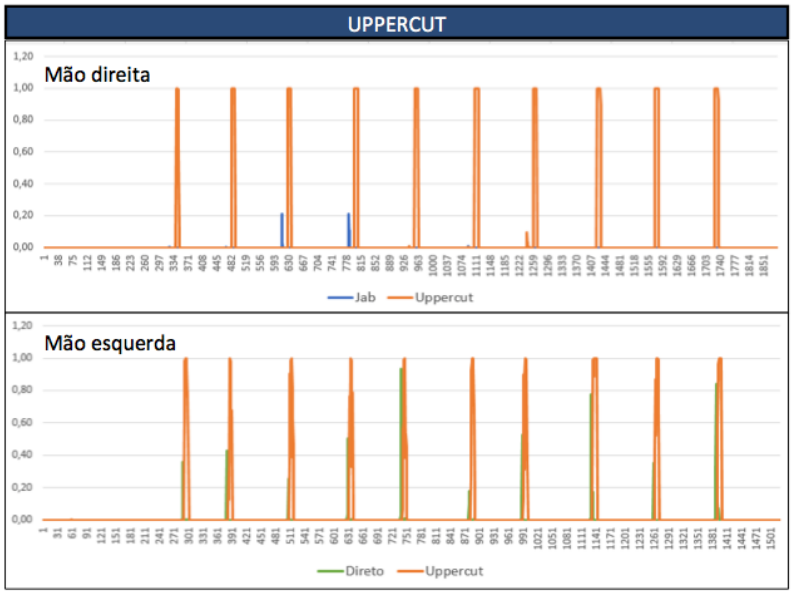
Para avaliar mais a fundo a *performance* do aplicativo, o segundo teste foi efetuado de forma mais complexa. Foi solicitado para o lutador que executasse diferentes sequências de golpes contendo pelo menos três socos em um intervalo de tempo pré estabelecido de 5 segundos, com 5 repetições.

Figura 27: Identificação do Cruzado.



Fonte: Autoria Própria.

Figura 28: Identificação do *Uppercut*.



Fonte: Autoria Própria.

Tabela 5: Resultados da primeira etapa de testes.

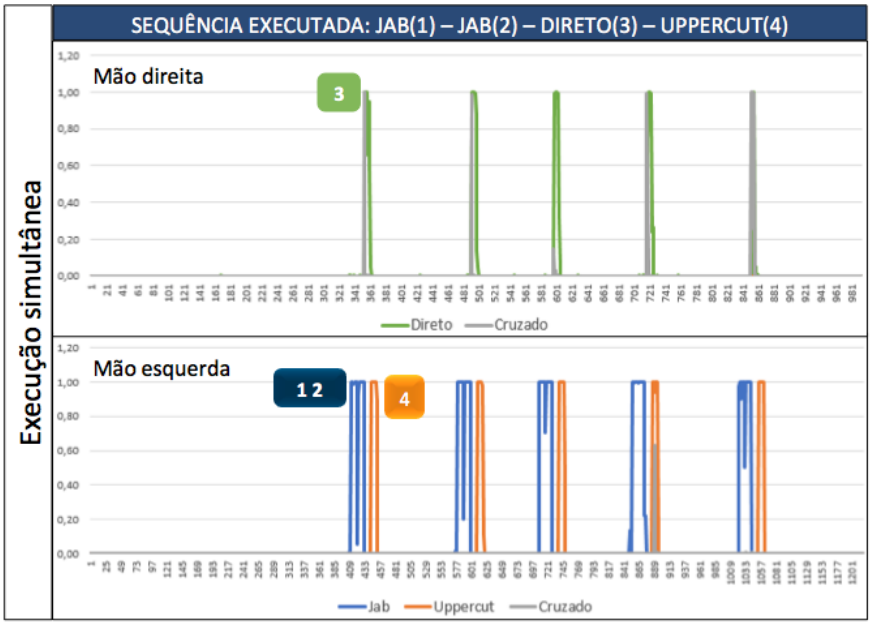
M. Direita	Número de Golpes	Golpes Identificados	% de Acerto
DIRETO	10	10	100%
UPPERCUT	10	10	100%
CRUZADO	10	2	20%
M. Esquerda	Número de vezes	Golpes Identificados	% de Acerto
JAB	10	10	100%
UPPERCUT	10	10	100%
CRUZADO	10	8	80%

Fonte: Autoria Própria.

Os gráficos presentes nas Figuras de número 29 até 32 se referem a identificação das sequências de golpes pela rede neural. Os gráficos estão divididos de acordo com a mão que executou o soco, e sua sequência é representada pelos números indicados próximos aos picos. Por exemplo, no gráfico da Figura 29 de sequência *jab*, *jab*, direto e *uppercut*, espera-se que na saída da rede neural da mão direita se tenha um pico que representa o direto, e a da mão esquerda, dois picos de *jabs* e um de *uppercut*. Como foram feitas 5 repetições, espera-se que este ciclo se repita 5 vezes.

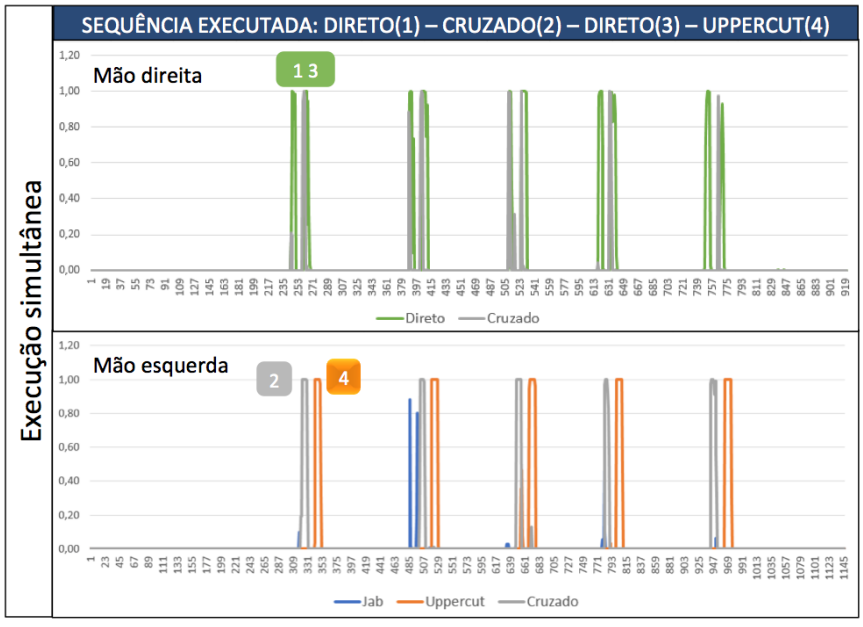
Na execução dos combos é possível observar que a rede encontrava padrões nos resultados que representavam outros golpes. No entanto quando isso ocorre em um instante curto, esta identificação incorreta não é contabilizada pela função “ContaPicos”.

Figura 29: Identificação do *Jab*-*Jab*-Direto-*Uppercut*.



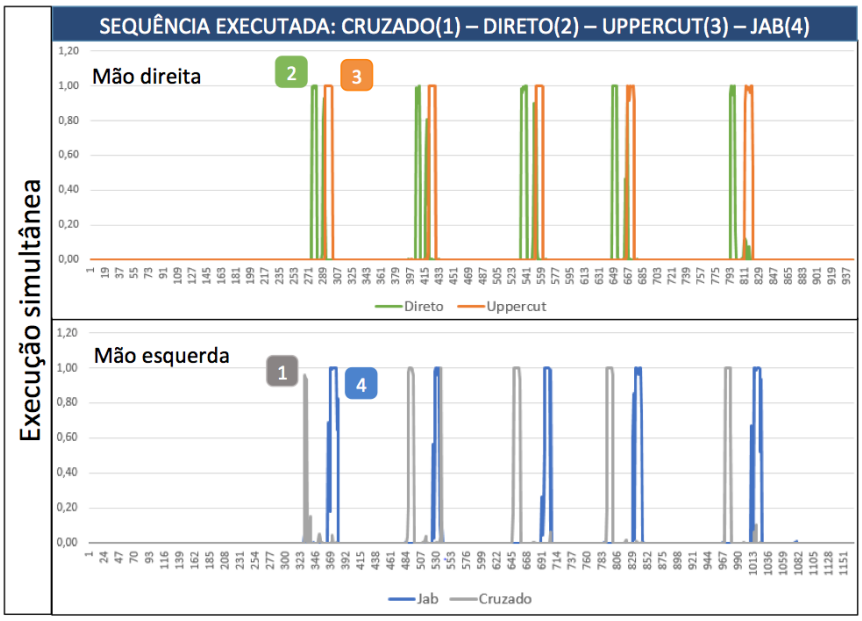
Fonte: Autoria Própria.

Figura 30: Identificação Direto-Cruzado-Direto-*Uppercut*.



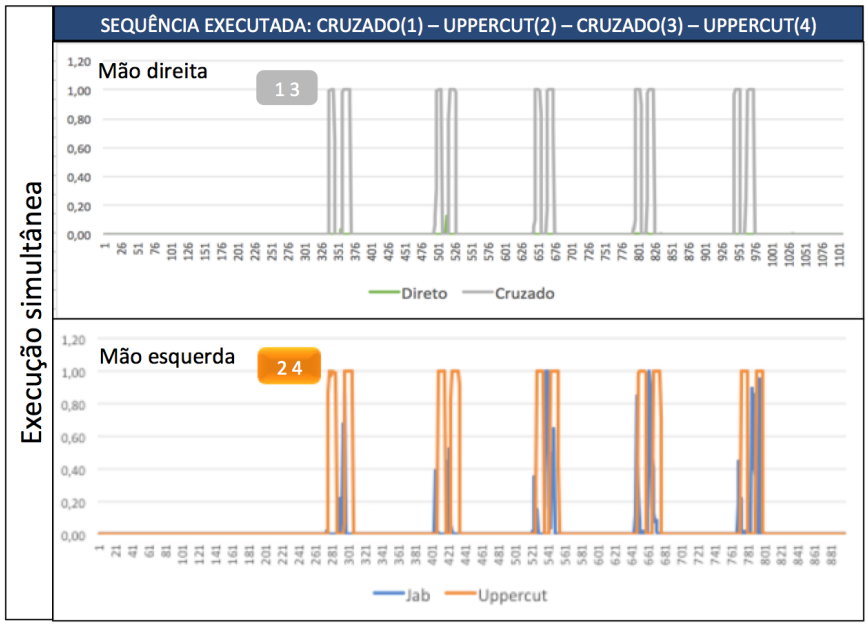
Fonte: Autoria Própria.

Figura 31: Identificação Cruzado-Direto-*Uppercut*-*Jab*.



Fonte: Autoria Própria.

Figura 32: Identificação Cruzado-*Uppercut*-Cruzado-*Uppercut*.



Fonte: Autoria Própria.

Neste treino, a identificação dos golpes foi ligeiramente superior. Acredita-se que

isso ocorreu devido às mudanças na execução dos golpes quando comparados com os golpes isolados. Os resultados estão presentes na Tabela 6.

Tabela 6: Resultados da segunda etapa de testes.

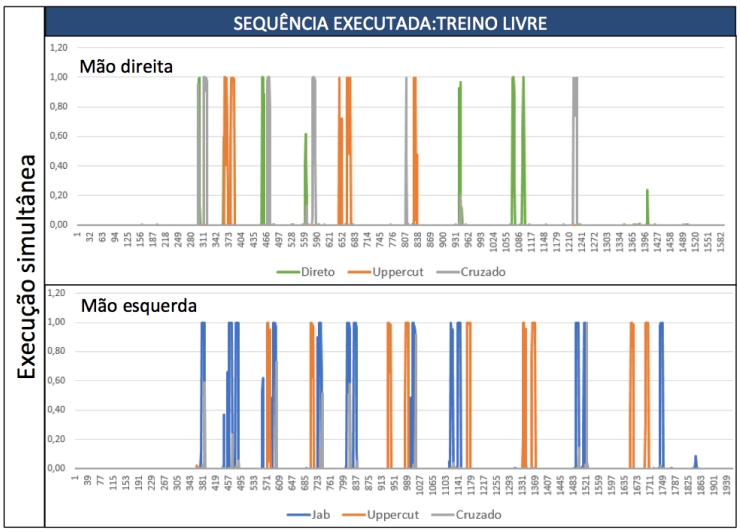
M. Direita	Número de Golpes	Golpes Identificados	% de Acerto
DIRETO	20	12	60%
UPPERCUT	15	15	100%
CRUZADO	10	10	100%
M. Esquerda	Número de vezes	Golpes Identificados	% de Acerto
JAB	15	15	100%
UPPERCUT	10	10	100%
CRUZADO	20	19	95%

Fonte: Autoria Própria.

8.3 ÚLTIMA ETAPA DE TESTES

O terceiro e último teste executado buscou simular o ambiente de treino, onde não existem restrições dos golpes aplicados nem um tempo definido entre eles. O lutador executou sequências de golpes comuns nas lutas, durante um período de 1 minuto por sessão, como apresentado na Figura 33.

Figura 33: Identificação Treinamento Livre.



Fonte: Autoria Própria.

As sequências executadas estão na Tabela 7, onde os números nas células representam a ordem em que os golpes foram executados e as cores indicam se os golpes foram identificados adequadamente. Os dados obtidos nesta etapa estão presentes na Tabela 8, Tabela 9.

Tabela 7: Golpes executados na última etapa de testes.

Treino Livre												
Sequência Executada	1ª	2ª	3ª	4ª	5ª	6ª	7ª	8ª	9ª	10ª	11ª	12ª
Jab	3	2 4	4	4	2 4		3	1 2		1 3		2
Uppercut Esquerdo	1		2	2		1	1	4	2 4		1 3	
Cruzado Esquerdo	4											
Direto	2		1 3	1				3	1 3			1
Uppercut Direito		1 3			1 3		4					
Cruzado Direito				3			2			2	2	

Golpes identificados | Um dos golpes identificado | Golpe não Identificado

Fonte: Autoria Própria.

Tabela 8: Resultados da última etapa de testes.

Treino Livre													
Sequência Executada	1ª	2ª	3ª	4ª	5ª	6ª	7ª	8ª	9ª	10ª	11ª	12ª	**Total**
Nº de golpes	4	4	4	4	4	1	4	4	4	3	3	2	41
Nº identificados	2	4	3	3	3	1	3	3	4	2	2	1	31
Taxa acerto	50,00%	100,00%	75,00%	75,00%	75,00%	100,00%	75,00%	75,00%	100,00%	66,67%	66,67%	50,00%	**75,61%**

Fonte: Autoria Própria.

8.4 ANÁLISE DOS RESULTADOS E DIFICULDADES

Após os testes, foi possível observar que a execução dos golpes varia bastante de acordo com os movimentos que os antecedem. Uma vez que a luta esteja ocorrendo, o golpe nem sempre é feito com a melhor técnica possível, podendo interferir em sua identificação. No entanto, mesmo com variações nos golpes, as taxas de acerto médio foram de $83,3\%$, $92,5\%$ e $75,61\%$ para cada uma das etapas de teste, na mesma ordem em que foram apresentados, o que pode ser considerado como promissor, dado o pequeno número de golpes coletados para o treinamento da rede.

Para a primeira etapa de testes, que buscava identificar golpes isolados, a rede neural conseguiu identificar quase todos os golpes desferidos, exceto pelo cruzado da mão direita, que teve uma taxa de 20% de acerto. Em contrapartida, os jabs, diretos e *uppercut* conseguiram ser

Tabela 9: Resultados da última etapa de testes.

M. Direita	Número de Golpes	Golpes Identificados	% de Acerto
DIRETO	8	3	37,50%
UPPERCUT	5	4	80,00%
CRUZADO	4	2	50,00%
M. Esquerda	Número de vezes	Golpes Identificados	% de Acerto
JAB	13	12	92,31%
UPPERCUT	10	9	90,00%
CRUZADO	1	1	100%

Fonte: Autoria Própria.

todos identificados. Para melhorar a identificação do cruzado direito, seria necessário coletar mais amostras do golpe para adequar o treinamento da rede neural.

Na segunda etapa, que buscava identificar 3 ou mais golpes, houve um índice de acerto de quase 100% para todos os golpes, exceto para o direto. Este, com taxa de identificação de 60%, necessita de mais amostras para que sua identificação não seja confundida com o golpe cruzado. As melhores identificações foram o jab, *uppercut*, e cruzado esquerdo, com 100% acerto. Acredita-se que isso ocorreu devido a falta de discernimento da rede em relação ao ângulo de rotação necessário para a execução do direto e do cruzado.

A terceira etapa, onde o lutador ficava livre para determinar os golpes e sequências a serem desferidos, também apresentou problemas de identificação do direto e cruzado direito, com $37,5\%$ e $50,0\%$ respectivamente, confirmando a necessidade da coleta de mais amostras de direto e cruzado direito para aumentar o índice de acertos. Os golpes que foram mais reconhecidos foram jab, *uppercut* e cruzado esquerdo, entretanto, este último só ocorreu uma vez.

Então, de forma geral, acredita-se que com um maior número de dados durante a coleta, e sua separação de forma mais rígida, o desempenho do protótipo teria alcançado marcas ainda melhores. Como mencionado no capítulo anterior, o processo de treinamento da rede neural deve levar em conta diversos parâmetros na seleção dos golpes de treinamento, como a sua quantidade, variações na execução e emprego correto da técnica. Estes aspectos impactam diretamente no desempenho do aplicativo.

Durante o treinamento de boxe, para validar o protótipo, a falta de conhecimento técnico muitas vezes dificultou a identificação do movimento executado pelo lutador. Para verificar as sequências executadas foi necessário conferir a ordem dos golpes com o atleta.

Levando em conta os resultados obtidos e os parâmetros para a melhoria de desempenho, algumas dificuldades existentes durante o processo de criação do projeto ficaram em evidência.

As instabilidades da conexão *Bluetooth* do celular com o sensor faziam com que em algumas situações não fosse possível conectar as duas luvas ao aplicativo. Buscando entender o comportamento dos dispositivos, entrou-se em contato com o seu fabricante. Este informou que devido a execução assimétrica das operações dentro do *MetaMotion R*, quando a conexão *Bluetooth* está fraca e é enviado um conjunto de instruções de configurações extenso, como ocorre no projeto, o tempo limite de execução dos comandos é excedido resultando na falha da conexão. Com base nas informações recebidas, foi verificado que o sinal *Bluetooth* vindo dos dispositivos variava bastante devido às barreiras físicas existentes para sua acomodação nas luvas de boxe. Para evitar este comportamento é desejável que o aparelho celular esteja próximo dos dispositivos ao pressionar o botão *start* do aplicativo.

Outro problema encontrado com a conexão *Bluetooth* foi que em sua versão de baixo consumo de energia, a taxa de transferência com o sinal fraco fazia com que o *download* dos dados para o dispositivo fosse lento.

Limitando a frequência dos sensores à 30Hz, observou-se que o sensor da luva esquerda registrava amostras à uma taxa de 30Hz, enquanto o da mão direita, à 25Hz. Isso impossibilitou a criação de uma única rede neural para receber os dados dos dois sensores, que era o plano original. Para contornar este problema, foi decidido que seriam criadas duas redes neurais, uma para cada mão.

9 CONCLUSÕES E PROJETOS FUTUROS

Durante o desenvolvimento do projeto foi possível compreender os elementos fundamentais do boxe, quais são os golpes de maior relevância em sua prática e as grandezas associadas a sua execução. Validou-se o uso de sensores MEMs na mensuração destas grandezas e a viabilidade do uso de redes neurais para a identificação dos golpes.

Os testes executados demostram taxas de acerto de $83,3\%$ para a identificação de golpes executados individualmente, $92,5\%$ na identificação de golpes em sequências separadas e $75,61\%$ na condição mais crítica, que é a execução livre dos golpes. O golpe de pior identificação nos testes foi o cruzado com um índice de $56,67\%$ de acerto em média. Para melhorar o índice de identificação dos golpes seria necessário a coleta de mais amostras de golpes.

Estes resultados estão de acordo com os objetivos propostos, pois demonstram que a identificação através do uso de sensores MEMs combinados com redes neurais é possível.

De forma mais detalhada, com o estudo foi possível compreender os principais elementos e eventos do boxe, bem como a modificação de sua prática ao longo dos anos, partindo de uma prática bastante brutal até o esporte como é conhecido. A pesquisa de seus fundamentos auxiliou no detalhamento dos golpes que foram identificados, focando em padrões de aceleração angular, na movimentação do corpo e as diferenças que estes apresentam nos golpes *Jab*, Direto, Cruzado e *Uppercut*.

O uso de sensores MEMs foi uma boa alternativa devido ao seu baixo custo, tamanho reduzido e avanço tecnológico nos últimos anos. Foram selecionados o giroscópio, magnetômetro e acelerômetro como os sensores de interesse. O primeiro apresenta uma extensa aplicação na determinação da orientação, no entanto, sofre com efeitos de *drift*, o que tornou os dois sensores seguintes uma necessidade. Através do acelerômetro, giroscópio e magnetômetro, o dispositivo efetuou a fusão sensorial para obter os dados de aceleração e orientação.

A compreensão do funcionamento das redes neurais demonstrou seu potencial para o reconhecimento de padrões. Durante o levantamento bibliográfico foram estudados os

principais contribuintes do seu desenvolvimento ao longo da história, como também os tipos de treinamentos e algoritmos disponíveis. Através destes levantamentos, o algoritmo de Levenberg-Marquardt foi escolhido.

Acredita-se que os conceitos aplicados no desenvolvimento deste projeto, com foco na avaliação do treino do atleta no local da prática do esporte apresentam uma contribuição importante no desenvolvimento de ferramentas capazes de auxiliar lutadores, independentemente do nível de habilidade.

Utilizando-se da mesma estratégia deste trabalho, supõe-se que é possível também o reconhecimento de outros movimentos biomecânicos como a identificação de tipos de chutes, para esportes como muai thai e jiu jitsu, identificação de queda de pessoas, visando o público idoso que é mais susceptível à quedas, entre outras aplicações.

Com a validação do uso do dispositivo *MetaMotion R* e do uso das redes neurais para a identificação dos movimentos, a próxima fase deste trabalho será expandir a coleta de amostras de golpes possíveis e desenvolver mais funcionalidades do aplicativo, como o armazenamento das sessões de treinamento, funções de comparação entre as sessões de treino e opções para compartilhar os resultados

Estudos recentes indicam que atletas apresentam rendimento baixo quando treinados sob fadiga. Uma outra aplicação possível do protótipo seria identificar exatamente quando o lutador entra em fadiga, com o objetivo controlar mais eficientemente períodos de treino e descanso, melhorando o rendimento do treinamento do atleta.

Para melhorar o desempenho da rede neural, acredita-se que seja necessário expandir o número de coletas de exemplos dos golpes que se deseja identificar. No entanto, para que estas coletas melhorem o desempenho do projeto, estas devem ser coletadas levando em conta a qualidade, momento execução do golpe e técnica empregada. O auxílio de um profissional das artes marciais na separação dos golpes é bastante desejável.

No que se refere ao aplicativo, novas funções como armazenamento de treinos para consultas posteriores, ferramentas comparativas e opções de compartilhamento são desejáveis. Melhoras na interface do aplicativo também são necessárias, inserir um número maior de janelas de *feedback* para o usuário e alertas sonoros para facilitar o uso do aplicativo.

REFERÊNCIAS

AGUIRRE, L. A. **Introducão à identificação de sistemas- Técnicas lineares e não lineares aplicadas a sistemas reais.** Editora UFMG, 2004.

AHMADI, A.; Mitchell, E.; Destelle, F.; Gowing, M.; O'Connor, N.E.; Richter, C.; Moran, K. **"Automatic Activity Classification and Movement Assessment During a Sports Training Session Using Wearable Inertial Sensors"**, Wearable and Implantable Body Sensor Networks (BSN), 2014 11th International Conference on, On page(s): 98 ? 103

ARUS, Emeric. **Biomechanics of human motion: Applications in the martial arts.** CRC Press, 2012.

BAO L., S. S. Intille, **"Activity Recognition from User-Annotated Acceleration Data"**, in Proc. of PERVASIVE 2004, Linz, Vienna, Austria, Apr 2004.

BEMHARD E. Boser, **"Electronics for Micromachined Inertial Sensors"**, Trairsducers '97, 1997 Ittler7rotiud i?oifercncc on Solid-Sfate Sensors arid Acttiatom, 4B1.01, pp 1169- 1172.

BISHOP, Christopher M. **Neural networks for pattern recognition.** Oxford university, 1995.

BLUETOOTH, S. I. G. Bluetooth specification version 4.2. **Bluetooth SIG**, 2014.

BOUFFARD, Joshua Lee, **"An Alternative Sensor Fusion Method For Object Orientation Using Low-Cost Mems Inertial Sensors"** (2016). Graduate College Dissertations and Theses. Paper 537.

BUJARI, Armir, Bogdan Licar, and Claudio E. Palazzi. **"Movement pattern recognition through smartphone's accelerometer."** 2012 IEEE Consumer Communications and Networking Conference (CCNC). IEEE, 2012.

CARATTI, Jônatas Marques. **Calçando as luvas: primeiros comentários sobre a formação do boxe gaúcho** (Porto Alegre, 1920)."Revista Latino-Americana de História 1.3 (2012): 508-524.

CORBEN, Herbert Charles, and Philip Stehle. **Classical mechanics.** Courier Corporation, 1994.

DAVEY N., JAMES D. A. and RICE T., **"An accelerometer based sensor platform for insitu**

elite athlete performance analysis", Proc. IEEE Sensors, pp. 1373-1376, 2004

EDELSTEIN, Alan. Advances in magnetometry. **Journal of Physics: Condensed Matter**, v. 19, n. 16, p. 165217, 2007.

EL-DIASTY, M. **"An Accurate Heading Solution using MEMS-based Gyroscope and Magnetometer Integrated System (Preliminary Results)."** ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences 2.2 (2014): 75.

FEITOSA, Mario; LEITE, Nívea; LIMA, Amanda. Boxe. In: DACOSTA, Lamartine (ORG.). **Atlas do esporte no Brasil.** Rio de Janeiro; CONFEF, 2006.

GAFFNEY, **"An automated calibration tool for high performance wireless inertial measurement in professional sports"**.sensors, 2011 IEEE, pp. 262-265.

GARCIA, Vagner. **Classes e Objetos** Disponível em: http://fabrica.ms.senac.br/2015/03/classes-objetos-atributos-e-metodos-em-java/ Acesso em: 06 de mai. 2017.

GEEN, John A., et al. **"Single-chip surface micromachined integrated gyroscope with 50/h Allan deviation."** IEEE Journal of Solid-State Circuits 37.12 (2002): 1860-1866.

GRAHAM, Brian Barkley. **Using an Accelerometer Sensor to Measure Human Hand Motion.** 11 maio, 2000. Dissertação de Mestrado em Engenharia Elétrica e Ciência da Computação, Massachusetts Institute of Technology.

GURNEY, Kevin. **An introduction to neural networks.** CRC press, 1997.

HAYKIN, Simon; NETWORK, Neural. **A comprehensive foundation. Neural Networks**, v. 2, 2004.

ISPORTBOXING. **Basic Boxing Defense.** Disponível em: http://boxing.isport.com/boxing-guides/basic-boxing-defense. Acesso em: 15 de mai. 2016.

KAVANAGH, Justin J.; MENZ, Hylton B. Accelerometry: a technique for quantifying movement patterns during walking. Gait and posture, v. 28, n. 1, p. 1-15, 2008.

KRIESEL, David. **A brief Introduction on Neural Networks.** 2007.

KUIPERS, J. B. **Quaternions and rotation sequences: A primer with applications to orbits, aerospace and virtual reality.** September 1999.

LABAYRADE, Raphael; AUBERT, Didier. A single framework for vehicle roll, pitch, yaw estimation and obstacles detection by stereovision. In: Intelligent Vehicles Symposium, 2003.

Proceedings. IEEE. IEEE, 2003. p. 31-36.

LEMKIN, Mark A., et al. **"A 3-axis force balanced accelerometer using a single proof-mass."** Solid State Sensors and Actuators, 1997. TRANSDUCERS'97 Chicago., 1997 International Conference on. Vol. 2. IEEE, 1997.

LI, Wei; WANG, Jinling. Effective adaptive Kalman filter for MEMS-IMU/magnetometers integrated attitude and heading reference systems. **Journal of Navigation**, v. 66, n. 01, p. 99-113, 2013.

LORA, E. E. S.; NASCIMENTO, M. A. R. **Geração Termelétrica: planejamento, projeto e operação.** Rio de Janeiro. 1ª Edição. 2004. PG-155.

MACHADO, Jeremias Barbosa et al. **Modelagem de sistemas não-lineares por base de funções ortonormais generalizadas com funções internas.** 2011.

MENDOZA, Daniel. **The art of boxing.** Gale ECCO, 2010. Barbour, N.; Schmidt, G. Inertial sensor technology trends. IEEE Sens. J. 2001, 1, 332?339.

MESTER, Rudolf. Orientation estimation: Conventional techniques and a new non-differential approach. In: **Signal Processing Conference, 2000 10th European.** IEEE, 2000. p. 1-4.

MINSKY, Marvin; PAPERT, Seymour. Perceptrons. 1969.

NAVAS VX, Destefano J, Koo BJ, Doty E, Westerfeld D. **Smart Glove.** In: Systems, Applications and Technology Conference (LISAT), 2012, IEEE Long Island, Maio, 2012.

OKAZAKI, Victor Hugo Alves et al. **Ciência e tecnologia aplicada à melhoria do desempenho esportivo.** Revista Mackenzie de Educação Física e Esporte, v. 11, n. 1, 2012.

QIU, Jiaoming, and Yongyao Cai. **"Magnetometer with angled set/reset coil."** U.S. Patent Application No. 13/890,723.

RICARTE, Ivan Luiz Marques. **Programação Orientada a Objetos: uma abordagem com Java**. Universidade Estadual de Campinas, Campinas, SP, Brasil, 2001.

SILVA, Guilherme Barbosa da. **A prática do boxe como inclusão social de crianças e adolescentes: análise de um projeto social desenvolvido na cidade de Umbuzeiro-PB.** 2014.

SILVA NETO, Luiz Cordeiro da et al. **A prática esportiva como ferramenta de inclusão.** Artigo, 2013. Disponível em: http://creconline.com/index.php. Acesso em: 08 mai. 2016.

SANTOS, Rafael. **Introdução à programação orientada a objetos usando java**. Elsevier Brasil, 2013.

SPOON, TED. **Pugilatus - The history of boxing: Combat Sports and Self Defense.** Berforts Group Ltd, 2014.

TANENHAUS, M., et al. **"Miniature IMU/INS with optimally fused low drift MEMS gyro and accelerometers for applications in GPS-denied environments."** Position Location and Navigation Symposium (PLANS), 2012 IEEE/ION. IEEE, 2012.

THOMPSON M. J. , LI M. and HORSLEY D. A., **"Low power 3-axis Lorentz force navigation magnetometer"**, Micro Electro Mechanical Systems (MEMS), 2011 IEEE 24th International Conference on, Cancun, 2011, pp. 593-596.

TOURETZKY, David S.; POMERLEAU, Dean A. **What's hidden in the hidden layers.** Byte, August, p. 227-233, 1989.

WOODMAN, Oliver J. An introduction to inertial navigation. **University of Cambridge, Computer Laboratory, Tech. Rep. UCAMCL-TR-696**, v. 14, p. 15, 2007.

YU, Hao; WILAMOWSKI, Bogdan M. Levenberg-marquardt training. **Industrial Electronics Handbook**, v. 5, n. 12, p. 1, 2011.

APÊNDICE A – CÓDIGO NO MATLAB GERADOR DA MATRIZ DE TREINAMENTO DA RNA

```
clearvars -except network1 network2 network3 network4 network5...
 network6 a o neural neural1 neural2 neural3

ax = a(:,1); ay = a(:,2); az = a(:,3);
op = o(:,2); or = o(:,3);

zz=2;       %ordem de Z
z=zz+1;

target=0; %qual È o target para esses dados?
i=length(a);
vtarget=repmat(target,i,1); %monta o vetor target
vtarget=vtarget(z:end);

for p=1:z   %for loop cria a base FIR a partir de uma ordem z
        axz{p}=ax(z-p+1:end-p+1);
        ayz{p}=ay(z-p+1:end-p+1);
        azz{p}=az(z-p+1:end-p+1);

        opz{p}=op(z-p+1:end-p+1);
        orz{p}=or(z-p+1:end-p+1);
end


neural=cell2mat([axz ayz azz opz orz vtarget]); %contruÁ?o da...
%matriz de treinamento da RNA
neural=neural';

input=neural(1:end-1,:);    %matriz e entradas
vtarget=neural(end,:);      %vetor de saÌda target
```

APÊNDICE B – CÓDIGO NO MATLAB DE SIMULAÇÃO DA RNA TREINADA

```
function [Y,Xf,Af] = rede3(X,¬,¬)
%PATHNAME neural network simulation function.
%
% Generated by Neural Network Toolbox function genFunction,
% 09-Nov-2016 16:39:30.
% [Y] = pathname(X,¬,¬) takes these arguments:
%
%  X = 1xTS cell, 1 inputs over TS timesteps
%  Each X{1,ts} = 30xQ matrix, input #1 at timestep ts.
%
% and returns:
%  Y = 1xTS cell of 1 outputs over TS timesteps.
%  Each Y{1,ts} = 1xQ matrix, output #1 at timestep ts.
%
% where Q is number of samples (or series) and TS is the number .
%of timesteps
%#ok<*RPMT0>

% ===== NEURAL NETWORK CONSTANTS =====

% Input 1
x1_step1.xoffset = [vetor com 30 elementos que determinam o limite da...
 funÁ?o de ativaÁ?o de cada neurÙnio];
x1_step1.gain = [vetor com 30 elementos que determinam o fator da...
 funÁ?o digmoidal de cada neurÙnio];
x1_step1.ymin = -1;

% Layer 1
b1 = [vetor de 15 pesos];
IW1_1 = [matriz 15x30 (30 pesos para cada neurÙnio];
```

```
% Layer 2
b2 = [vetor de 15 pesos];
LW2_1 = [matriz15x15 de (15 pesos para cada neurÙnio) ];

% Layer 3
b3 = 0.20132864423337221;
LW3_2 = [vetor de 15 pesos do neurÙnio da camada de saÌda];

% Output 1
y1_step1.ymin = -1;
y1_step1.gain = 2;
y1_step1.xoffset = 0;

% ===== SIMULATION ========

% Format Input Arguments
isCellX = iscell(X);
if ¬isCellX, X = {X}; end;

% Dimensions
TS = size(X,2); % timesteps
if ¬isempty(X)
  Q = size(X{1},2); % samples/series
else
  Q = 0;
end

% Allocate Outputs
Y = cell(1,TS);

% Time loop
for ts=1:TS

    % Input 1
    Xp1 = mapminmax_apply(X{1,ts},x1_step1);

    % Layer 1
    a1 = tansig_apply(repmat(b1,1,Q) + IW1_1*Xp1);
```

```
    % Layer 2
    a2 = tansig_apply(repmat(b2,1,Q) + LW2_1*a1);

    % Layer 3
    a3 = tansig_apply(repmat(b3,1,Q) + LW3_2*a2);

    % Output 1
    Y{1,ts} = mapminmax_reverse(a3,y1_step1);
end

% Final Delay States
Xf = cell(1,0);
Af = cell(3,0);

% Format Output Arguments
if ¬isCellX, Y = cell2mat(Y); end
end

% ===== MODULE FUNCTIONS ========

% Map Minimum and Maximum Input Processing Function
function y = mapminmax_apply(x,settings)
  y = bsxfun(@minus,x,settings.xoffset);
  y = bsxfun(@times,y,settings.gain);
  y = bsxfun(@plus,y,settings.ymin);
end

% Sigmoid Symmetric Transfer Function
function a = tansig_apply(n,¬)
  a = 2 ./ (1 + exp(-2*n)) - 1;
end

% Map Minimum and Maximum Output Reverse-Processing Function
function x = mapminmax_reverse(y,settings)
  x = bsxfun(@minus,y,settings.ymin);
  x = bsxfun(@rdivide,x,settings.gain);
  x = bsxfun(@plus,x,settings.xoffset);
```

```
end
```

APÊNDICE C – CÓDIGO DO APLICATIVO *ANDROID*

```
package com.example.android.boxing;

import android.animation.ObjectAnimator;
import android.animation.ValueAnimator;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothManager;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.ServiceConnection;
import android.os.IBinder;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.view.animation.DecelerateInterpolator;
import android.widget.EditText;
import android.widget.ProgressBar;
import android.widget.TextView;

import com.mbientlab.metawear.MetaWearBoard;
import com.mbientlab.metawear.Route;
import com.mbientlab.metawear.android.BtleService;
import com.mbientlab.metawear.data.Acceleration;
import com.mbientlab.metawear.data.EulerAngles;
import com.mbientlab.metawear.module.Logging;
import com.mbientlab.metawear.module.SensorFusionBosch;
import com.opencsv.CSVWriter;
```

```
import java.io.FileWriter;
import java.io.IOException;
import java.util.Arrays;
import java.util.Calendar;

import Jama.Matrix;
import bolts.Continuation;
import bolts.Task;

public class MainActivity extends AppCompatActivity implements
ServiceConnection {

private BtleService.LocalBinder serviceBinder;
private MetaWearBoard board;
private MetaWearBoard board2;
private SensorFusionBosch sensorFusion;
private SensorFusionBosch sensorFusion2;
private Logging logging;
private Logging logging2;
private int i;
private int j;
private int numb;
private double[][] x;
private double[][] x2;
private String[] calendario;
private float[][] a;
private int i2;
private int j2;
private int numb2;
private float[][] a2;
public EditText filename;
public TextView estado;

ProgressBar mprogressBar;



@Override
```

```
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
filename = (EditText) findViewById(R.id.filename);
estado = (TextView) findViewById(R.id.estado);

// Bind the service when the activity is created
getApplicationContext().bindService(new Intent(this, BtleService.class),
      this, Context.BIND_AUTO_CREATE);
findViewById(R.id.csv).setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
CSVWriter writer = null;
try {
writer = new CSVWriter(new FileWriter(
"/sdcard/Boxinglogs/"+String.valueOf(filename.getText())+".csv"), ',');
String[] entries = {};
for (int numero = 0;numero<numb;numero++){
entries = (Arrays.toString(x[numero])).split("#");
writer.writeNext(entries);
}

writer.close();
} catch (IOException e) {
   e.printStackTrace();
}
}
});
findViewById(R.id.csv).setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
CSVWriter writer = null;
double[][] firD = baseFIR(x2);
double[][] saidaD = new double [firD.length][3];
for (int w = 0; w < firD.length; w++)
saidaD[w] = simulaRedeD(firD[w]);
double[][] firE = baseFIR(x);
double[][] saidaE = new double[firE.length][3];
```

```
for (int e = 0; e < firE.length; e++)
saidaE[e] = simulaRedeE(firE[e]);
Matrix direita = new Matrix(saidaD);
Matrix esquerda = new Matrix(saidaE);
double[][] golpesD = direita.transpose().getArrayCopy;
double[][] golpesE = esquerda.transpose().getArrayCopy();
int[] countDireita = new int[3];
int[] countEsquerda = new int[3];
for (int w = 0; w < 3; w++) {
countDireita[w] = contaPicos(golpesD[w], 10);
countEsquerda[w] = contaPicos(golpesE[w], 10);
}


try {
writer = new CSVWriter(new FileWriter("/sdcard/Boxinglogs/"
+String.valueOf(filename.getText())+"direita.csv"), ',');
String[] entries = {};
for (int numero = 0;numero < saidaD.length; numero++){
entries = (Arrays.toString(saidaD[numero])).split("#");
writer.writeNext(entries);
}

writer.close();
} catch (IOException e) {
e.printStackTrace();
}
writer = null;
try {
writer = new CSVWriter(new FileWriter("/sdcard/Boxinglogs/"
+String.valueOf(filename.getText())+"esquerda.csv"), ',');
String[] entries = {};
for (int numero = 0;numero < saidaE.length; numero++){
entries = (Arrays.toString(saidaE[numero])).split("#");
writer.writeNext(entries);
}

writer.close();
```

```
} catch (IOException e) {
e.printStackTrace();
}
writer = null;
try {
writer = new CSVWriter(new FileWriter("/sdcard/Boxinglogs/"
+String.valueOf(filename.getText())+"rawd.csv"), ',');
String[] entries = {};
for (int numero = 0;numero<numb2;numero++){
entries = (Arrays.toString(x2[numero])).split("#");
writer.writeNext(entries);
}

writer.close();
} catch (IOException e) {
e.printStackTrace();
}
writer = null;
try {
writer = new CSVWriter(new FileWriter("/sdcard/Boxinglogs/"
+String.valueOf(filename.getText())+"rawe.csv"), ',');
String[] entries = {};
for (int numero = 0;numero<numb;numero++){
entries = (Arrays.toString(x[numero])).split("#");
writer.writeNext(entries);
}

writer.close();
} catch (IOException e) {
e.printStackTrace();
}
writer = null;
try {
writer = new CSVWriter(new FileWriter("/sdcard/Boxinglogs/"
+String.valueOf(filename.getText())+"quantidade.csv"), ',');
String[] entries = {};
for (int numero = 0;numero<3;numero++){
entries = Double.toString(countDireita[numero]).concat
```

```
(Double.toString(countEsquerda[numero])).split("#");
writer.writeNext(entries);
}

writer.close();
} catch (IOException e) {
e.printStackTrace();
}
}
});
findViewById(R.id.start).setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
i=0;
j=0;
logging.start(true);
sensorFusion.eulerAngles().start();
sensorFusion.linearAcceleration().start();
sensorFusion.start();
i2=0;
j2=0;
logging2.start(true);
sensorFusion2.eulerAngles().start();
sensorFusion2.linearAcceleration().start();
sensorFusion2.start();
}
});
findViewById(R.id.stop).setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
logging.stop();
logging2.stop();
sensorFusion.eulerAngles().stop();
sensorFusion2.eulerAngles().stop();
sensorFusion.linearAcceleration().stop();
sensorFusion2.linearAcceleration().stop();
sensorFusion.stop();
sensorFusion2.stop();
```

```
mprogressBar = (ProgressBar) findViewById(R.id.progressBar2);
ObjectAnimator anim = ObjectAnimator.ofInt
(mprogressBar, "progress", 0, 100);
anim.setDuration(1500);
anim.setRepeatCount(ValueAnimator.INFINITE);
anim.setInterpolator(new DecelerateInterpolator());



estado.setText("Downloading");
logging.downloadAsync(100, new Logging.LogDownloadUpdateHandler() {
@Override
public void receivedUpdate(long nEntriesLeft, long totalEntries) {
if (i == 0) {
x = null;
numb = (int) (totalEntries/7);
x = new double[numb][6];
}
}
}, new Logging.LogDownloadErrorHandler() {
@Override
public void receivedError(Logging.DownloadError errorType,
 byte logId, Calendar timestamp, byte[] data) {
}
}).continueWithTask(new Continuation<Void, Task<Void>>()
{
@Override
public Task<Void> then(Task<Void> task) throws Exception
{
Log.i("MainActivity", "Download completed 1");
estado.setText("Download completed 1");
return null;
}
});

logging2.downloadAsync(100, new Logging.LogDownloadUpdateHandler() {
@Override
public void receivedUpdate(long nEntriesLeft, long totalEntries) {
```

```
if (i == 0) {
x2 = null;
numb2 = (int) (totalEntries/7);
x2 = new double[numb2][6];
}

}
}, new Logging.LogDownloadErrorHandler() {
@Override
public void receivedError(Logging.DownloadError errorType,
 byte logId, Calendar timestamp, byte[] data) {
}
}).continueWithTask(new Continuation<Void, Task<Void>>()
{
@Override
public Task<Void> then(Task<Void> task) throws Exception
{
Log.i("MainActivity", "Download completed 2");
estado.setText("Download completed 2");
return null;
}
});

}
});
}
@Override
public void onDestroy() {
super.onDestroy();

// Unbind the service when the activity is destroyed
getApplicationContext().unbindService(this);
}

@Override
public void onServiceConnected(ComponentName name, IBinder service) {
// Typecast the binder to the service's LocalBinder class
serviceBinder = (BtleService.LocalBinder) service;
```

```
Log.i("MainActivity", "Service Connected");

retrieveBoard();
}

@Override
public void onServiceDisconnected(ComponentName name) {

}

public void retrieveBoard() {
final BluetoothManager btManager=
(BluetoothManager) getSystemService(Context.BLUETOOTH_SERVICE);
final BluetoothDevice remoteDevice=
btManager.getAdapter().getRemoteDevice("DA:2C:D6:C4:CF:2E");
final BluetoothDevice remoteDevice2=
btManager.getAdapter().getRemoteDevice("CC:8E:87:B6:0F:80");

// Create a MetaWear board object for the Bluetooth Device
board= serviceBinder.getMetaWearBoard(remoteDevice);
board.connectAsync().onSuccessTask(
new Continuation<Void, Task<Route>>() {
@Override
public Task<Route> then(Task<Void> task) throws Exception{
Log.i("MainActivity","Connected to Metawear nosso");
/**if (task.isFaulted())
   Log.i("MainActivity", "Failed to connect to the board");
else*/

board2= serviceBinder.getMetaWearBoard(remoteDevice2);
board2.connectAsync().onSuccessTask(
new Continuation<Void, Task<Route>>() {
public Task<Route> then(Task<Void> task) throws Exception {
Log.i("MainActivity","Connected to Metawear deles");
board2.tearDown();
logging2 = board2.getModule(Logging.class);
logging2.clearEntries();
sensorFusion2= board2.getModule(SensorFusionBosch.class);
```

```
sensorFusion2.configure()
.mode(SensorFusionBosch.Mode.NDOF)
.accRange(SensorFusionBosch.AccRange.AR_16G)
.gyroRange(SensorFusionBosch.GyroRange.GR_2000DPS)
.commit();
//return null;
return sensorFusion2.eulerAngles().addRouteAsync(source2 ->
source2.limit(30).log((data, env) -> {
final EulerAngles angles2 = data.value(EulerAngles.class);
Log.i("MainActivity", "EULER deles" + angles2);
x2[i2][0] = angles2.yaw();
x2[i2][1] = angles2.pitch();
x2[i2][2] = angles2.roll();
i2++;
})).onSuccessTask(task2 -> sensorFusion2.linearAcceleration().
addRouteAsync(source2 ->
source2.limit(30).log((data, env) -> {
final Acceleration accel2 = data.value(Acceleration.class);
x2[j2][3] = accel2.x();
x2[j2][4] = accel2.y();
x2[j2][5] = accel2.z();
//x[j2][13] = data.formattedTimestamp();
j2++;
})));
}
}).continueWith(new Continuation<Route, Void>() {
@Override
public Void then(Task<Route> task) throws Exception {
if (task.isFaulted())

else
Log.i("MainActivity","App configured deles");
return null;
};
});

board.tearDown();
logging = board.getModule(Logging.class);
```

```
logging.clearEntries();
sensorFusion= board.getModule(SensorFusionBosch.class);
sensorFusion.configure()
.mode(SensorFusionBosch.Mode.NDOF)
.accRange(SensorFusionBosch.AccRange.AR_16G)
.gyroRange(SensorFusionBosch.GyroRange.GR_2000DPS)
.commit();
return // stream quaternion values from the board
sensorFusion.eulerAngles().addRouteAsync(source ->
source.limit(30).log((data, env) -> {

final EulerAngles angles = data.value(EulerAngles.class);
Log.i("MainActivity", "EULER nosso " + angles);
x[i][0] = angles.yaw();
x[i][1] = angles.pitch();
x[i][2] = angles.roll();
i++;
}))
.onSuccessTask(task1 -> sensorFusion.linearAcceleration()
.addRouteAsync(source ->
source.limit(30).log((data, env) -> {
final Acceleration accel = data.value(Acceleration.class);
x[j][3] = accel.x();
x[j][4] = accel.y();
x[j][5] = accel.z();
//x[j][6] = data.formattedTimestamp();
j++;
//Log.i("MainActiviyu", "Accel " + data.value(Acceleration.class) );
})));
}
}).continueWith(new Continuation<Route, Void>() {
@Override
public Void then(Task<Route> task) throws Exception {
if (task.isFaulted())
else
Log.i("MainActivity","App configured nosso");
return null;
};
```

```
});
}

public static int contaPicos(double[] output, int amostras){
int contaGolpe = 0;
int skip = 0;
for (int a = 0; a < output.length; a++){
if (output[a] > 0.8 && skip < 1) {
if ((output[a] + output[a + 1] + output[a + 2]) / 3 > 0.8 ||
 (output[a] + output[a + 1] + output[a + 2] + output[a + 3])
 / 4 > 0.8) {
contaGolpe++;
skip = amostras;
}
}
skip--;
}
return contaGolpe;
}

/************************SIMULA A REDE NEURAL*************/
public static double[] simulaRedeD(double[] x1){
double[] x1xoffset = {};
double[] x1gain = {};
double[] x1ymin = {};
double[] b1 };
double[][] IW1_1 = {};
double[] b2 = {};
double[][] LW2_1 = {};
double[] b3 = {};
double[][] LW3_1 = };
double[] y1ymin = {-1};
double[] y1gain = {2,2,2};
double[] y1xoffset = {0,0,0};
//Simulacao
//Input 1
double [] xp1 = mapminmax_apply(x1, x1xoffset, x1gain, x1ymin);
```

```
//Layer 1
double[] a1 = tansig_apply(b1, xp1, IW1_1);
//Layer 2
double[] a2 = tansig_apply(b2, a1, LW2_1);
//Layer 3
double[] a3 = tansig_apply(b3, a2, LW3_1);
//Output 1
double[] y1 = mapminmax_reverse(a3, y1ymin, y1gain, y1xoffset);

return y1;
   }

public static double[] mapminmax_apply(double[] x, double[] x1xoffset,
 double[] x1gain, double[] x1ymin){
for (int i = 0; i < x.length; i++){
x[i] = (x[i] - x1xoffset[i]) * x1gain[i] + x1ymin[0];
}
return x;
}
public static double[] tansig_apply(double[] m, double[] n,
 double[][] o){
Matrix a = new Matrix(m,1);
Matrix b = new Matrix(n,1);
Matrix c = new Matrix(o);
c = c.times(b.transpose());
a = a.transpose();
a = a.plus(c);
m = a.getColumnPackedCopy();
for (int i = 0; i < m.length; i++){
m[i] = 2 / (1 + Math.exp(-2 * m[i])) - 1;
}
return m;

}
public static double[] mapminmax_reverse(double[] x, double[] x1ymin
, double[] x1gain, double[] x1xoffset){
for (int i = 0; i < x.length; i++){
x[i] = (x[i] - x1ymin[0]) / x1gain[i] + x1xoffset[i];
```

```
}
return x;
}

public static double[] simulaRedeE(double[] x1){
double[] x1xoffset = {};
double[] x1gain = {};
double[] x1ymin = {};
double[] b1 ={};
double[][] IW1_1 = {};
double[] b2 = {};
double[][] LW2_1 = {};
double[] b3 = {};
double[][] LW3_1 = {};
double[] y1ymin = {};
double[] y1gain = {};
double[] y1xoffset = {};
//Simulacao
//Input 1
double [] xp1 = mapminmax_apply(x1, x1xoffset, x1gain, x1ymin);

//Layer 1
double[] a1 = tansig_apply(b1, xp1, IW1_1);
//Layer 2
double[] a2 = tansig_apply(b2, a1, LW2_1);
//Layer 3
double[] a3 = tansig_apply(b3, a2, LW3_1);
//Output 1
double[] y1 = mapminmax_reverse(a3, y1ymin, y1gain, y1xoffset);

return y1;
   }
}
```